Formal Languages, Automata and Codes

Oleg Gutik

Lecture 1

In this part of preliminary lectures, we give an account of some basic notions which will be used throughout our course "**Formal Languages, Automata and Codes**".

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i \colon i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i \colon i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i : i > 0, i \text{ is even}\} \quad (1)$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i \colon i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i : i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i : i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i : i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i : i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i : i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i \colon i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i \colon i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

A *set* is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written $x \notin S$. A set can be specified by enclosing some description of its elements in curly braces; for example, the set of integers $0, 1, 2$ is shown as

$$S = \{0, 1, 2\}.$$

Ellipses are used whenever the meaning is clear. Thus, $\{a, b, \ldots, z\}$ stands for all the lowercase letters of the English alphabet, while $\{2, 4, 6, \ldots\}$ denotes the set of all positive even integers. When the need arises, we use more explicit notation, in which we write

$$S = \{i\colon i > 0, i \text{ is even}\} \tag{1}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even," implying, of course, that i is an integer.

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\backslash$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\backslash$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\backslash$) defined as

$$
\begin{aligned}
S_1 \cup S_2 &= \{x : x \in S_1 \text{ or } x \in S_2\},\\
S_1 \cap S_2 &= \{x : x \in S_1 \text{ and } x \in S_2\},\\
S_1 - S_2 &= \{x : x \in S_1 \text{ and } x \notin S_2\}.
\end{aligned}
$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$
\begin{aligned}
S \cup \varnothing &= S - \varnothing = S,\\
S \cap \varnothing &= \varnothing,\\
\overline{\varnothing} &= U,\\
\overline{\overline{S}} &= S.
\end{aligned}
$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\backslash$) defined as

$$S_1 \cup S_2 = \{x \colon x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x \colon x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x \colon x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x \colon x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$\begin{aligned} S_1 \cup S_2 &= \{x\colon x \in S_1 \text{ or } x \in S_2\}, \\ S_1 \cap S_2 &= \{x\colon x \in S_1 \text{ and } x \in S_2\}, \\ S_1 - S_2 &= \{x\colon x \in S_1 \text{ and } x \notin S_2\}. \end{aligned}$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x\colon x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$\begin{aligned} S \cup \varnothing &= S - \varnothing = S, \\ S \cap \varnothing &= \varnothing, \\ \overline{\varnothing} &= U, \\ \overline{\overline{S}} &= S. \end{aligned}$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$\begin{aligned} S_1 \cup S_2 &= \{x\colon x \in S_1 \text{ or } x \in S_2\}, \\ S_1 \cap S_2 &= \{x\colon x \in S_1 \text{ and } x \in S_2\}, \\ S_1 - S_2 &= \{x\colon x \in S_1 \text{ and } x \notin S_2\}. \end{aligned}$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x\colon x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$\begin{aligned} S \cup \varnothing &= S - \varnothing = S, \\ S \cap \varnothing &= \varnothing, \\ \overline{\varnothing} &= U, \\ \overline{\overline{S}} &= S. \end{aligned}$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\backslash$) defined as

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$S_1 \cup S_2 = \{x\colon x \in S_1 \text{ or } x \in S_2\},$$
$$S_1 \cap S_2 = \{x\colon x \in S_1 \text{ and } x \in S_2\},$$
$$S_1 - S_2 = \{x\colon x \in S_1 \text{ and } x \notin S_2\}.$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x\colon x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$S \cup \varnothing = S - \varnothing = S,$$
$$S \cap \varnothing = \varnothing,$$
$$\overline{\varnothing} = U,$$
$$\overline{\overline{S}} = S.$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference* ($-$ or $\setminus$) defined as

$$\begin{aligned} S_1 \cup S_2 &= \{x : x \in S_1 \text{ or } x \in S_2\}, \\ S_1 \cap S_2 &= \{x : x \in S_1 \text{ and } x \in S_2\}, \\ S_1 - S_2 &= \{x : x \in S_1 \text{ and } x \notin S_2\}. \end{aligned}$$

Another basic operation is *complementation*. The *complement* of a set S, denoted by $\overline{S}$, consists of all elements which are not in S. To make this meaningful, we need to know what the universal set U of all possible elements is. If U is specified, then

$$\overline{S} = \{x : x \in U, x \notin S\}.$$

The set with no elements, called the *empty set* or the *null set*, is denoted by $\varnothing$. From the definition of a set, it is obvious that

$$\begin{aligned} S \cup \varnothing &= S - \varnothing = S, \\ S \cap \varnothing &= \varnothing, \\ \overline{\varnothing} &= U, \\ \overline{\overline{S}} &= S. \end{aligned}$$

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.

