Обробка зображень і мультимедіа

Олег Гутік

Лекція 4: Статистичні методи стиснення зображень

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під "статистичними властивостями" зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, "ta", "he", "ca", зустрічаються дуже часто, інші, наприклад, "xa", "hz", "qe", — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під "статистичними властивостями" зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, "ta", "he", "ca", зустрічаються дуже часто, інші, наприклад, "xa", "hz", "qe", — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень

Статистичні методи компресії використовують статистичні властивості даних, що стискуються, і надають всім символам коди зі змінною довжиною. Під “статистичними властивостями” зазвичай розуміється ймовірність (або, що те саме, частота появи) кожного символу в потоці даних, проте цей термін може мати інше, більш складне значення. Пару послідовних символів називатимемо **біграмою**. Довгі спостереження показали, що в типовому англійському тексті деякі біграми, наприклад, “ta”, “he”, “ca”, зустрічаються дуже часто, інші, наприклад, “xa”, “hz”, “qe”, — рідко. Тому розумний статистичний метод може надавати коди змінної довжини багатьом біграмам (і навіть триграмам), а не тільки індивідуальним символам.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P \log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P \log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

Основи теорії інформації було закладено Клодом Шеноном 1948 року у лабораторії Белла. Ця теорія виявилася справжнім сюрпризом, оскільки всі звикли сприймати інформацію лише якісно. Для наших цілей стиснення даних нам необхідно освоїти лише одне теоретико-інформаційне поняття, а саме ентропію. Під *ентропією* символу a, що має ймовірність P, мається на увазі кількість інформації, що міститься в a, яка дорівнює $-P\log_2 P$. Наприклад, якщо ймовірність P_a символу a дорівнює 0.5, то його ентропія дорівнює $-P_a \log_2 P_a = 0.5$.

Якщо символи деякого алфавіту із символами від a_1 до a_n мають ймовірності від P_1 до P_n, то *ентропія всього алфавіту* дорівнює сумі $\sum_{i=1}^{n} -P_i \log_2 P_i$. Якщо задано рядок символів цього алфавіту, то для нього ентропія визначається аналогічно.

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

Статистичні методи стиснення зображень. Ентропія

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

За допомогою поняття ентропії теорія інформації показує, як обчислювати ймовірності рядків символів алфавіту, і передбачає їхнє найкраще стиснення, тобто, найменше, в середньому, число біт, необхідне для зображення цього рядка символів. Продемонструємо це на простому прикладі. Для послідовності символів "ABCDE" з ймовірностями 0.5, 0.2, 0.1, 0.1 та 0.1, відповідно, ймовірність рядка "AAAAABBCDE" дорівнює $P = 0.5^5 \cdot 0.2^2 \cdot 0.1^1 = 1.25 \cdot 10^{-6}$. Логарифм з основою 2 цього числа $\log_2 P = -19.6096$. Тоді найменша в середньому кількість необхідних біт для кодування цього рядка дорівнює $-\lceil \log_2 P \rceil$, тобто 20. Кодер, що досягає цього стиснення, називається *ентропійним кодером*.

Приклад

Проаналізуємо ентропію алфавіту, що складається всього з двох символів a_1 та a_2 з ймовірностями P_1 та P_2. Оскільки $P_1 + P_2 = 1$, то ентропія цього алфавіту виражається числом

$$-P_1 \log_2 P_1 - (1 - P_1) \log_2 (1 - P_1).$$

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

У табл. наведено різні значення величин P_1 та P_2 разом з відповідною ентропією. Коли $P_1 = P_2$, необхідний принаймні один біт кодування кожного символу. Це означає, що ентропія досягла свого максимуму, надмірність дорівнює нулю, і дані неможливо стиснути. Однак, якщо ймовірності символів сильно відрізняються, мінімальна кількість необхідних біт на символ знижується. Ми, швидше за все, не зможемо безпосередньо пред'явити метод стиснення, який використовує 0.08 біт на символ, але точно знаємо, що з $P_1 = 0.99$ такий алгоритм теоретично можливий.

P_1	P_2	Ентропія
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Перше правило побудови кодів із змінною довжиною цілком очевидне. Короткі коди слід присвоювати символам, що часто зустрічаються, а довгі — рідко зустрічаються. Проте є інша проблема. Ці коди треба призначати так, щоб їх можна було декодувати однозначно, а не двозначно. Невеликий приклад проілюструє це.

