Обробка зображень і мультимедіа

Олег Гутік

Лекція 5: Статистичні методи стиснення зображень

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Кодування Гаффмана є простим алгоритмом для побудови кодів змінної довжини, що мають мінімальну середню довжину. Цей дуже популярний алгоритм є основою багатьох комп'ютерних програм стиснення текстової та графічної інформації. Деякі з них використовують безпосередньо алгоритм Гаффмана, а інші беруть його як один із ступенів багаторівневого процесу стиснення. Метод Гаффмана робить ідеальне стиснення (тобто, стискає дані до їхньої ентропії), якщо ймовірності символів точно дорівнюють негативним ступеням числа 2. Алгоритм починає будувати кодове дерево знизу вгору, потім ковзає вниз по дереву, щоб побудувати кожен індивідуальний код ліворуч (від наймолодшого біта до найстаршого). Починаючи з робіт Д. Гаффмана 1952 року, цей алгоритм був предметом багатьох досліджень.[1]

[1] Відомо, що найкращий код змінної довжини можна іноді отримати без цього алгоритму.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

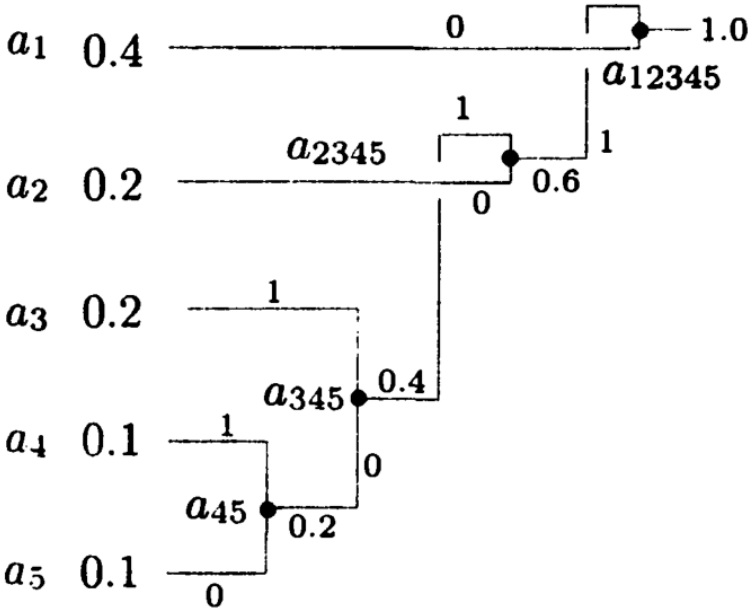
Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

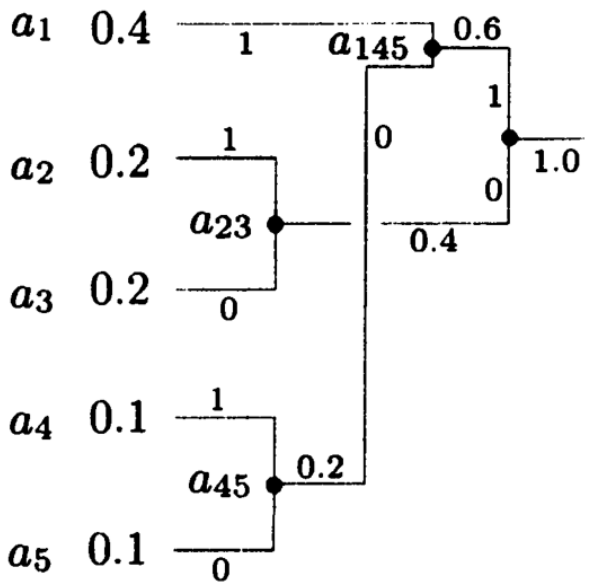
Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Алгоритм починається складання списку символів алфавіту в порядку зменшення їхніх ймовірностей. Потім від кореня будується дерево, листям якого є ці символи. Це робиться по кроках, причому на кожному кроці вибираються два символи з найменшими ймовірностями, додаються догори часткового дерева, видаляються зі списку і замінюються допоміжним символом, що представляє ці два символи. Допоміжному символу приписується ймовірність, що дорівнює сумі ймовірностей, вибраних на цьому кроці символів. Коли список скорочується до одного символу, що представляє весь алфавіт, дерево оголошується побудованим. Завершується алгоритм спуском по дереву та побудовою кодів усіх символів.

Найкраще проілюструвати цей алгоритм на простому прикладі. Є п'ять символів із ймовірностями, заданими на рис.(а).

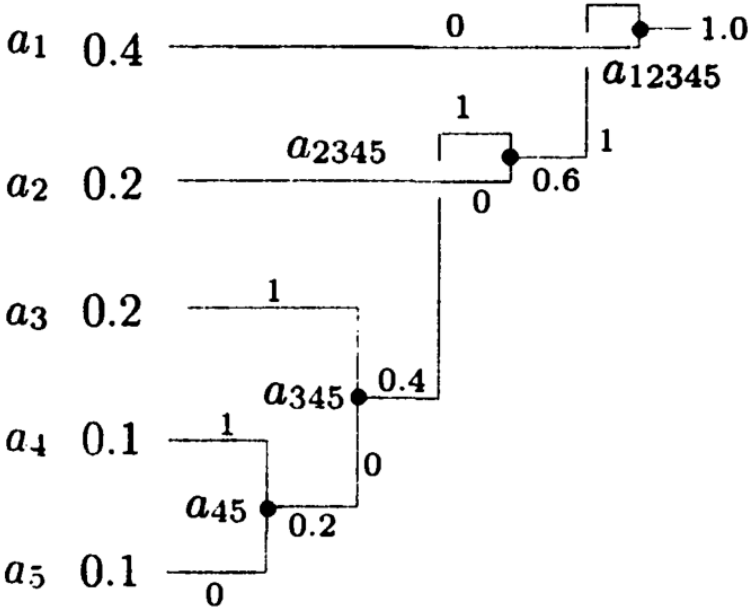


(a)

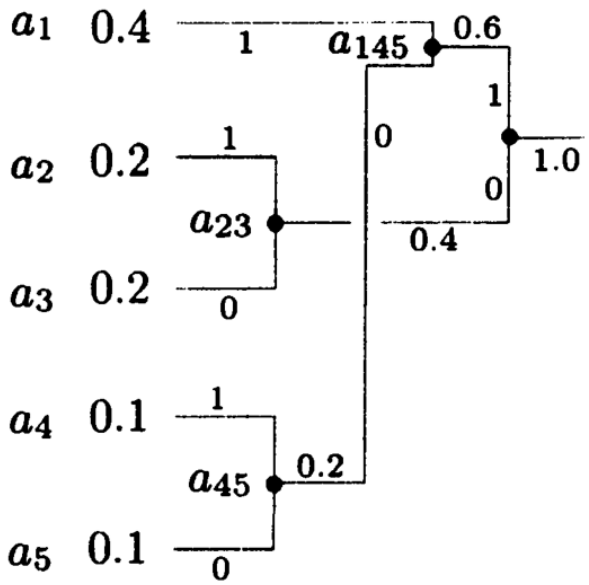


(b)

Найкраще проілюструвати цей алгоритм на простому прикладі. Є п'ять символів із ймовірностями, заданими на рис.(а).

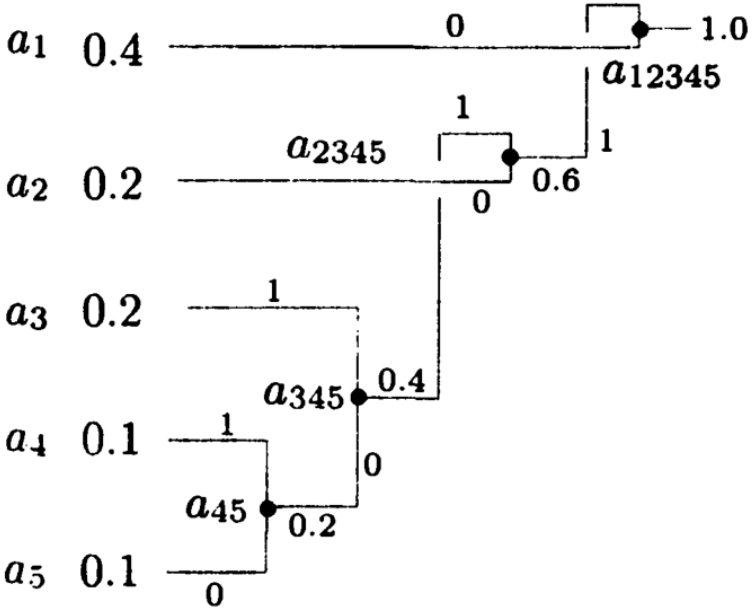


(a)

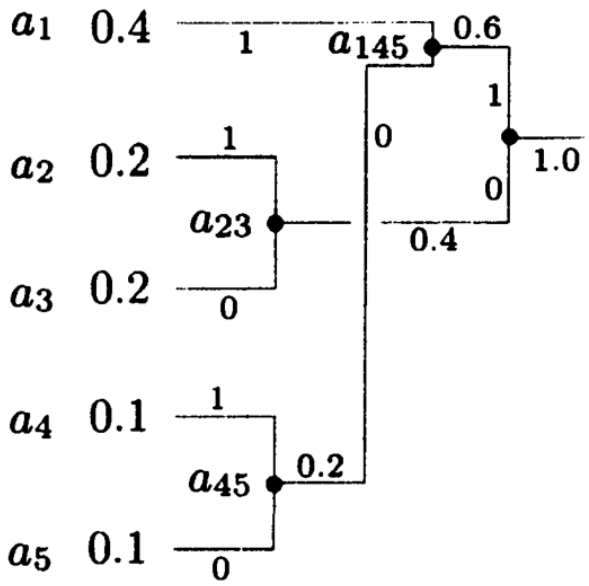


(b)

Найкраще проілюструвати цей алгоритм на простому прикладі. Є п'ять символів із ймовірностями, заданими на рис.(а).

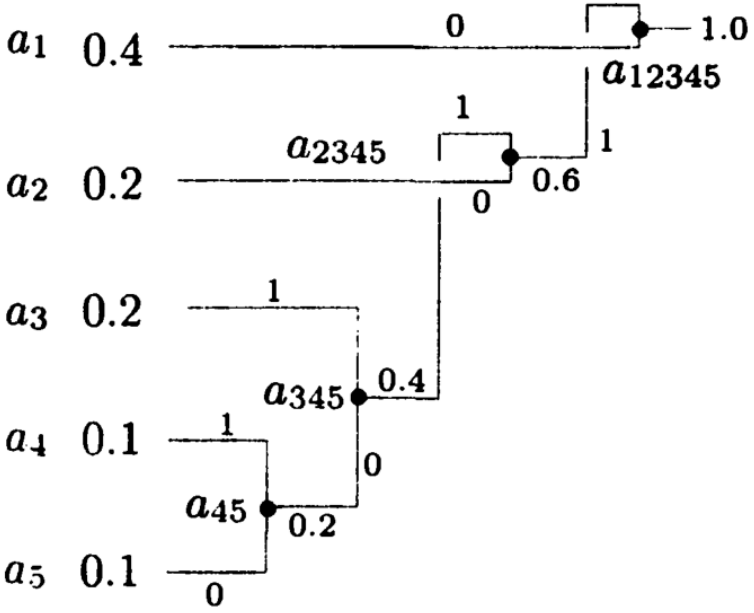


(a)

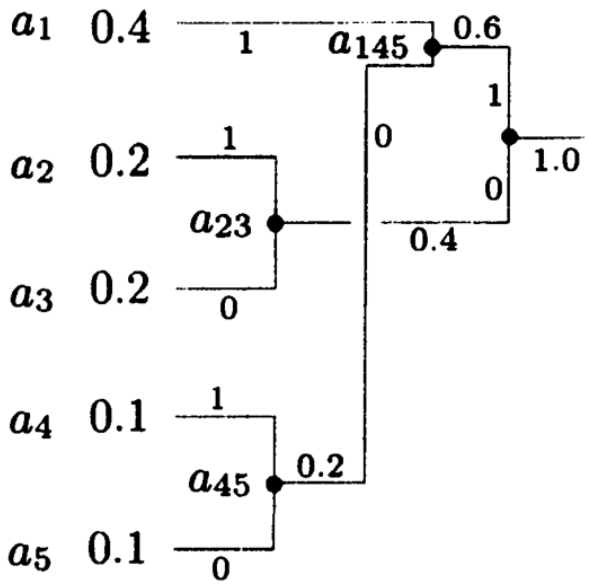


(b)

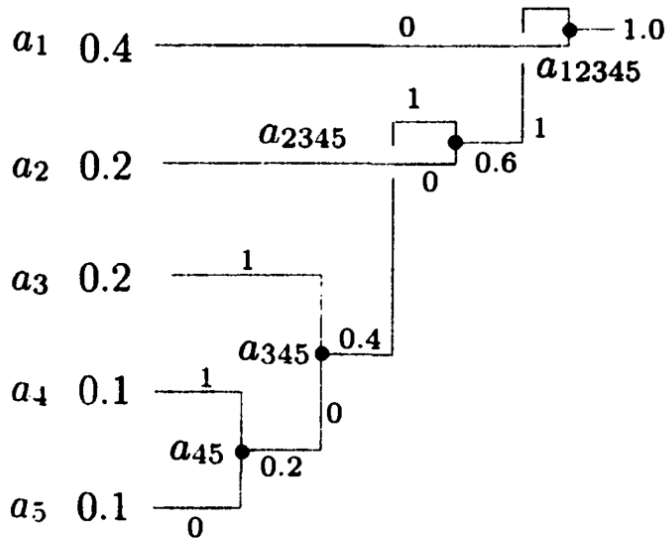
Найкраще проілюструвати цей алгоритм на простому прикладі. Є п'ять символів із ймовірностями, заданими на рис.(а).