A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.

A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.

A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.

A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.

A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.

A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$

$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \tag{2}$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \tag{3}$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

The following useful identities, known as *DeMorgan's laws*,

$$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2, \quad (2)$$
$$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2. \quad (3)$$

are needed on several occasions.
A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S.$$

If the sets S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \varnothing$, then the sets S_1 and S_2 are said to be *disjoint*.
A set is said to be *finite* if it contains a finite number of elements; otherwise it is infinite. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\} .$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\} .$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

A given set normally has many subsets. The set of all subsets of a set S is called the *powerset* of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If $S = \{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\} .$$

Here $|S| = 3$ and $\left|2^S\right| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2, 2), (2, 3), (2, 5), (2, 6), (4, 2), (4, 3), (4, 5), (4, 6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) : x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) : x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\} .$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\} .$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\} .$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) : x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2,2), (2,3), (2,5), (2,6), (4,2), (4,3), (4,5), (4,6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) : x_i \in S_i\}.$$

In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the *Cartesian product* of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \colon x \in S_1, y \in S_2\}.$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2, 2), (2, 3), (2, 5), (2, 6), (4, 2), (4, 3), (4, 5), (4, 6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair $(4, 2)$ is in $S_1 \times S_2$, but $(2, 4)$ is not.

This notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) \colon x_i \in S_i\}.$$

A set can be divided by separating it into a number of subsets. Suppose that $S_1, S_2, \ldots, S_n$ are subsets of a given set S and that the following holds:

1) the subsets $S_1, S_2, \ldots, S_n$ are mutually disjoint;
2) $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3) every of S_i is nonempty.

Then $S_1, S_2, \ldots, S_n$ is called a *partition* of S.

A set can be divided by separating it into a number of subsets. Suppose that $S_1, S_2, \ldots, S_n$ are subsets of a given set S and that the following holds:

1) the subsets $S_1, S_2, \ldots, S_n$ are mutually disjoint;
2) $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3) every of S_i is nonempty.

Then $S_1, S_2, \ldots, S_n$ is called a *partition* of S.

A set can be divided by separating it into a number of subsets. Suppose that $S_1, S_2, \ldots, S_n$ are subsets of a given set S and that the following holds:

1) the subsets $S_1, S_2, \ldots, S_n$ are mutually disjoint;
2) $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3) every of S_i is nonempty.

Then $S_1, S_2, \ldots, S_n$ is called a *partition* of S.

A set can be divided by separating it into a number of subsets. Suppose that $S_1, S_2, \ldots, S_n$ are subsets of a given set S and that the following holds:

1) the subsets $S_1, S_2, \ldots, S_n$ are mutually disjoint;
2) $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3) every of S_i is nonempty.

Then $S_1, S_2, \ldots, S_n$ is called a *partition* of S.

A set can be divided by separating it into a number of subsets. Suppose that $S_1, S_2, \ldots, S_n$ are subsets of a given set S and that the following holds:

1) the subsets $S_1, S_2, \ldots, S_n$ are mutually disjoint;
2) $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3) every of S_i is nonempty.

Then $S_1, S_2, \ldots, S_n$ is called a *partition* of S.

A set can be divided by separating it into a number of subsets. Suppose that $S_1, S_2, \ldots, S_n$ are subsets of a given set S and that the following holds:

1) the subsets $S_1, S_2, \ldots, S_n$ are mutually disjoint;
2) $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3) every of S_i is nonempty.

Then $S_1, S_2, \ldots, S_n$ is called a *partition* of S.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f: S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

A *function* is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the *domain* of f, and the second set is its *range*. We write

$$f: S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function*.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f *has order at most* g. We write this as

$$f(n) = O(g(n)).$$

If

$$|f(n)| \geq c|g(n)|,$$

then f has *order at least* g, for which we use

$$f(n) = \Omega(g(n)).$$

Finally, if there exist constants c_1 and c_2 such that

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,$$

f and g have the *same order of magnitude*, expressed as

$$f(n) = \Theta(g(n)).$$

In this order-of-magnitude notation, we ignore multiplicative constants and lower-order terms that become negligible as n increases.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$\begin{aligned} f(n) &= 2n^2 + 3n, \\ g(n) &= n^3, \\ h(n) &= 10n^2 + 100. \end{aligned}$$

Then

$$\begin{aligned} f(n) &= O(g(n)), \\ g(n) &= \Omega(h(n)), \\ f(n) &= \Theta(h(n)). \end{aligned}$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol $=$ should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later lectures.