Розглянемо чотири символи a_1, a_2, a_3 та a_4. Якщо вони появляються в послідовності даних з однаковою ймовірністю ($= 0.25$ кожний), то ми їм просто надамо чотири двобітні коди 00, 01, 10 і 11. Всі ймовірності рівні, і тому коди змінної довжини не стиснуть ці дані. Для кожного символу з коротким кодом знайдеться символ із довгим кодом і середня кількість бітів на символ буде не менше 2. Надмірність даних з рівноймовірними символами дорівнює нулю, і рядок таких символів неможливо стиснути за допомогою змінної довжини (або будь-яким іншим методом).

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

Припустимо, що ці чотири символи з'являються з різними ймовірностями, зазначеними в табл.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

тобто символ a_1 появляється у рядку даних у середньому майже в половині випадків, символи a_2 і a_3 мають рівні ймовірності, а символ a_4 виникає вкрай рідко. У цьому випадку є надмірність, яку можна видалити за допомогою змінних кодів і стиснути дані так, що потрібно менше 2 біт на символ. Насправді теорія інформації говорить нам про те, що найменша кількість необхідних біт на символ в середньому дорівнює 1.57, тобто, ентропії цієї множини символів.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1 a_3 a_2 a_1 a_3 a_3 a_4 a_2 a_1 a_1 a_2 a_2 a_1 a_1 a_3 a_1 a_1 a_2 a_3 a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

У табл. запропонований код Code 1, що присвоює символу, який найчастіше зустрічається найкоротший код.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Якщо закодувати дані символи за допомогою коду Code 1, то середня кількість біт на символ дорівнює

$$1 \cdot 0.49 + 2 \cdot 0.25 + 3 \cdot 0.25 + 3 \cdot 0.01 = 1.77.$$

Це дуже близько до теоретичного мінімуму. Розглянемо послідовність із 20 символів

$$a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1,$$

в якій чотири символи з'являються приблизно з зазначеними частотами. Цьому рядку буде відповідати кодовий рядок коду Code 1 довжини 37 біт

$$1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1,$$

який для зручності розділений вертикальними рисочками. Нам знадобилося 37 бітів, щоб закодувати 20 символів, тобто в середньому 1.85 біт/символ, що не надто далеко від обчисленої вище середньої величини.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Однак, якщо ми тепер спробуємо декодувати цю двійкову послідовність, то негайно виявимо, що Code 1 зовсім не придатний. Перший біт послідовності дорівнює 1, а тому першим символом може бути тільки a_1, оскільки ніякий інший код таблиці для Code 1 не починається з 1. Наступний біт дорівнює 0, але коди для a_2, a_3 і a_4 всі починаються з 0, а тому декодер повинен читати наступний біт. Він дорівнює 1, однак коди для a_2 та a_3 обидва мають на початку 01. Декодер не знає, як йому вчинити. Чи то декодувати рядок як $1|010|01\ldots$, тобто, $a_1a_3a_2\cdots$, чи як $1|01|001\cdots$, тобто $a_1a_2a_4\ldots$. Причому зауважимо, що подальші біти послідовності вже не допоможуть виправити становище. Код Code 1 є двозначним[1]. На відміну від нього код Code 2 з табл. дає при декодуванні завжди однозначний результат.

Символ	Імовірність	Code 1	Code 2
a_1	0.48	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

[1] Насправді Code 1 не є кодом в розумінні теорії кодів.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Code 2 має одну важливу властивість, яка робить його кращим, ніж Code 1, і вона називається *префіксною властивістю*. Цю властивість можна сформулювати так: якщо деяка послідовність бітів обрана як код якогось символу, то жоден код іншого символу не повинен мати на початку цю послідовність (не може бути префіксом, тобто, приставкою). Якщо рядок "1" вже вибрано як цілий код для a_1, то жоден інший код не може починатися з 1 (тобто всі вони повинні починатися символом 0). Якщо рядок "01" є кодом для a_2, то інші коди не повинні починатися з 01. Ось чому коди для a_3 і a_4 мають починатися з 00. Природно, вони будуть "000" і "001".

Отже, вибираючи множину кодів змінної довжини, необхідно дотримуватися двох принципів:

(1) слід призначати більш короткі коди символам які найчастіше зустрічаються, і

(2) коди повинні бути префіксними, тобто задовольняти префіксні властивості.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Дотримуючись цих принципів, можна побудувати короткі, однозначні коди, що декодуються, але не обов'язково найкращі (тобто, найкоротші) коди. На додаток до цих принципів необхідний алгоритм, який завжди породжує множину найкоротших кодів (які у середньому мають найменшу довжину). Вихідними даними цього алгоритму мають бути частоти (або ймовірності) символів алфавіту. На щастя, такий простий алгоритм існує. Він був придуманий Давидом Гаффманом і названий його ім'ям.