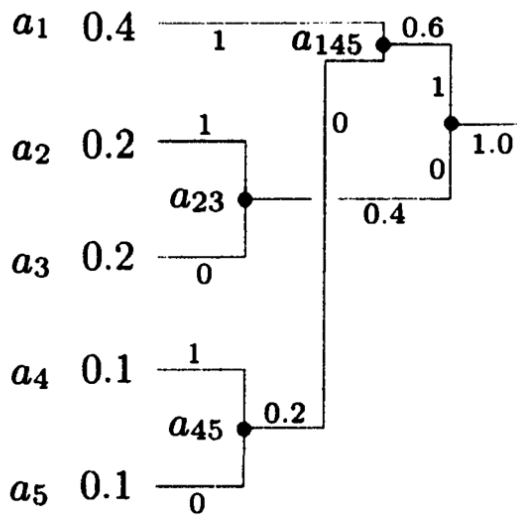
(a)



(b)



(a)



(b)

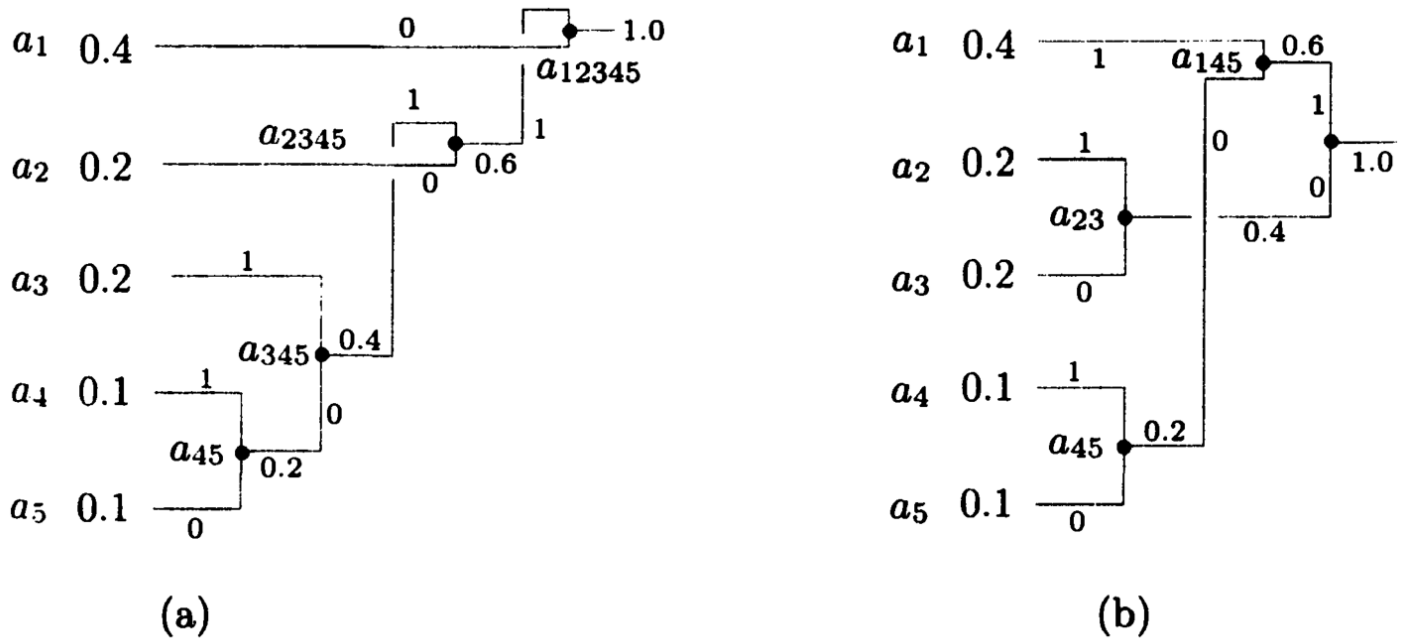
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



(a) (b)

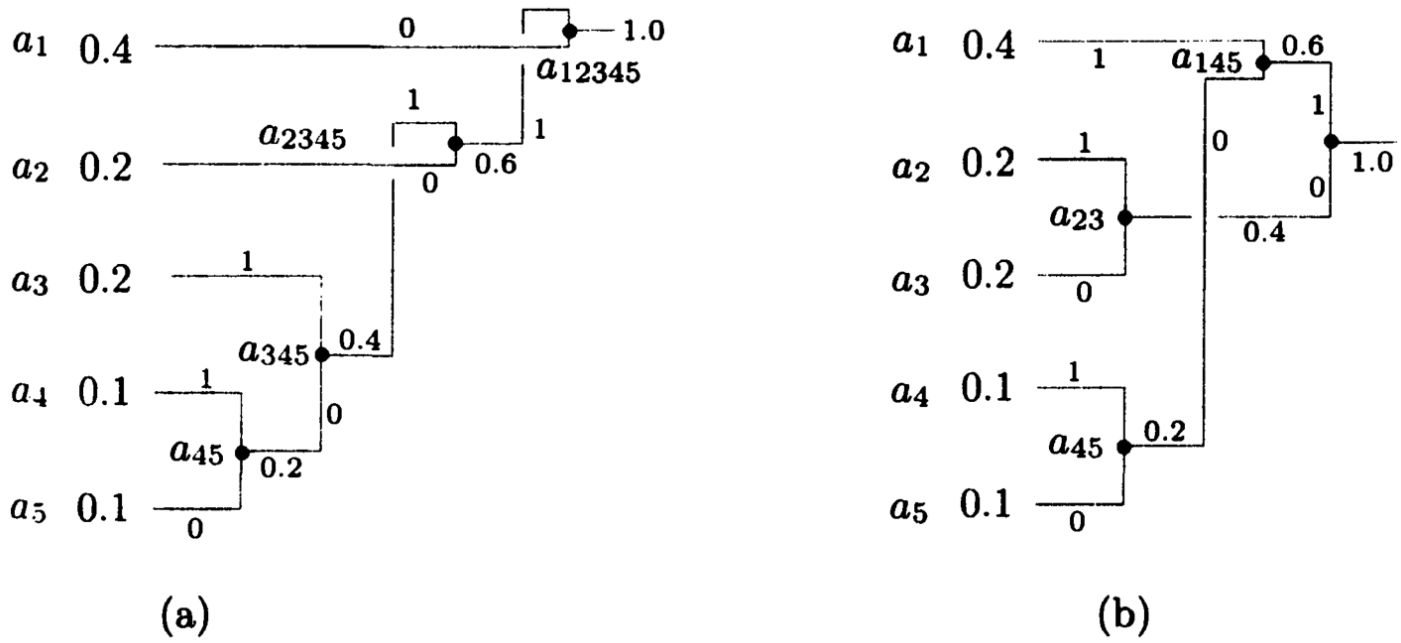
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



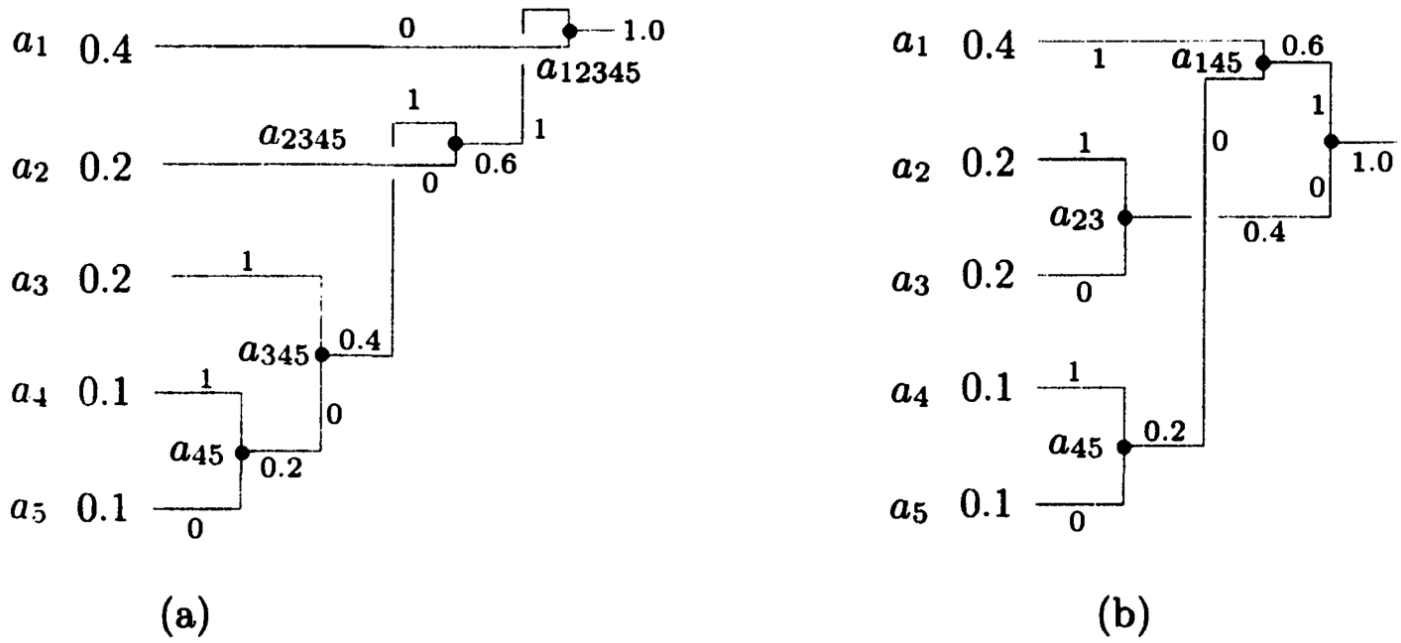
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



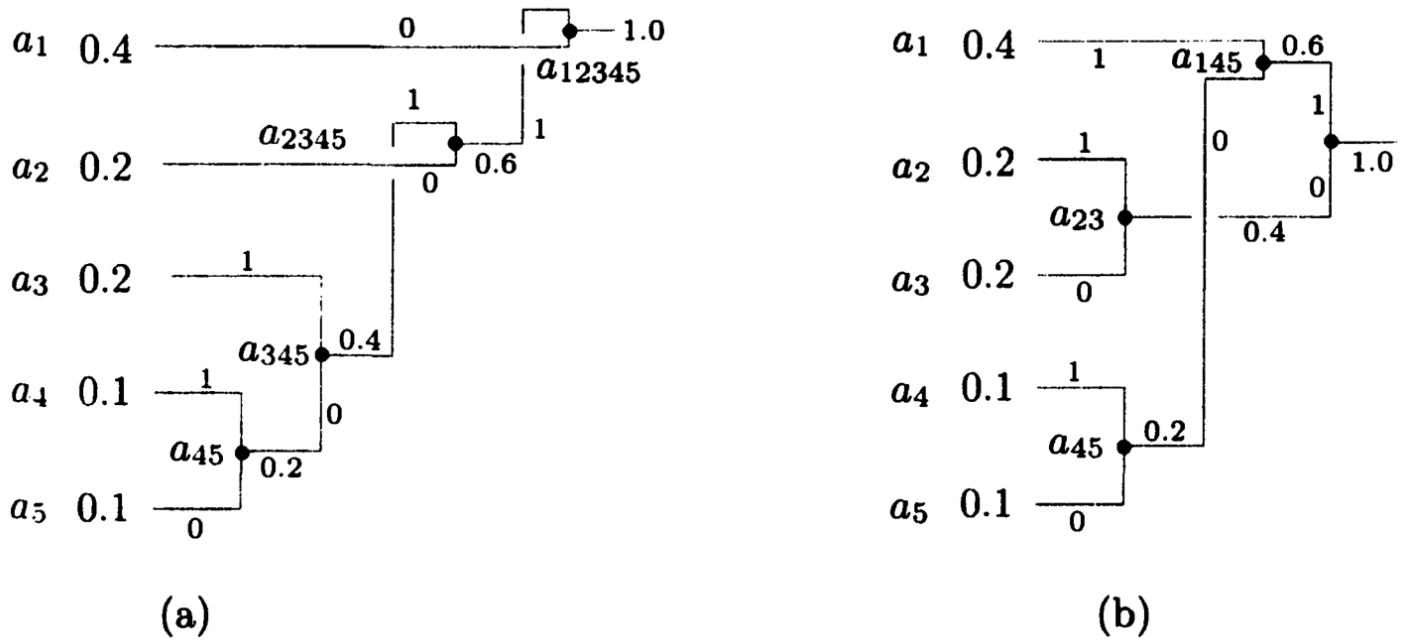
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