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Some functions can be represented by a set of pairs

$$\{(x_1, y_1), (x_2, y_2), \ldots\},$$

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a *relation*. Relations are more general than functions: In a function each element of the domain has exactly one associated element in the range; in a relation there may be several such elements in the range.

One kind of relation is that of *equivalence*, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \mod 3 = y \mod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into equivalence classes. Each equivalence class contains all and only equivalent elements.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \dots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \dots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \dots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \dots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \ldots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \ldots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

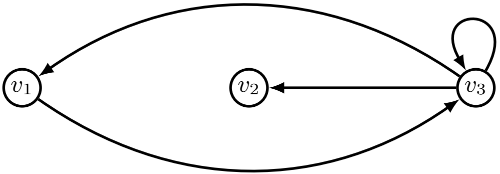
$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.

A *graph* is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \dots, v_n\}$ of *vertices* and the set $E = \{e_1, e_2, \dots, e_m\}$ of *edges*. Each edge is a pair of vertices from V, for instance,

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an *outgoing edge* for v_j and an *incoming edge* for v_k. Such a construct is actually a directed graph (digraph), since we associate a direction (from v_j to v_k) with each edge. Graphs may be labeled, a label being a name or other information associated with parts of the graph. Both vertices and edges may be labeled. Graphs are conveniently visualized by diagrams in which the vertices are represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in the following Figure.



A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

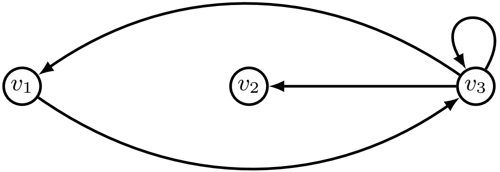
A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.

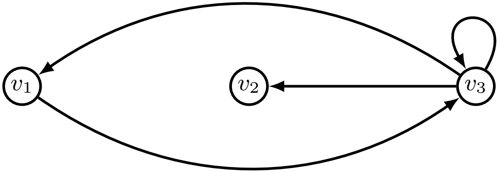
The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



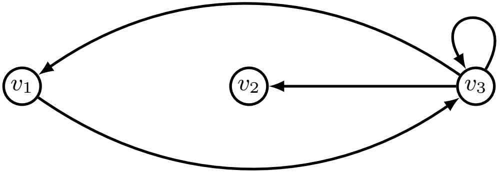
The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



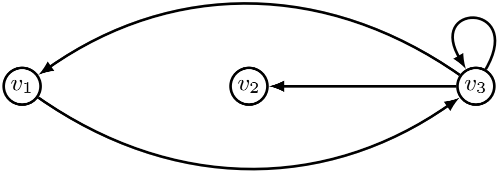
The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



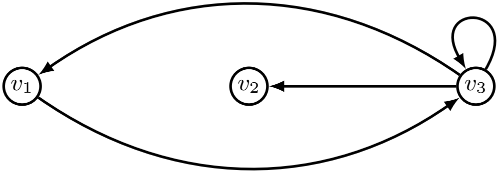
The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



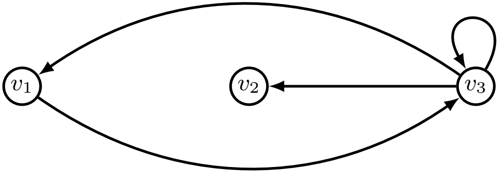
The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



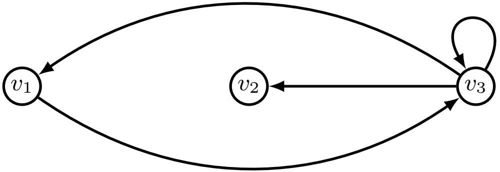
The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a *walk* from v_i to v_n. The *length of a walk* is the total number of edges traversed in going from the initial vertex to the final one. A walk in which no edge is repeated is said to be a *path*; a path is *simple* if no vertex is repeated. A walk from v_i to itself with no repeated edges is called a *cycle with base* v_i. If no vertices other than the base are repeated in a cycle, then it is said to be *simple*. In the Figure, $(v_1, v_3), (v_3, v_2)$ is a simple path from v_1 to v_2.