Слід зазначити, що статистичні методи компресії використовують коди змінної довжини при кодуванні індивідуальних символів. Чудовим прикладом є арифметичні коди, про які буде розказано в наступних лекціях.

Статистичні методи стиснення зображень. Декодування

Перед тим як описувати статистичні методи стиснення даних, важливо зрозуміти спосіб взаємодії кодера та декодера (компресора та декомпресора). Припустимо, що певний файл (з текстом, зображенням чи ще чимось) був стиснутий за допомогою кодів змінної довжини (префіксних кодів). Для того щоб зробити декомпресію, декодер повинен знати префіксний код кожного символу. Цю проблему можна розв'язати трьома способами.

Статистичні методи стиснення зображень. Декодування

Перед тим як описувати статистичні методи стиснення даних, важливо зрозуміти спосіб взаємодії кодера та декодера (компресора та декомпресора). Припустимо, що певний файл (з текстом, зображенням чи ще чимось) був стиснутий за допомогою кодів змінної довжини (префіксних кодів). Для того щоб зробити декомпресію, декодер повинен знати префіксний код кожного символу. Цю проблему можна розв'язати трьома способами.

Статистичні методи стиснення зображень. Декодування

Перед тим як описувати статистичні методи стиснення даних, важливо зрозуміти спосіб взаємодії кодера та декодера (компресора та декомпресора). Припустимо, що певний файл (з текстом, зображенням чи ще чимось) був стиснутий за допомогою кодів змінної довжини (префіксних кодів). Для того щоб зробити декомпресію, декодер повинен знати префіксний код кожного символу. Цю проблему можна розв'язати трьома способами.

Статистичні методи стиснення зображень. Декодування

Перед тим як описувати статистичні методи стиснення даних, важливо зрозуміти спосіб взаємодії кодера та декодера (компресора та декомпресора). Припустимо, що певний файл (з текстом, зображенням чи ще чимось) був стиснутий за допомогою кодів змінної довжини (префіксних кодів). Для того щоб зробити декомпресію, декодер повинен знати префіксний код кожного символу. Цю проблему можна розв'язати трьома способами.

Статистичні методи стиснення зображень. Декодування

Перед тим як описувати статистичні методи стиснення даних, важливо зрозуміти спосіб взаємодії кодера та декодера (компресора та декомпресора). Припустимо, що певний файл (з текстом, зображенням чи ще чимось) був стиснутий за допомогою кодів змінної довжини (префіксних кодів). Для того щоб зробити декомпресію, декодер повинен знати префіксний код кожного символу. Цю проблему можна розв'язати трьома способами.

Статистичні методи стиснення зображень. Декодування

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

Статистичні методи стиснення зображень. Декодування

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

1. Множину префіксних кодів один раз вибрано та використовується всіма кодерами та декодерами. Такий метод використовується у факсимільному зв'язку. Творці стандарту стиснення для факс-машин обрали вісім "еталонних" документів, проаналізували їхні статистичні властивості та взяли їх за основу при відборі відповідного префіксного коду, представленого в деякій табл. Кажучи технічною мовою, еталонні документи були обрані для навчання та тренування алгоритму. Навчання алгоритму є найпростішим підходом до статистичного стиснення, і його якість залежить від того, наскільки реальні стисливі файли схожі на обрані для тренування та навчання зразки.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

2. Кодер робить свою роботу у два проходи. На першому проході він читає файл, що стискається, і збирає необхідні статистичні відомості. На другому проході відбувається власне стиснення. У перерві між проходами декодер на основі зібраної інформації створює найкращий префіксний код саме для цього файлу. Такий метод дає чудові результати зі стиснення, але він, зазвичай, занадто повільний завдяки практичному використанню. Крім того, у нього є ще один істотний недолік. Необхідно додавати таблицю побудованих префіксних кодів до стисненого файлу, щоб її знав декодер. Це погіршує загальну продуктивність алгоритму. Такий підхід у статистичному стиску прийнято називати *напівадаптивною компресією*.

Статистичні методи стиснення зображень. Декодування

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

3. Адаптивне стиснення застосовується як кодером, і декодером. Кодер починає працювати, не знаючи статистичних властивостей об'єкта стиснення. Тому перша частина даних стискається не оптимальним чином, однак, у міру стиснення та збору статистики, кодер покращує використовуваний префіксний код, що призводить до поліпшення компресії. Алгоритм повинен бути розроблений таким чином, щоб декодер міг повторити кожен крок кодера, зібрати ту ж статистику та покращити префіксний код точно таким же способом. Приклад адаптивного стиснення буде розглянуто у наступних лекціях.

Дякую за увагу!