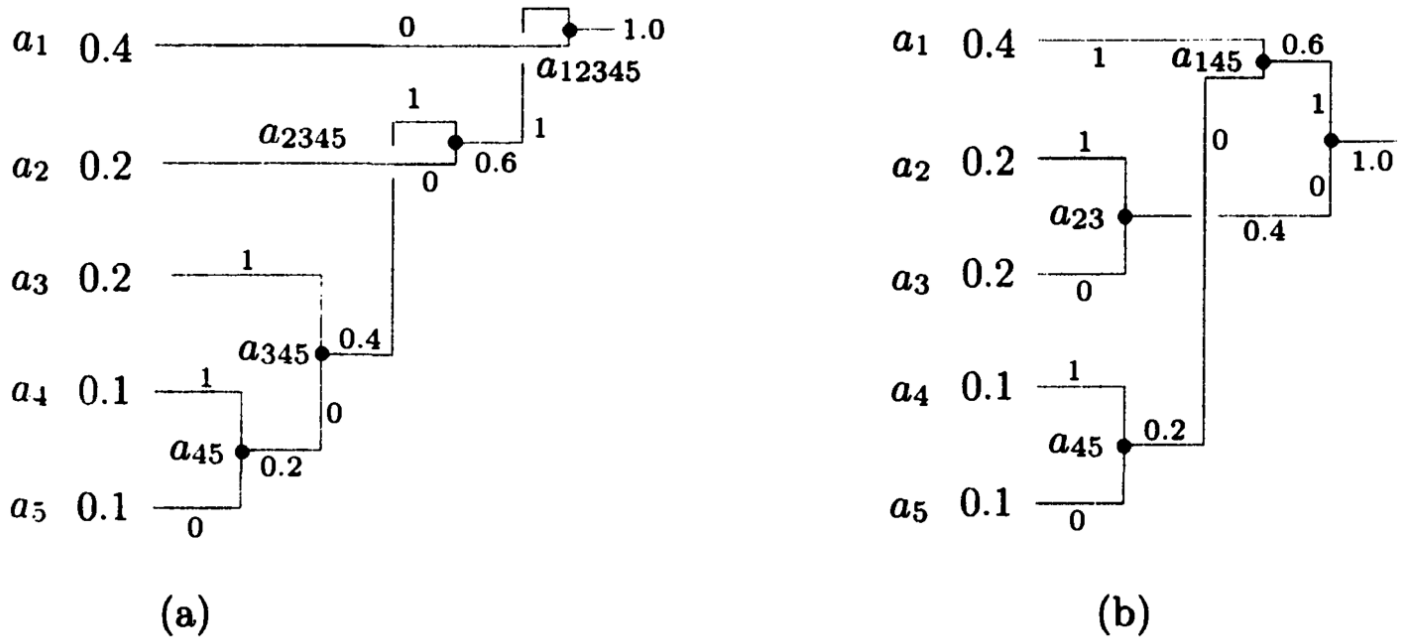
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



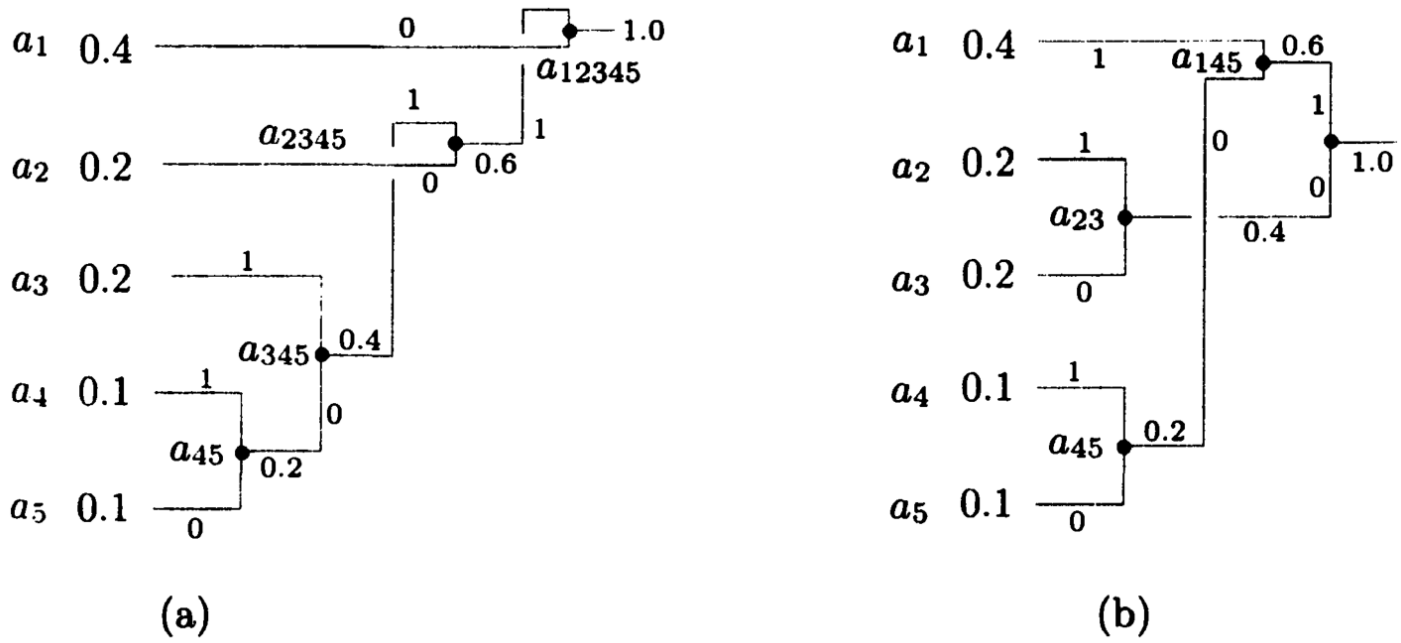
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



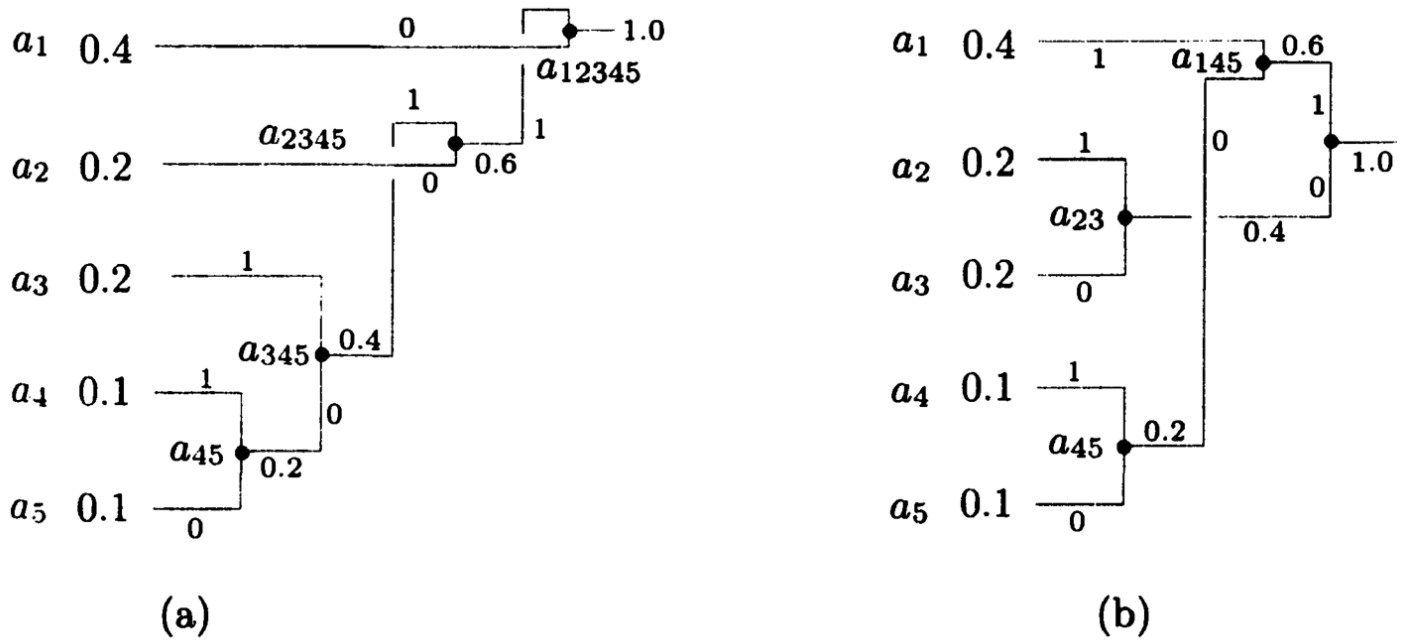
Символи об'єднуються у пари у такому порядку:

(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.



Символи об'єднуються у пари у такому порядку:

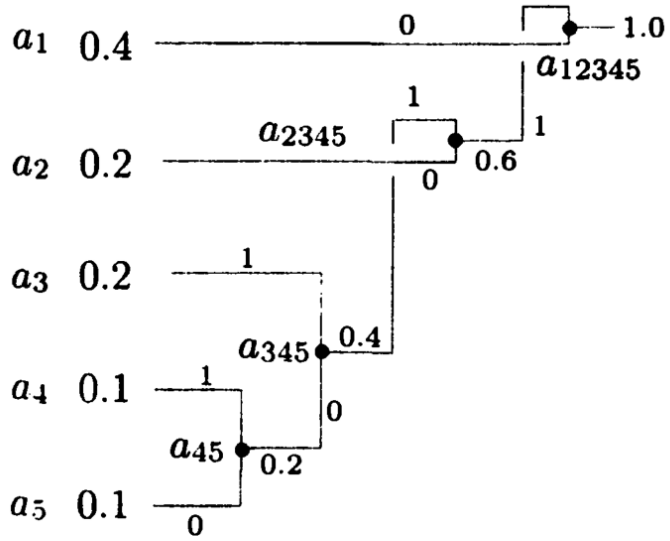
(1) символ a_4 об'єднується з символом a_3 і обидва замінюються комбінованим символом a_{45} з ймовірністю 0.2;

(2) залишилося чотири символи, a_1 з ймовірністю 0.4, а також a_2, a_3 і a_{45} з ймовірностями по 0.2. Довільно вибираємо a_3 і a_{45}, об'єднуємо їх і замінюємо допоміжним символом a_{345} з імовірністю 0.4;

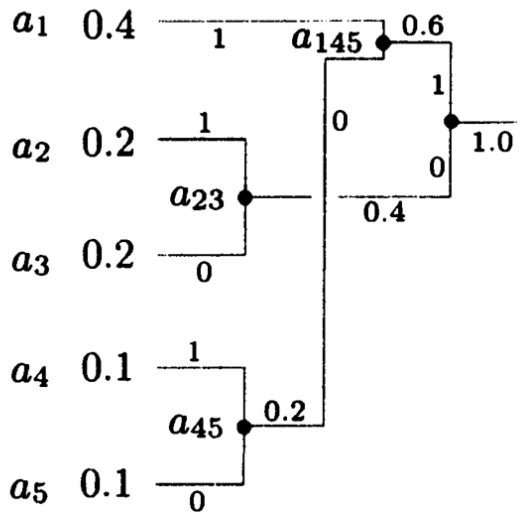
(3) тепер є три символи a_1, a_2 та a_{345} з ймовірностями 0.4, 0.2 та 0.4, відповідно. Вибираємо та об'єднуємо символи a_2 та a_{345} у допоміжний символ a_{2345} з ймовірністю 0.6;

(4) нарешті, об'єднуємо два символи a_1 і a_{2345} і замінюємо на a_{12345} з ймовірністю 1.

Статистичні методи стиснення зображень. Кодування Гаффмана

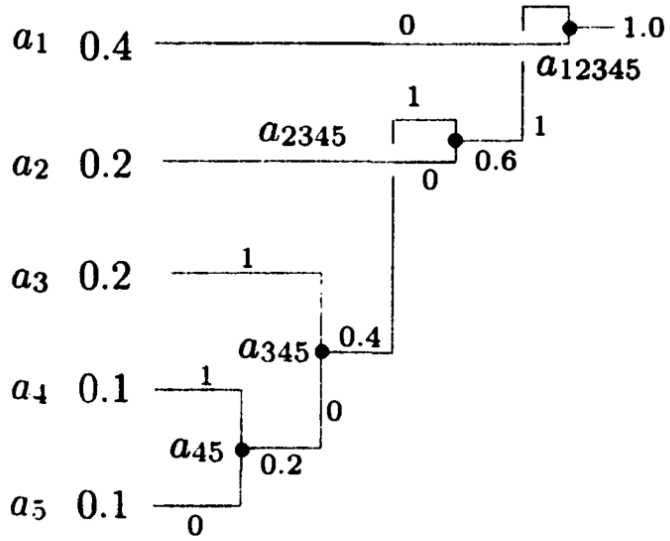


(a)

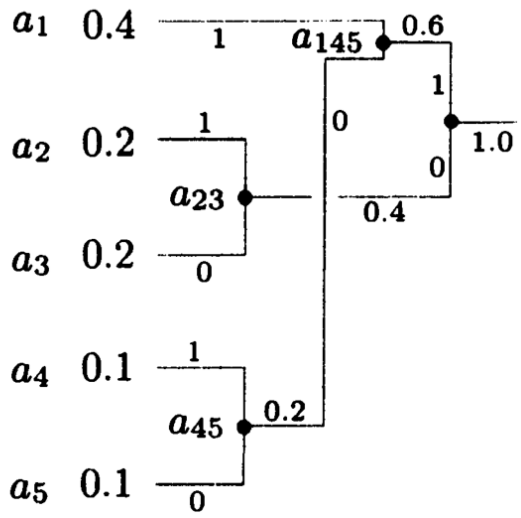


(b)

Дерево побудовано. Воно зображено на рис.(а), "лежачи на боці", з коренем праворуч і п'ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.