The sequence of edges $(v_1, v_3), (v_3, v_3), (v_3, v_1)$ is a cycle, but not a simple one. If the edges of a graph are labeled, we can talk about the label of a walk. This label is the sequence of edge labels encountered when the path is traversed. Finally, an edge from a vertex to itself is called a *loop*. In the Figure, there is a loop on vertex v_3.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l), \ldots$. At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l), \ldots$. At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, $\ldots$. At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, $\ldots$. At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

On several occasions, we will refer to an algorithm for finding all simple paths between two given vertices (or all simple cycles based on a vertex). If we do not concern ourselves with efficiency, we can use the following obvious method. Starting from the given vertex, say v_i, list all outgoing edges $(v_i, v_k), (v_i, v_l)$, ... At this point, we have all paths of length one starting at v_i. For all vertices $v_k, v_l, \ldots$ so reached, we list all outgoing edges as long as they do not lead to any vertex already used in the path we are constructing. After we do this, we will have all simple paths of length two originating at v_i. We continue this until all possibilities are accounted for. Since there are only a finite number of vertices, we will eventually list all simple paths beginning at v_i. From these we select those ending at the desired vertex.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

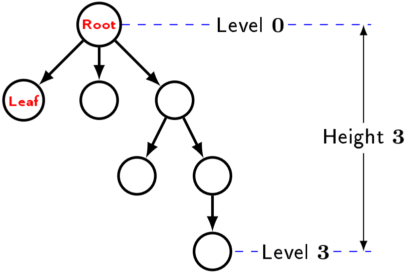
Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.

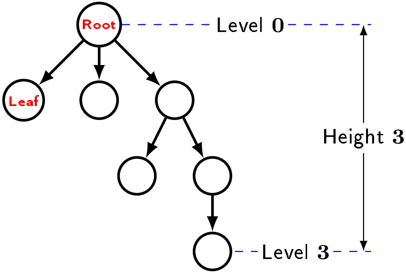
At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.



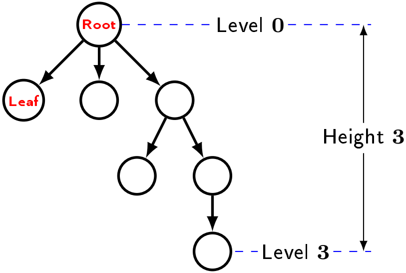
At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.



At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

Trees are a particular type of graphs. A *tree* is a directed graph that has no cycles and that has one distinct vertex, called the *root*, such that there is exactly one path from the root to every other vertex. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the *leaves of the tree*. If there is an edge from v_i to v_j, then v_i is said to be the *parent* of v_j, and v_j the *child* of v_i. The *level associated with each vertex* is the number of edges in the path from the root to the vertex. The *height* of the tree is the largest level number of any vertex. These terms are illustrated in the Figure.



At times, we want to associate an ordering with the nodes at each level; in such cases we talk about *ordered trees*.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \dots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \dots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \dots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

An important requirement for reading this text is the ability to follow proofs. In mathematical arguments, we employ the accepted rules of deductive reasoning, and many proofs are simply a sequence of such steps. Two special proof techniques are used so frequently that it is appropriate to review them briefly. These are *proof by induction* and *proof by contradiction*.

Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose we have a sequence of statements $P_1, P_2, \ldots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

1. For some $k \geq 1$, we know that $P_1, P_2, \ldots, P_k$ are true.
2. The problem is such that for any $n \geq k$, the truths of $P_1, P_2, \ldots, P_n$ imply the truth of P_{n+1}.

In a proof by induction, we argue as follows: From **Condition 1** we know that the first k statements are true. Then **Condition 2** tells us that P_{k+1} also must be true. But now that we know that the first $k+1$ statements are true, we can apply **Condition 2** again to claim that P_{k+2} must be true, and so on. We need not explicitly continue this argument, because the pattern is clear. The chain of reasoning can be extended to any statement. Therefore, every statement is true.

The starting statements $P_1, P_2, \ldots, P_k$ are called the basis of the induction. The step connecting P_n with P_{n+1} is called the *inductive step*. The inductive step is generally made easier by the inductive assumption that $P_1, P_2, \ldots, P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}. In a formal inductive argument, we show all three parts explicitly.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n + 1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n + 1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n + 1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n + 1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \leq 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \leq 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n + 1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n + 1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n + 1) \leq 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n + 1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Example 1.5

A binary tree is a tree in which no parent can have more than two children. Prove that a binary tree of height n has at most 2^n leaves.

Proof: If we denote the maximum number of leaves of a binary tree of height n by $l(n)$, then we want to show that $l(n) \le 2^n$.

Basis: Clearly $l(0) = 1 = 2^0$, because a tree of height 0 can have no nodes other than the root, that is, it has at most one leaf.

Inductive Assumption:

$$l(i) \le 2^i, \quad \text{for } i = 0, 1, \ldots, n. \tag{4}$$

Inductive Step: To get a binary tree of height $n+1$ from one of height n, we can create, at most, two leaves in place of each previous one. Therefore,

$$l(n+1) = 2l(n).$$

Now, using the inductive assumption, we get that

$$l(n+1) \le 2 \cdot 2^n = 2^{n+1}.$$

Thus, if our claim is true for n, it must also be true for $n+1$. Since n can be any number, the statement must be true for all n. ■

Here we introduce the symbol ■ that is used in our course of lectures to denote the end of a proof.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, ..., $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, ..., $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, ..., $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, ..., $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, ..., $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, ..., $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

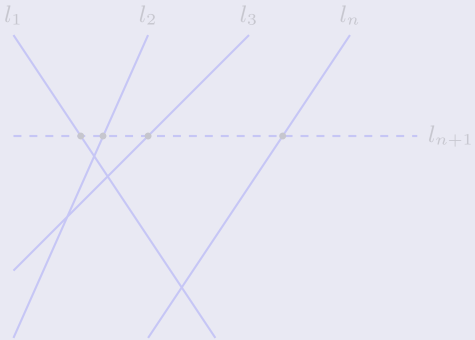
Inductive reasoning can be difficult to grasp. It helps to notice the close connection between induction and recursion in programming. For example, the recursive definition of a function $f(n)$, where n is any positive integer, often has two parts. One involves the definition of $f(n+1)$ in terms of $f(n)$, $f(n-1)$, $\ldots$, $f(1)$. This corresponds to the inductive step. The second part is the "escape" from the recursion, which is accomplished by defining $f(1)$, $f(2)$, $\ldots$, $f(k)$ nonrecursively. This corresponds to the basis of induction. As in induction, recursion allows us to draw conclusions about all instances of the problem, given only a few starting values and using the recursive nature of the problem.

Sometimes, a problem looks difficult until we look at it in just the right way. Often looking at it recursively simplifies matters greatly.

Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

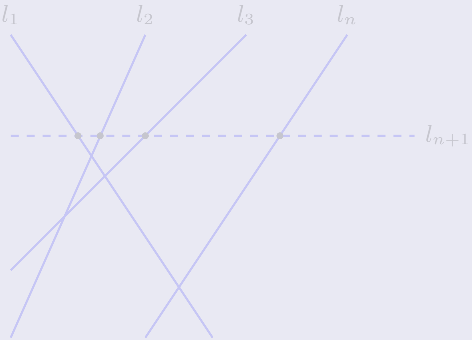
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

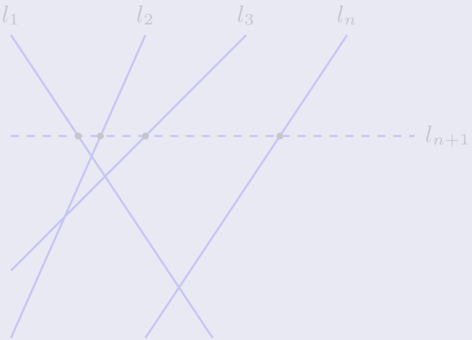
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

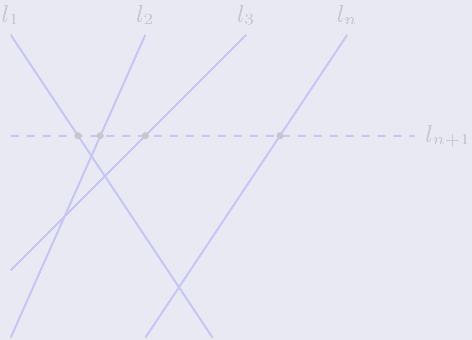
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

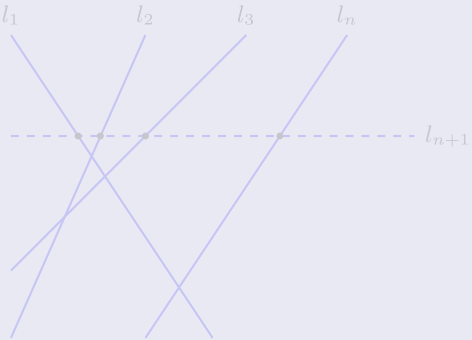
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

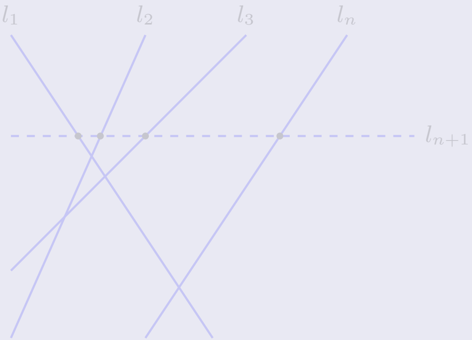
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