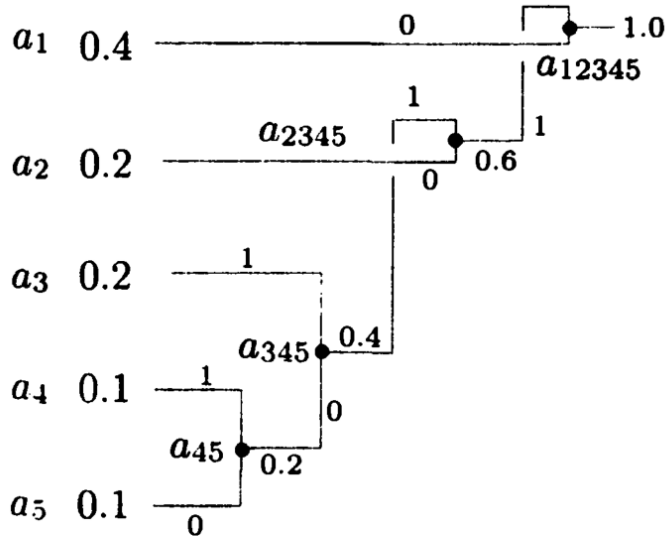


(a)

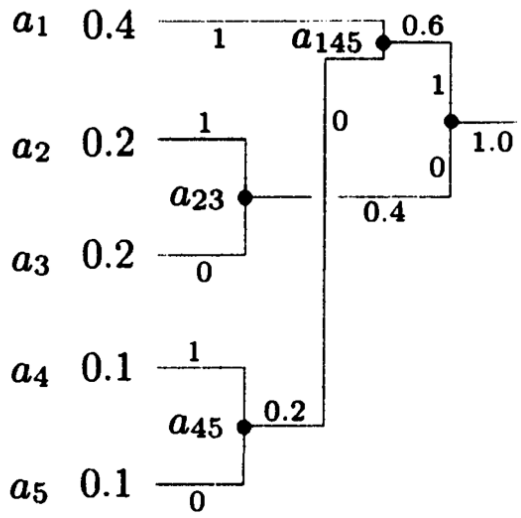


(b)

Дерево побудовано. Воно зображено на рис.(a), "лежачи на боці", з коренем праворуч і п'ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.

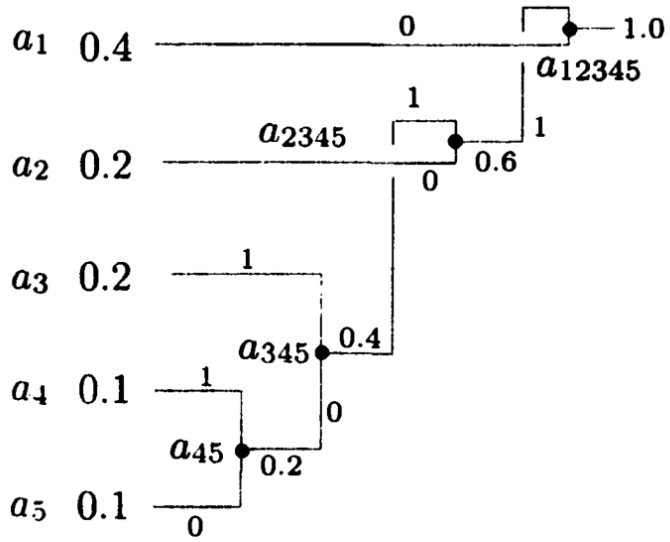


(a)

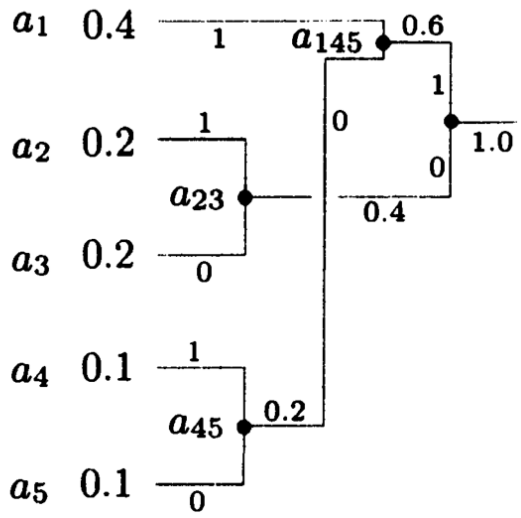


(b)

Дерево побудовано. Воно зображено на рис.(a), "лежачи на боці", з коренем праворуч і п'ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.

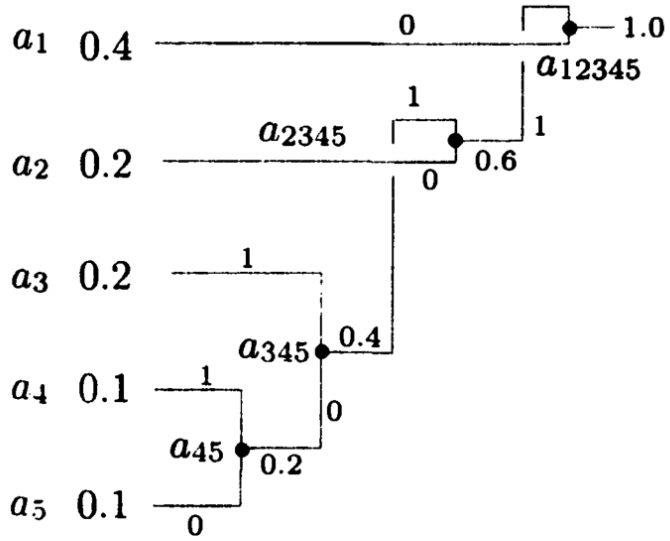


(a)

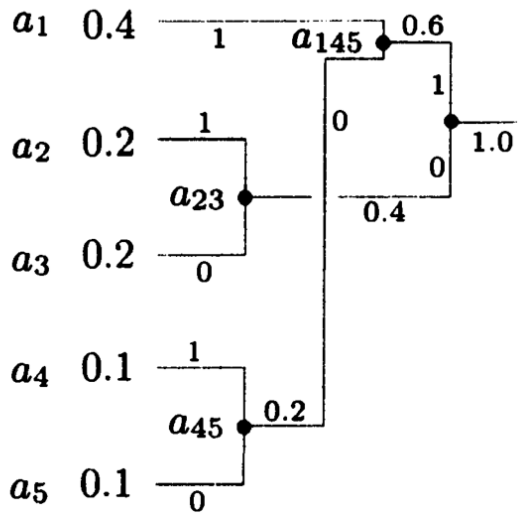


(b)

Дерево побудовано. Воно зображено на рис.(a), “лежачи на боці”, з коренем праворуч і п’ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.

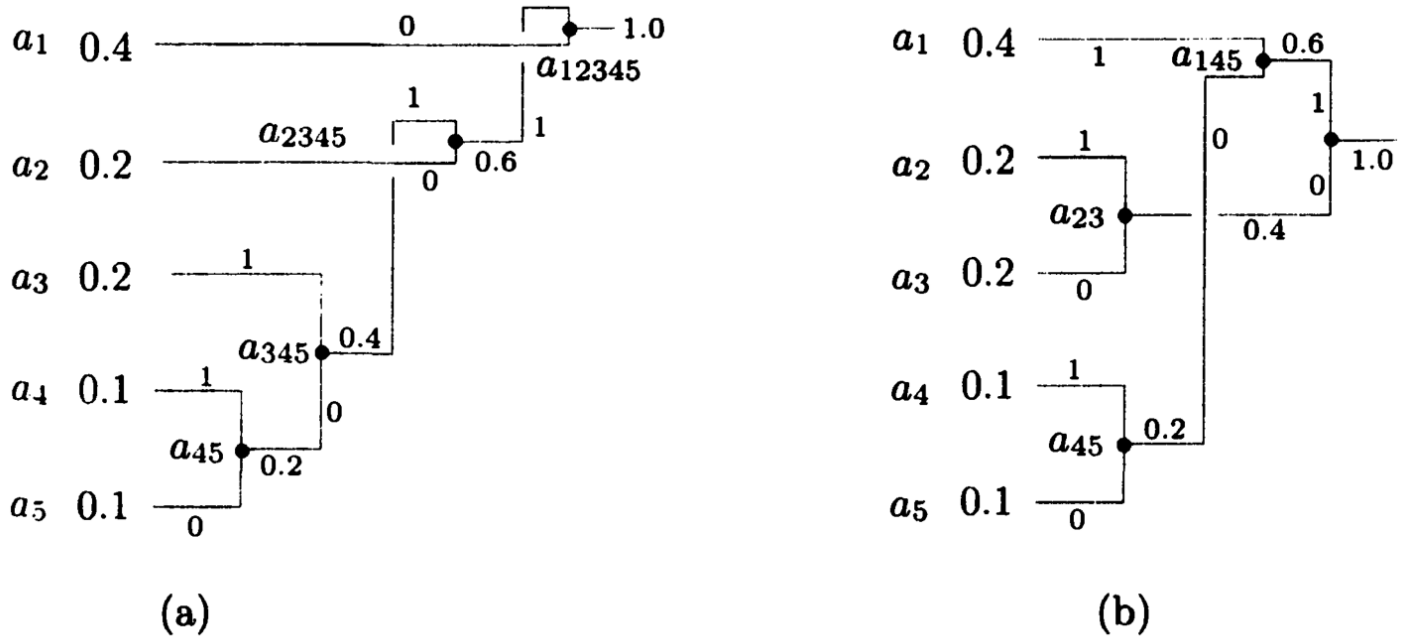


(a)

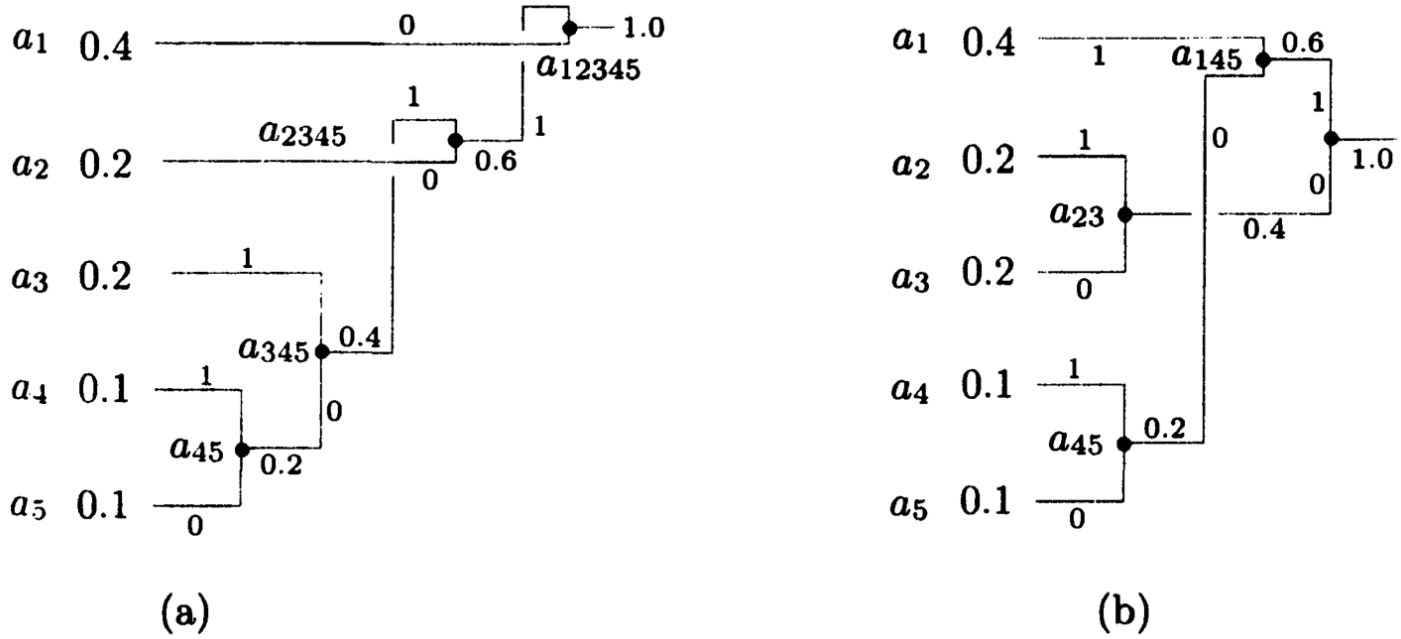


(b)

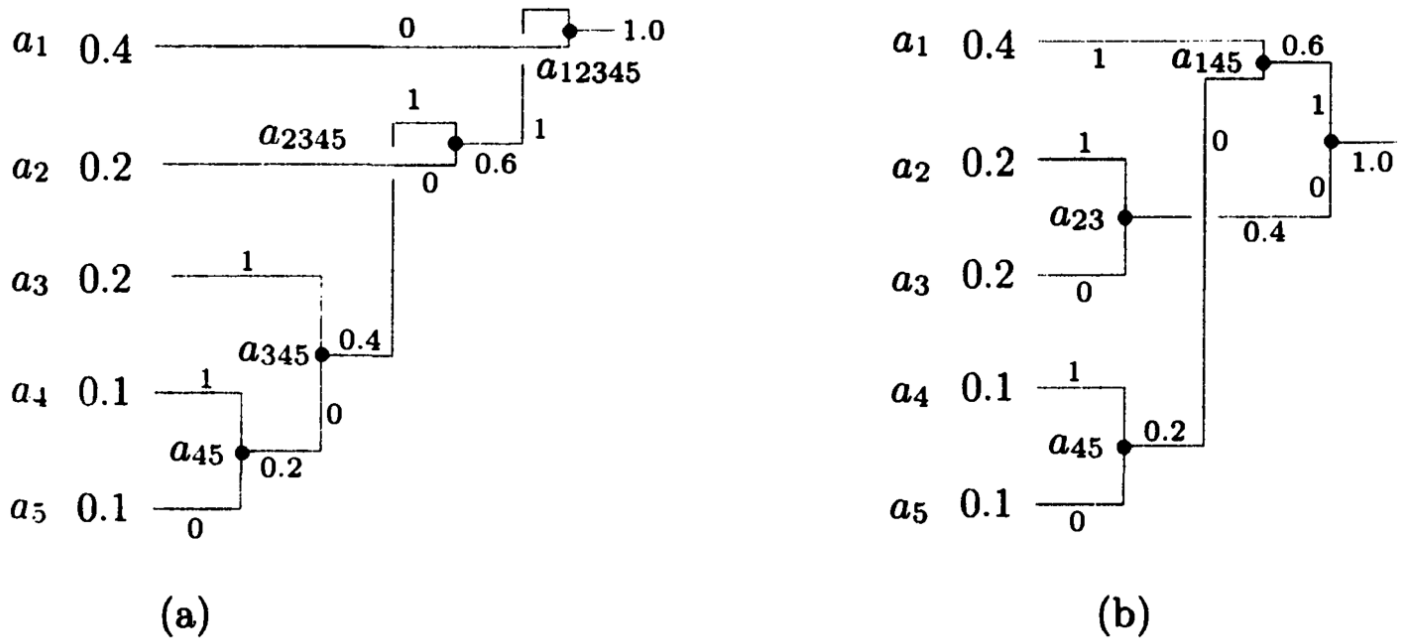
Дерево побудовано. Воно зображено на рис.(a), “лежачи на боці”, з коренем праворуч і п’ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.