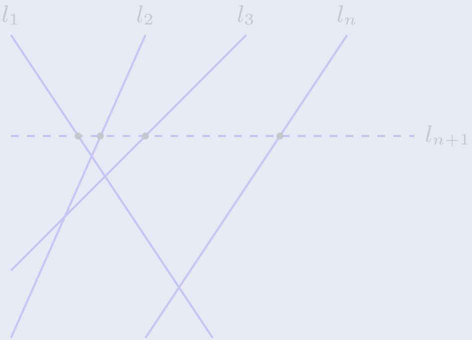
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

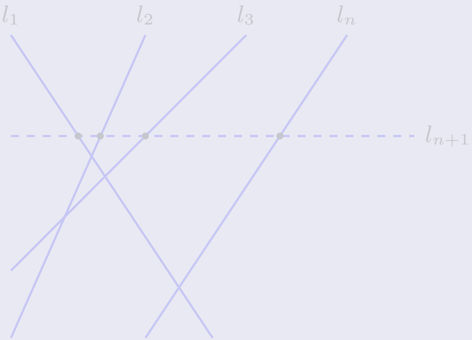
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

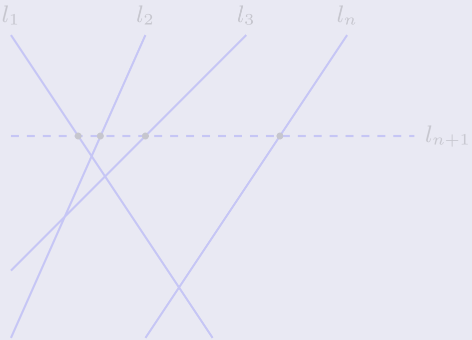
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

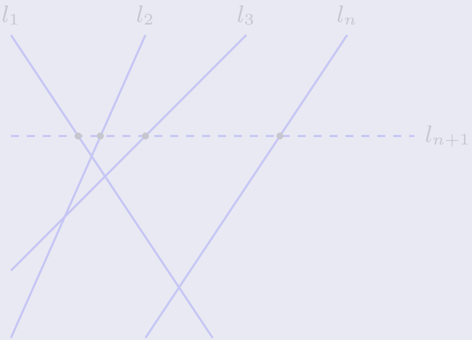
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

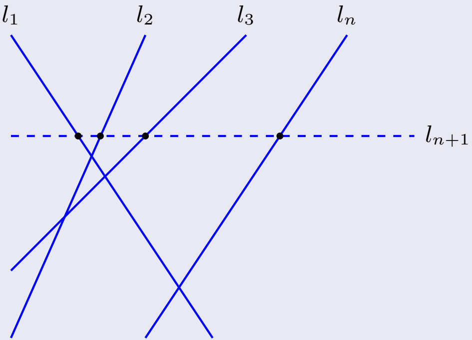
Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



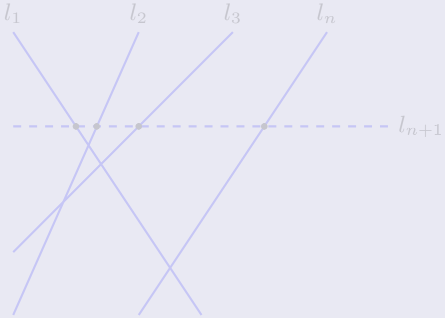
Example 1.6

A set $l_1, l_2, \ldots, l_n$ of mutually intersecting straight lines divides the plane into a number of separated regions. A single line divides the plane into two parts, two lines generate four regions, three lines make seven regions, and so on. This is easily checked visually for up to three lines, but as the number of lines increases it becomes difficult to spot a pattern. Let us try to solve this problem recursively.

Look at the Figure to see what happens if we add a new line l_{n+1} to existing n lines.



Example 1.6

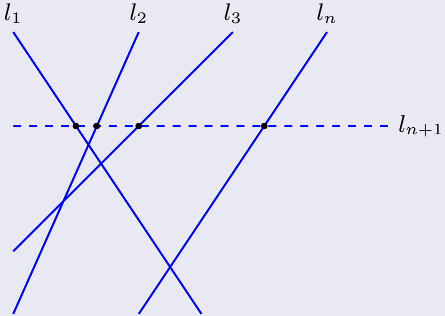


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

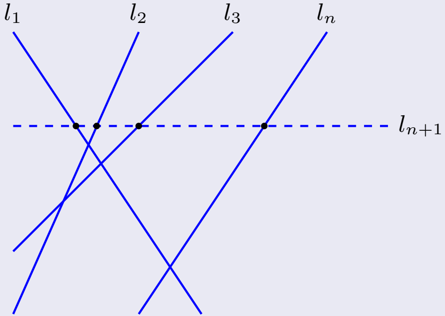


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

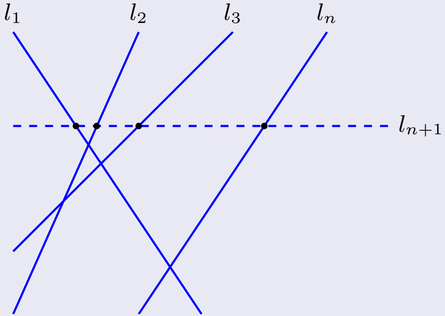


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

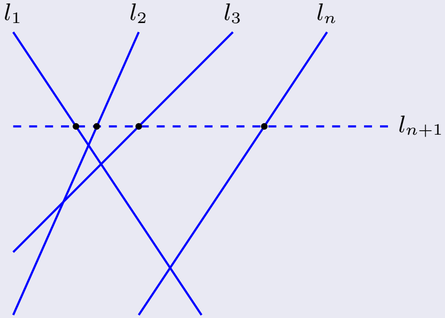


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

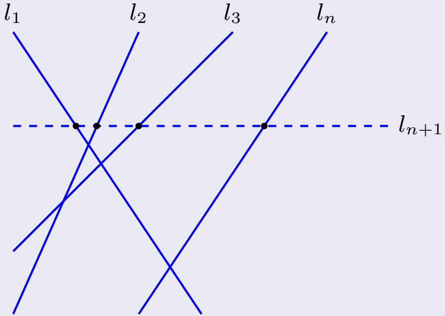


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

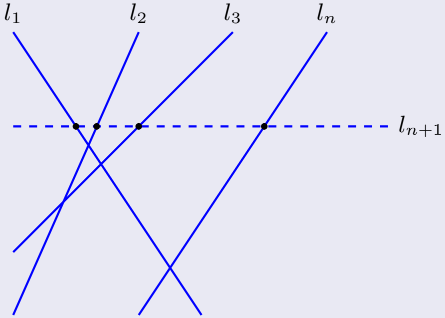


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

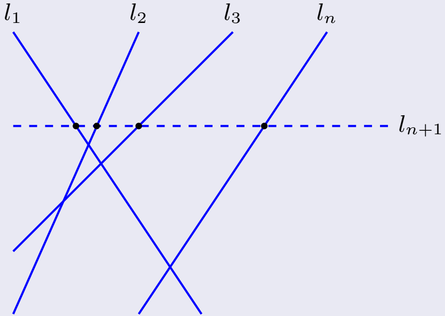


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

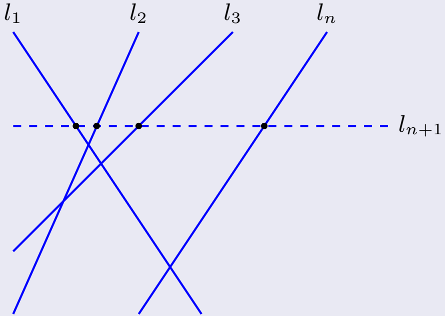


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

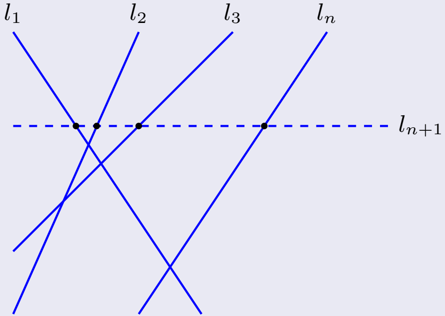


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

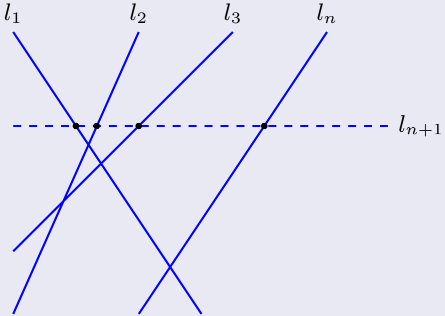


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

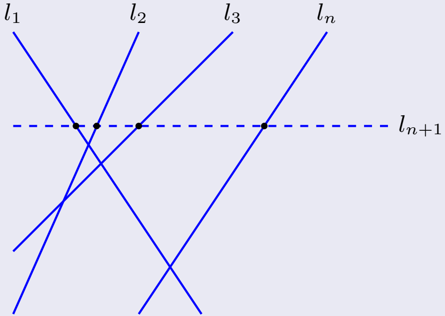


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

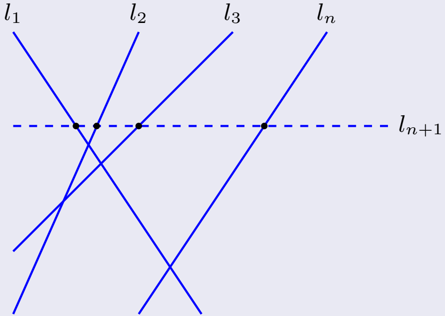


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

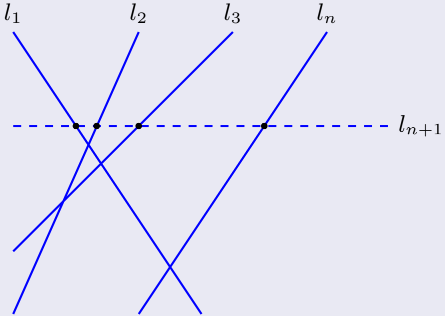


The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6



The region to the left of l_1 is divided into two new regions, so is the region to the left of l_2, and so on until we get to the last line. At the last line, the region to the right of l_n is also divided. Each of the n intersections then generates one new region, with one extra at the end. So, if we let $A(n)$ denote the number of regions generated by n lines, we see that

$$A(n+1) = A(n) + n + 1, \qquad n = 1, 2, \ldots,$$

with $A(1) = 2$. From this simple recursion we then calculate $A(2) = 4$, $A(3) = 7$, $A(4) = 11$, and so on.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of **Example 1.5**.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of **Example 1.5**.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of **Example 1.5**.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of **Example 1.5**.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of Example 1.5.

Example 1.6

To get a formula for $A(n)$ and to show that it is correct, we use induction. If we conjecture that

$$A(n) = \frac{n(n+1)}{2} + 1,$$

then

$$\begin{aligned} A(n+1) &= \frac{n(n+1)}{2} + 1 + n + 1 = \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} + 1 = \\ &= \frac{(n+1)(n+2)}{2} + 1 \end{aligned}$$

justifies the inductive step. The basis is easily checked, completing the argument.

In this example we have been a little less formal in identifying the basis, inductive assumption, and inductive step, but they are there and are essential. To keep our subsequent discussions from becoming too formal, we shall generally prefer the style of this second example. However, if you have difficulty in following or constructing a proof, go back to the more explicit form of **Example 1.5**.