Дерево побудовано. Воно зображено на рис.(a), “лежачи на боці”, з коренем праворуч і п’ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.



Дерево побудовано. Воно зображено на рис.(a), “лежачи на боці”, з коренем праворуч і п’ятьма листами зліва. Для призначення кодів ми довільно приписуємо 1 біт верхній гілці і 0 біт нижній гілці дерева для кожної пари. В результаті отримуємо такі коди: 0, 10, 111, 1101 та 1100. Розподіл бітів по краях — довільний.



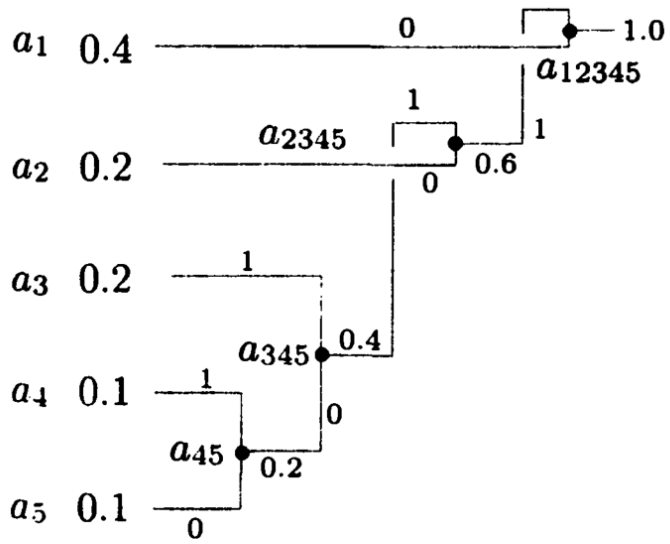
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

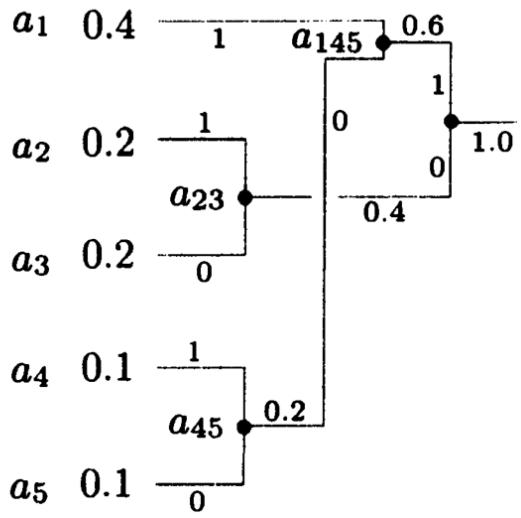
біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



(a)



(b)

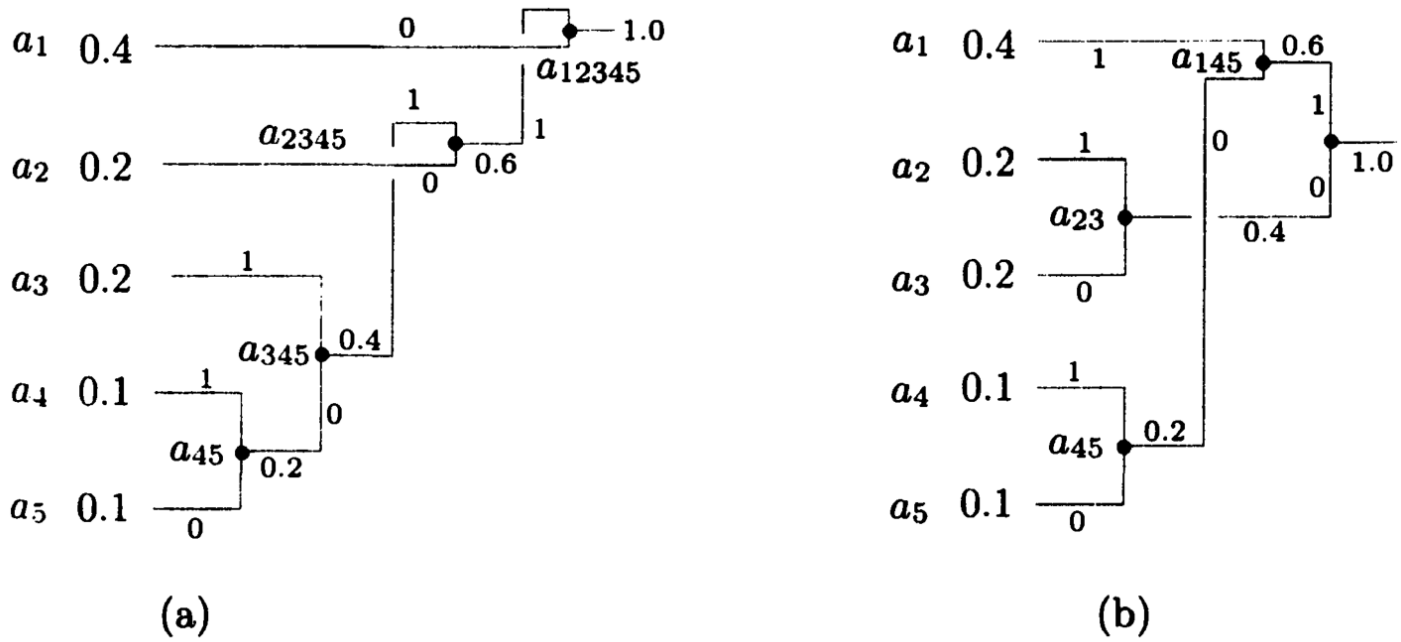
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



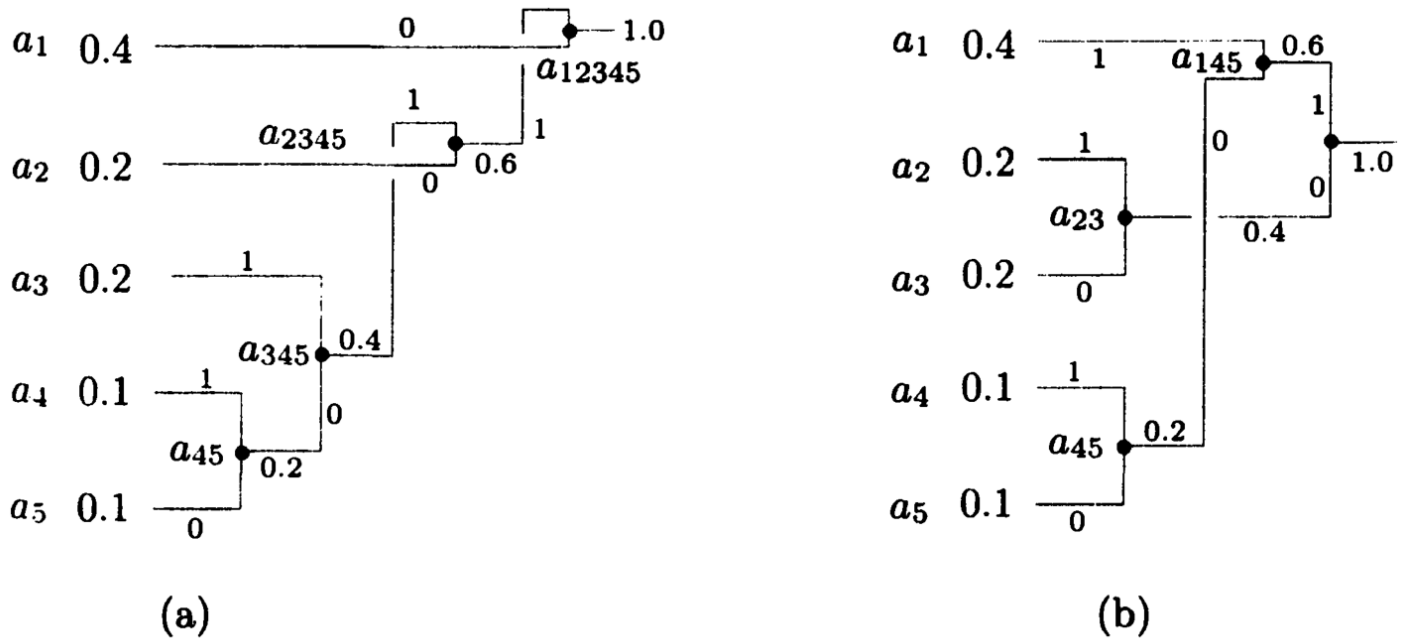
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



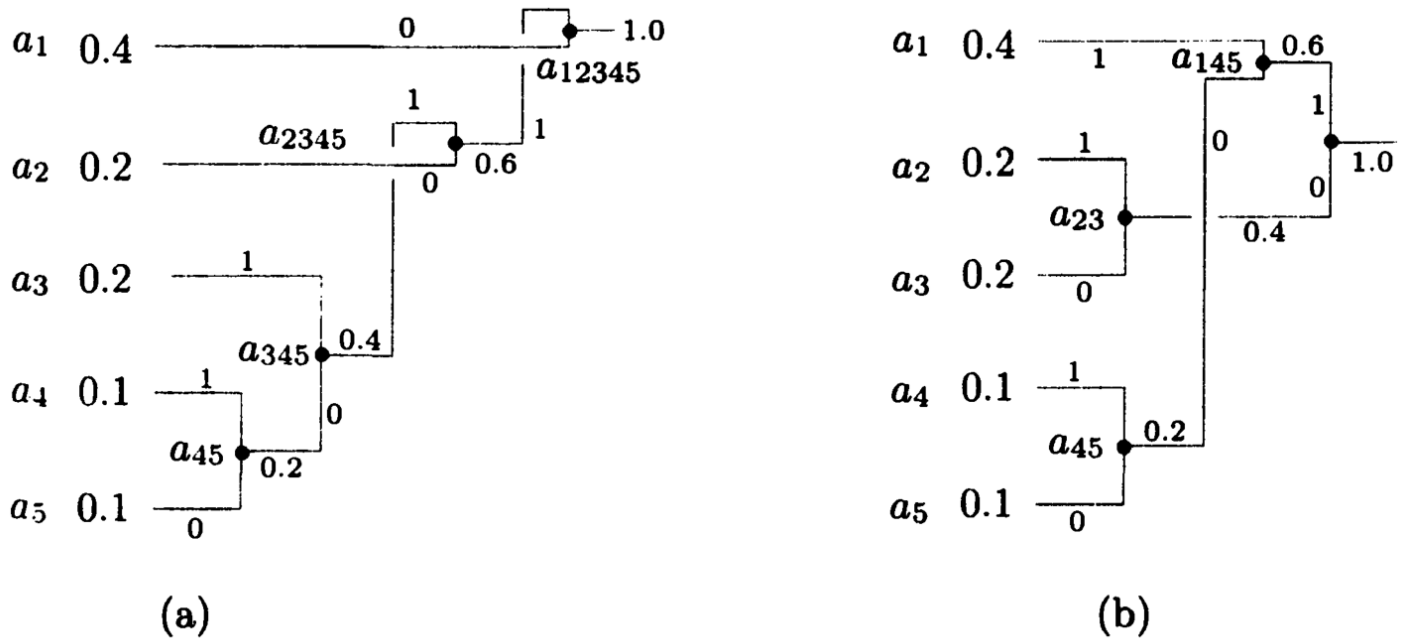
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