Proof by contradiction is another powerful technique that often works when everything else fails. Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true. The following is a classic and elegant example.

Proof by contradiction is another powerful technique that often works when everything else fails. Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true. The following is a classic and elegant example.

Proof by contradiction is another powerful technique that often works when everything else fails. Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true. The following is a classic and elegant example.

Proof by contradiction is another powerful technique that often works when everything else fails. Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true. The following is a classic and elegant example.

Proof by contradiction is another powerful technique that often works when everything else fails. Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true. The following is a classic and elegant example.

Proof by contradiction is another powerful technique that often works when everything else fails. Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true. The following is a classic and elegant example.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.
As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.
As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

Example 1.7

A *rational number* is a number that can be expressed as the ratio of two integers n and m so that n and m have no a common factor. A real number that is not rational is said to be *irrational*. Show that $\sqrt{2}$ is irrational.

As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m}, \tag{5}$$

where n and m are integers without a common factor. Rearranging (5), we have that

$$2m^2 = n^2.$$

Therefore, n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2,$$

and

$$m^2 = 2k^2.$$

Therefore, m is even. But this contradicts our assumption that n and m have no common factors. Thus, m and n in (5) cannot exist and $\sqrt{2}$ is not a rational number.

This example exhibits the essence of a proof by contradiction. By making a certain assumption we are led to a contradiction of the assumption or some known fact. If all steps in our argument are logically sound, we must conclude that our initial assumption was false.

This example exhibits the essence of a proof by contradiction. By making a certain assumption we are led to a contradiction of the assumption or some known fact. If all steps in our argument are logically sound, we must conclude that our initial assumption was false.

This example exhibits the essence of a proof by contradiction. By making a certain assumption we are led to a contradiction of the assumption or some known fact. If all steps in our argument are logically sound, we must conclude that our initial assumption was false.

Thank You for attention!