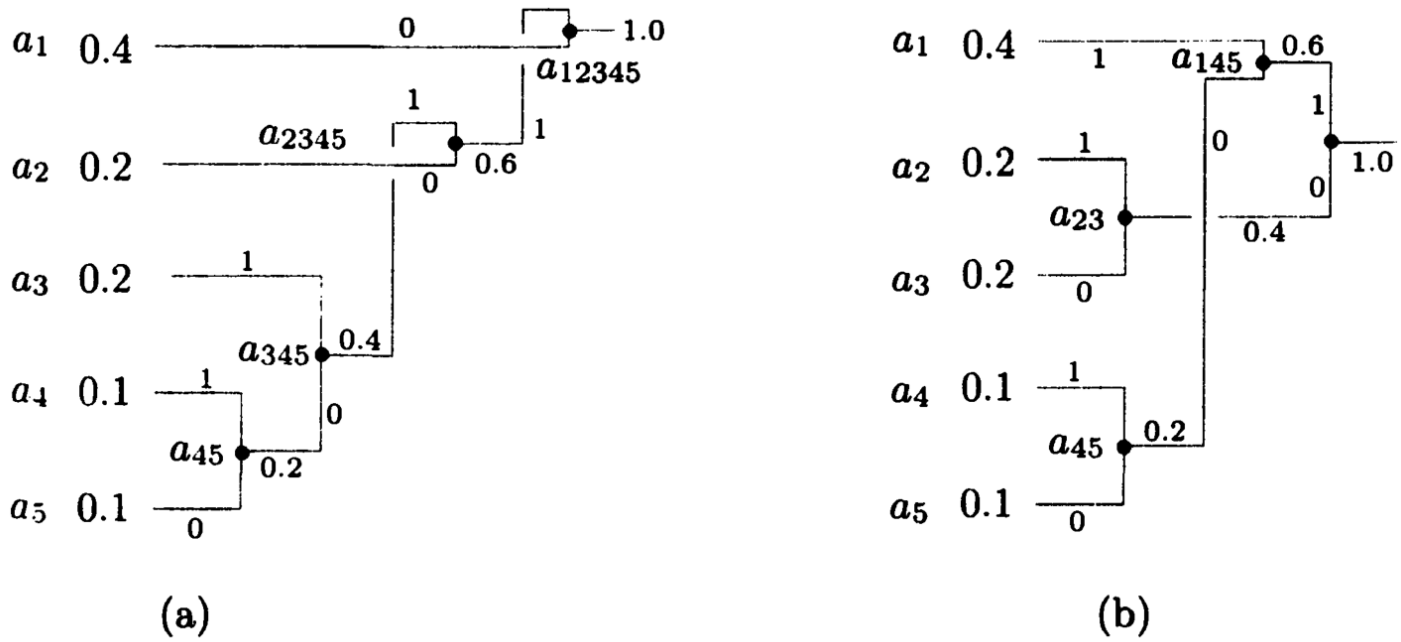
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



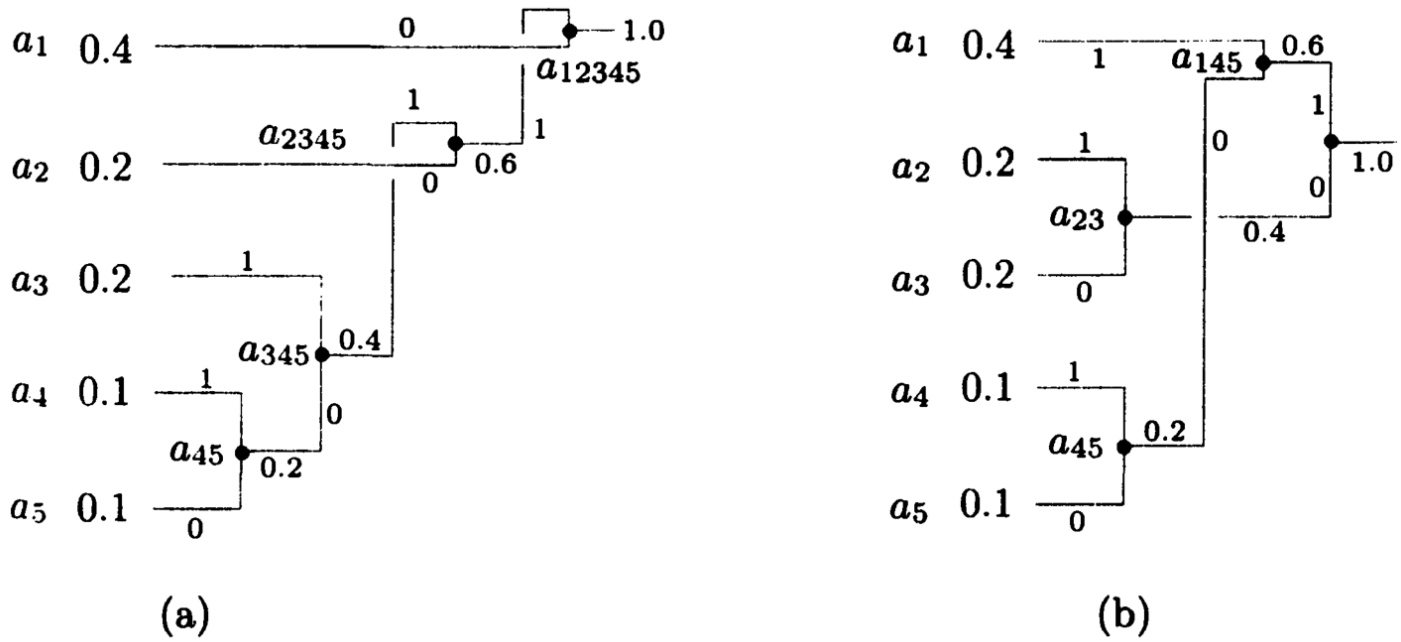
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



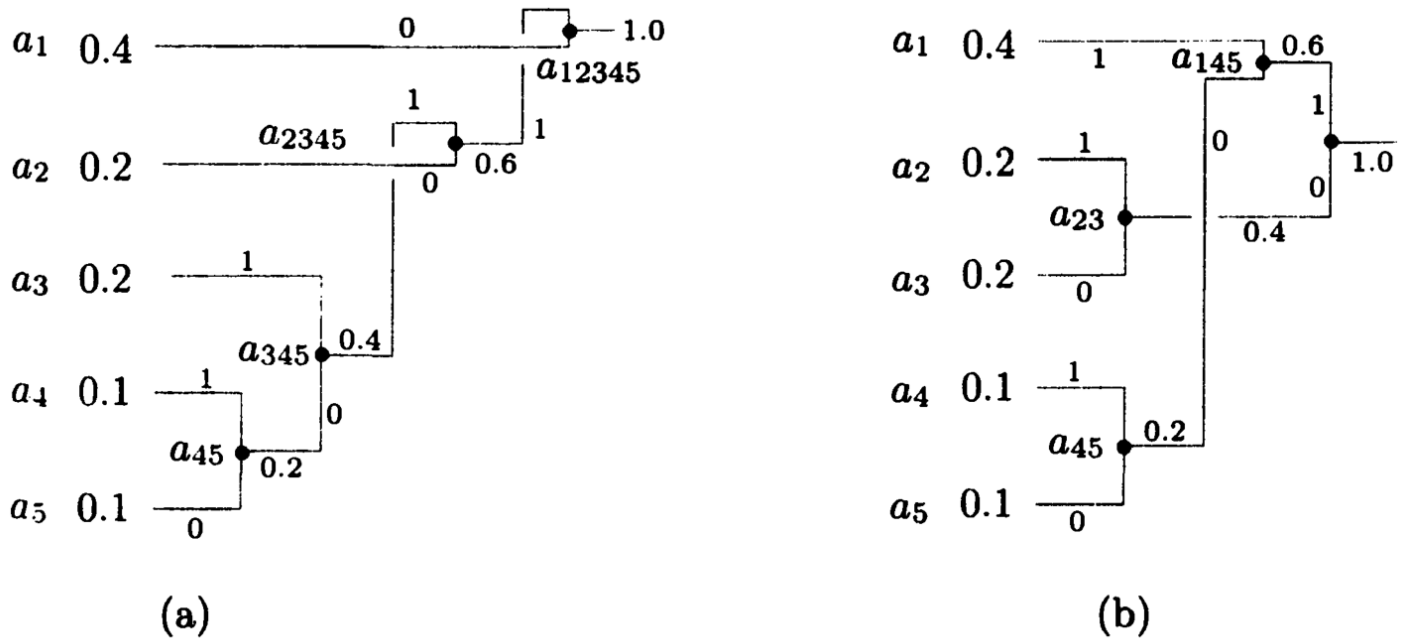
Середня довжина цього коду дорівнює

$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

біт/символ як і в попереднього коду.



Середня довжина цього коду дорівнює

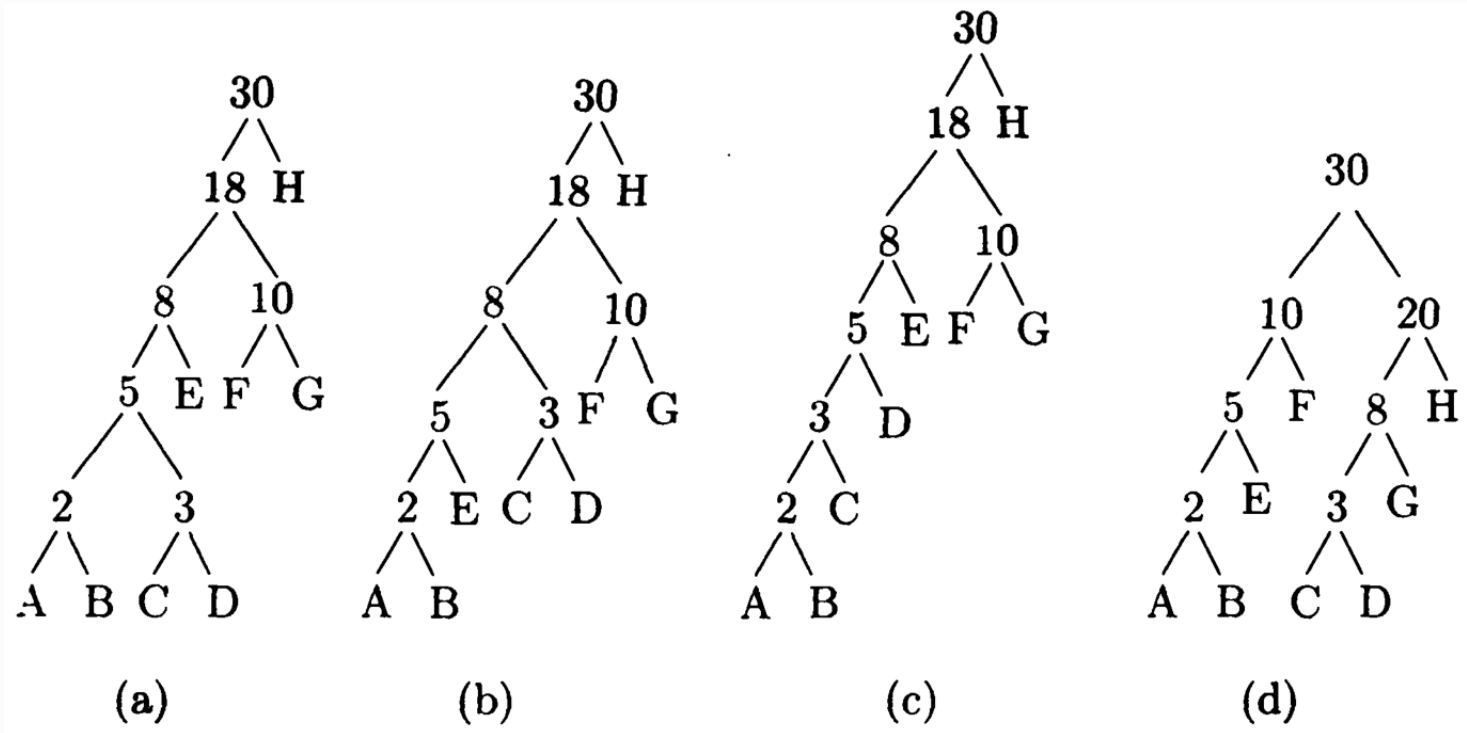
$$0.4 \cdot 1 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.1 \cdot 4 + 0.1 \cdot 4 = 2.2$$

біт/символ. Дуже важливим є те, що кодів Гаффмана буває багато. Деякі кроки алгоритму вибиралися довільним чином, оскільки було більше символів із мінімальною ймовірністю. На рис. (b) зображено, як можна об'єднати символи по-іншому та отримати інший код Гаффмана (11, 01, 00, 101 та 100). Середня довжина дорівнює

$$0.4 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 2.2$$

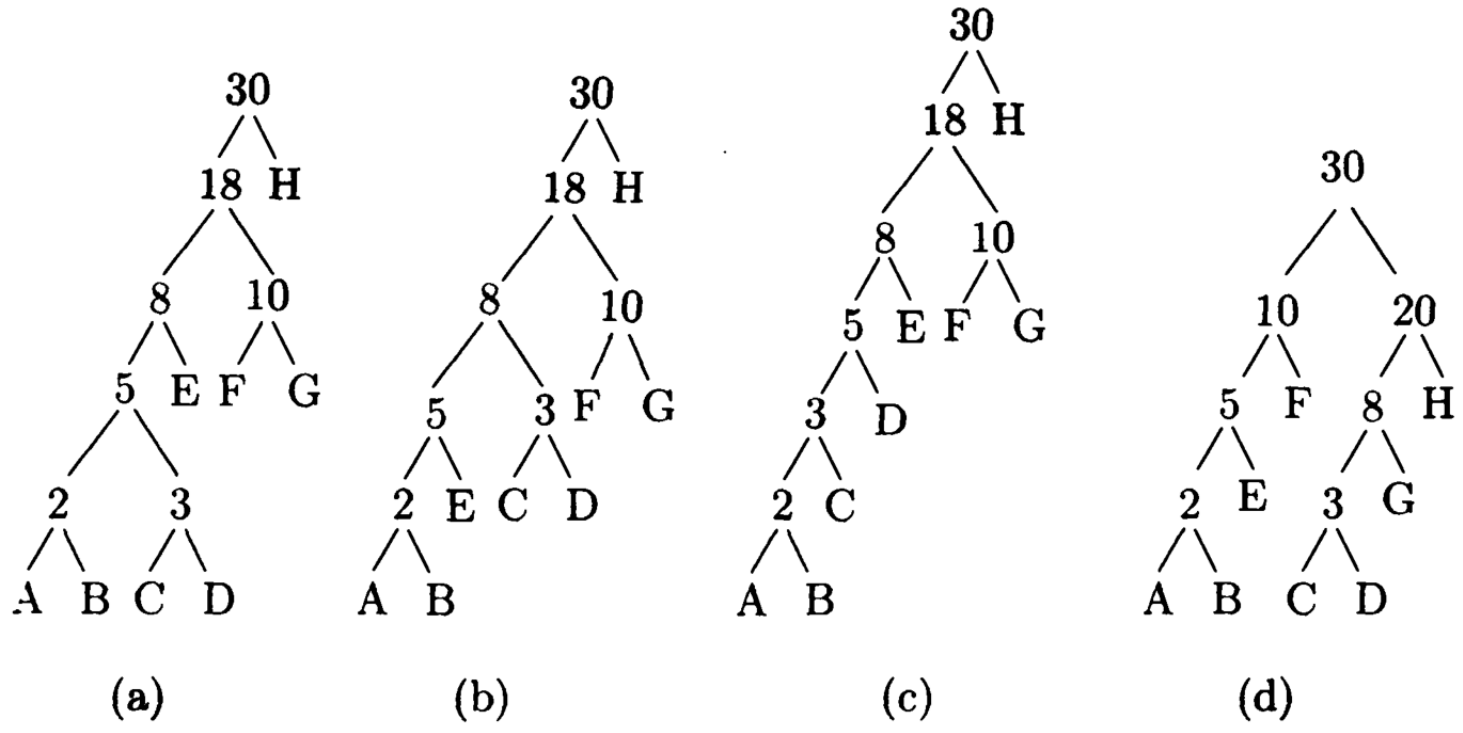
біт/символ як і в попереднього коду.

Приклад


30
18 H
8 10
5 E F G
2 3
A B C D
(a)
30
18 H
8 10
5 3 F G
2 E C D
A B
(b)
30
18 H
8 10
5 E F G
3 D
2 C
A B
(c)
30
10 20
5 F 8 H
2 E 3 G
A B C D
(d)

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту. Середня довжина цих кодів (у бітах на символ) дорівнює

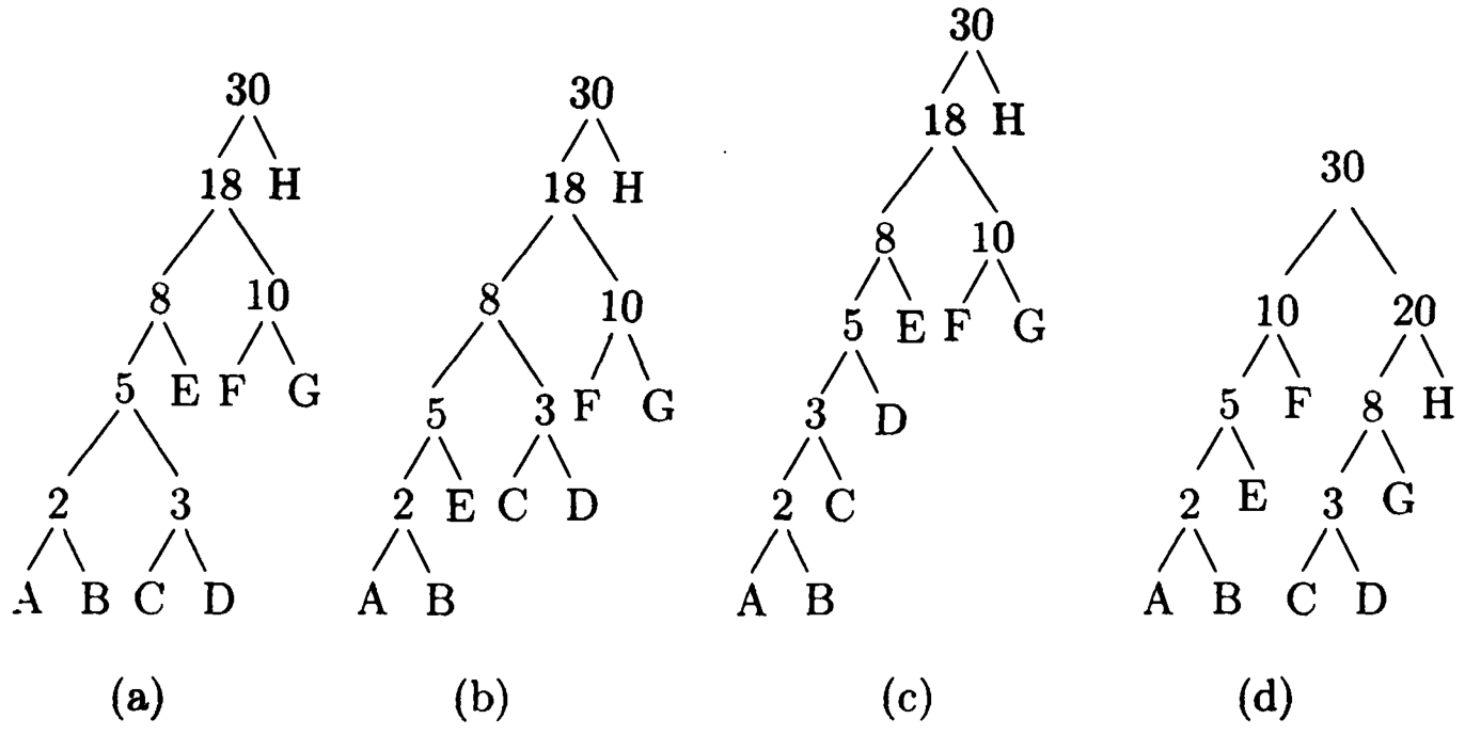
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту. Середня довжина цих кодів (у бітах на символ) дорівнює

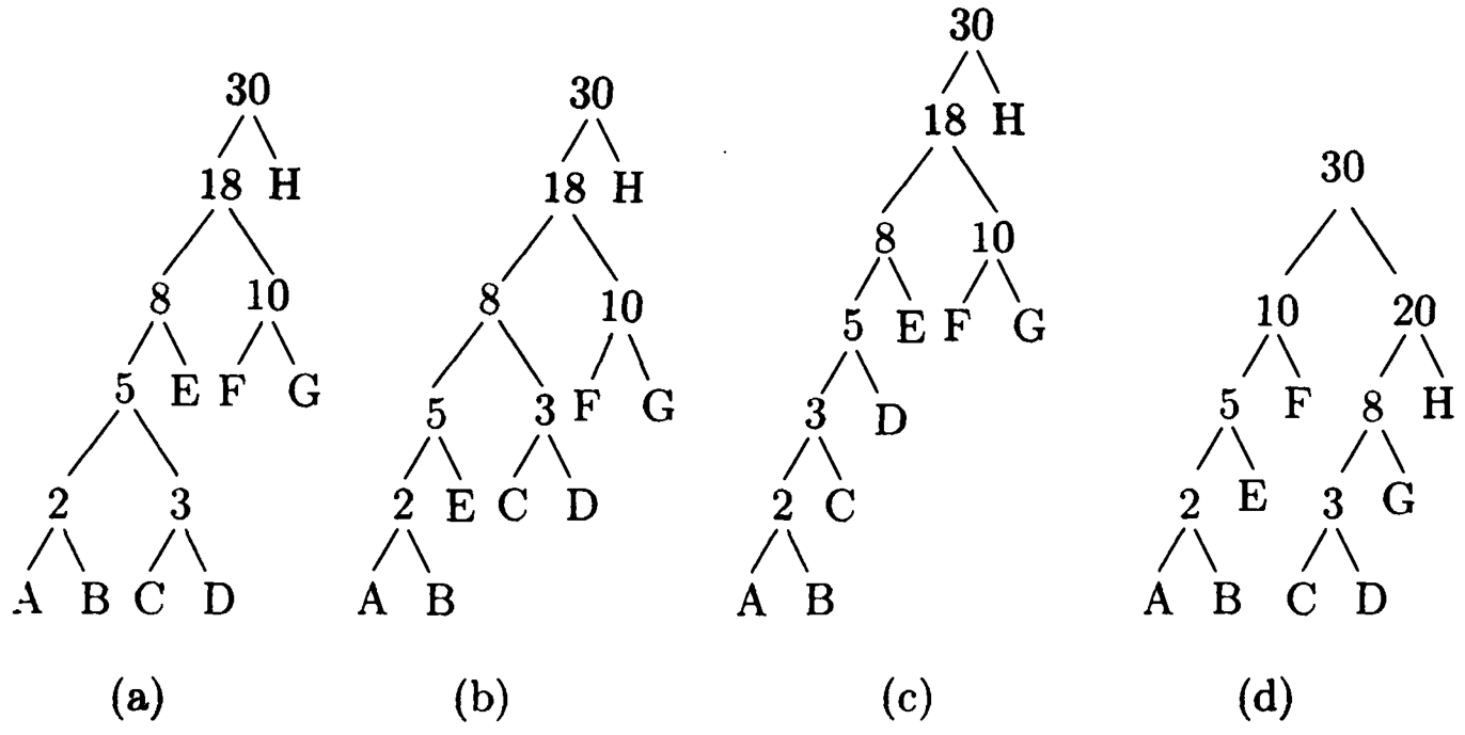
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту. Середня довжина цих кодів (у бітах на символ) дорівнює

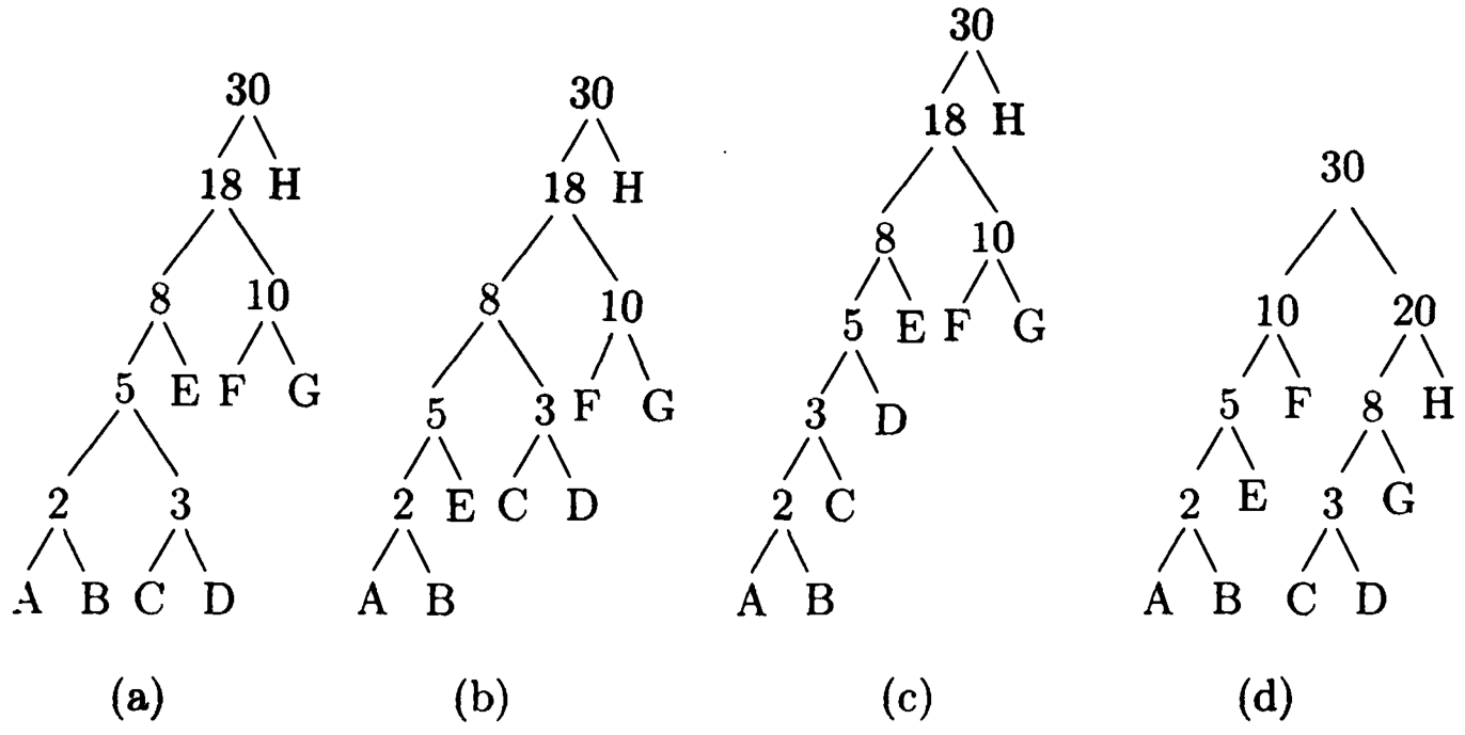
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту. Середня довжина цих кодів (у бітах на символ) дорівнює

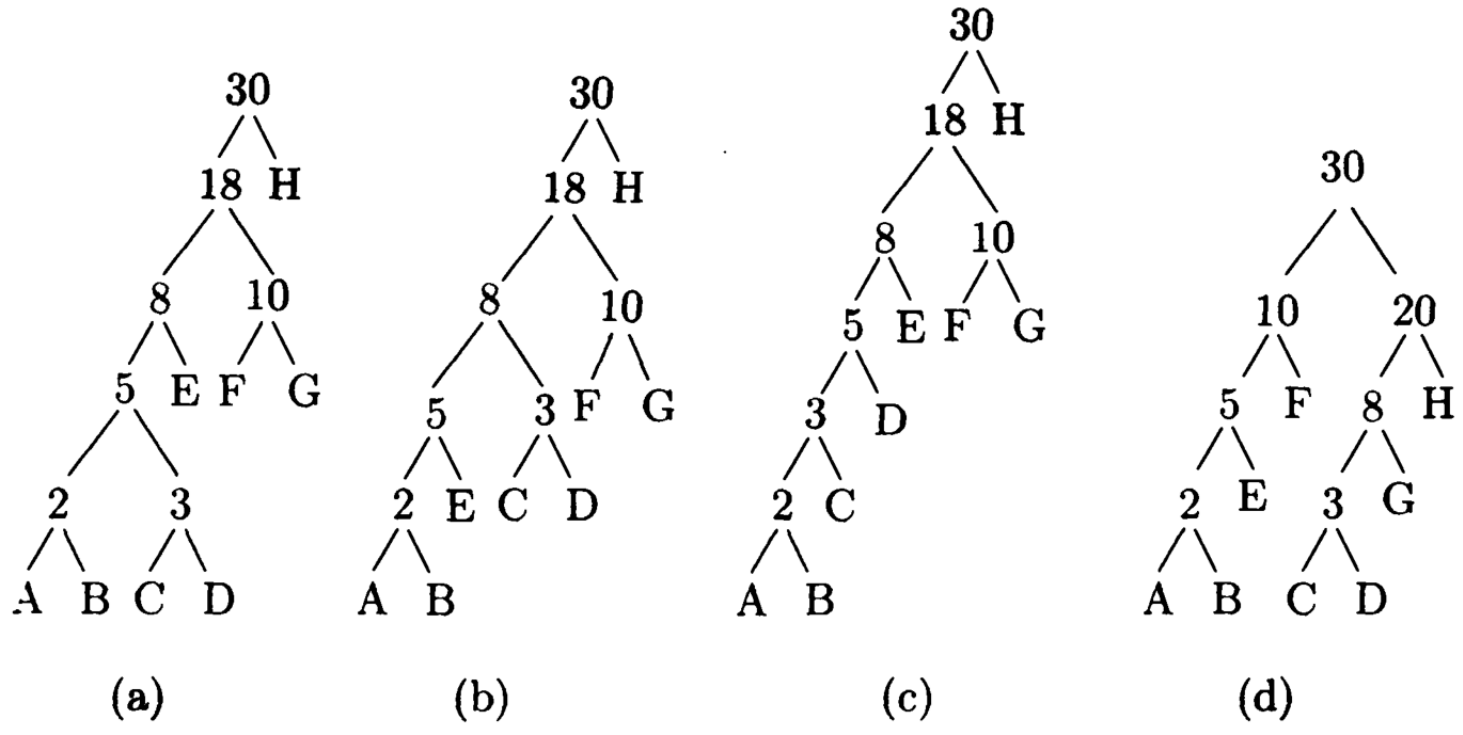
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту.

Середня довжина цих кодів (у бітах на символ) дорівнює

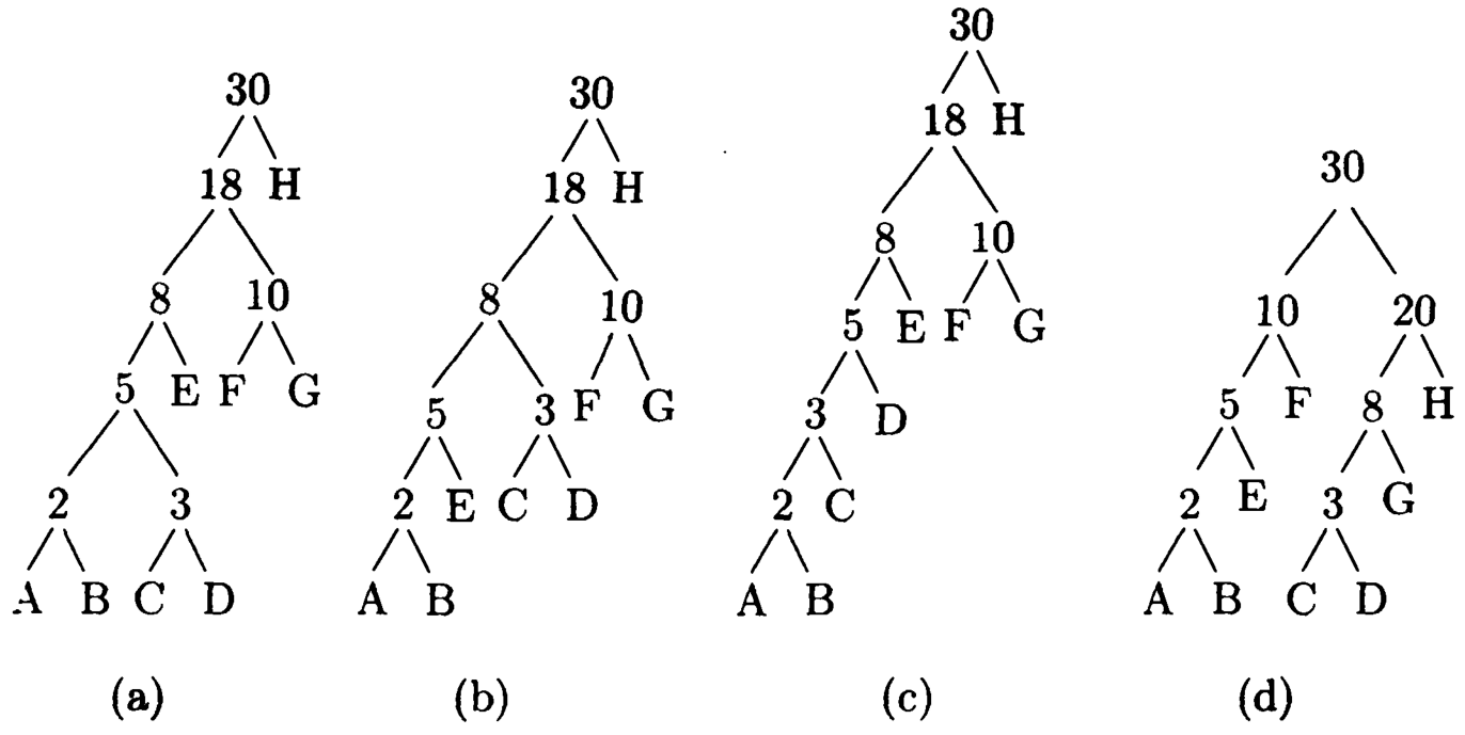
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту. Середня довжина цих кодів (у бітах на символ) дорівнює

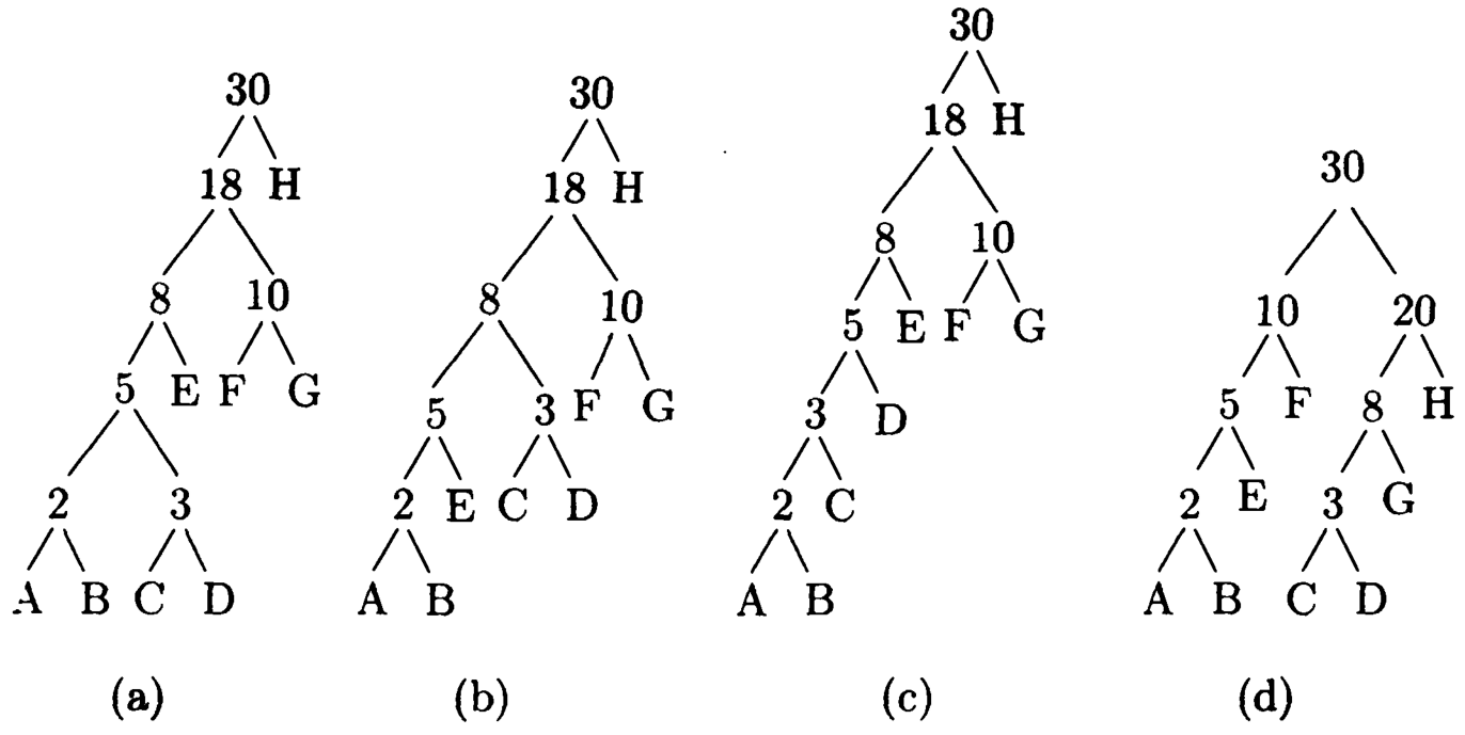
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

Дано 8 символів A, B, C, D, E, F, G та H з ймовірностями $1/30$, $1/30$, $1/30$, $2/30$, $3/30$, $5/30$, $5/30$ та $12/30$. На рис. (a),(b),(c)



зображені три дерева кодів Гаффмана висоти 5 і 6 для цього алфавіту. Середня довжина цих кодів (у бітах на символ) дорівнює

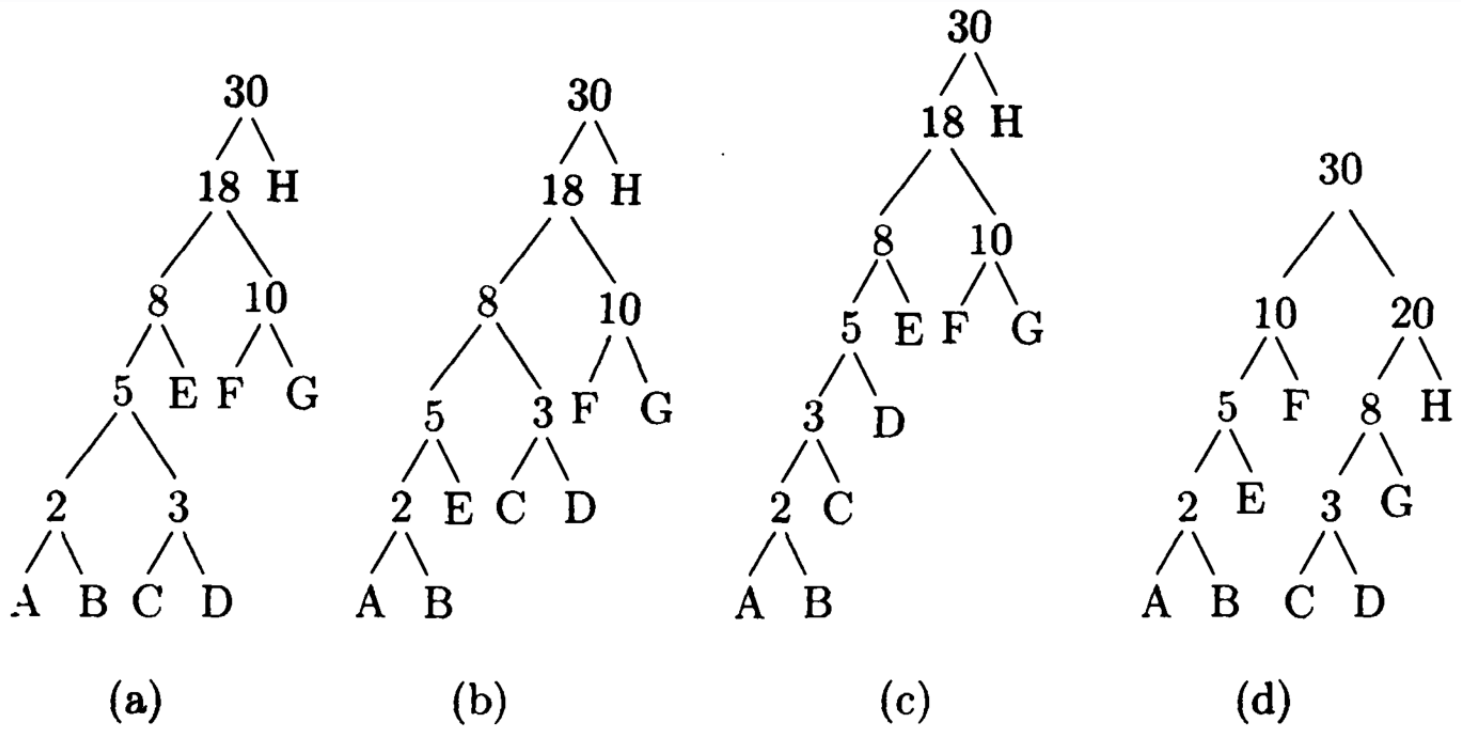
$$(5+5+5+5\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(5+5+4+4\cdot 2+4\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30,$$

$$(6+6+5+4\cdot 2+3\cdot 3+3\cdot 5+3\cdot 5+12)/30 = 76/30.$$

Приклад

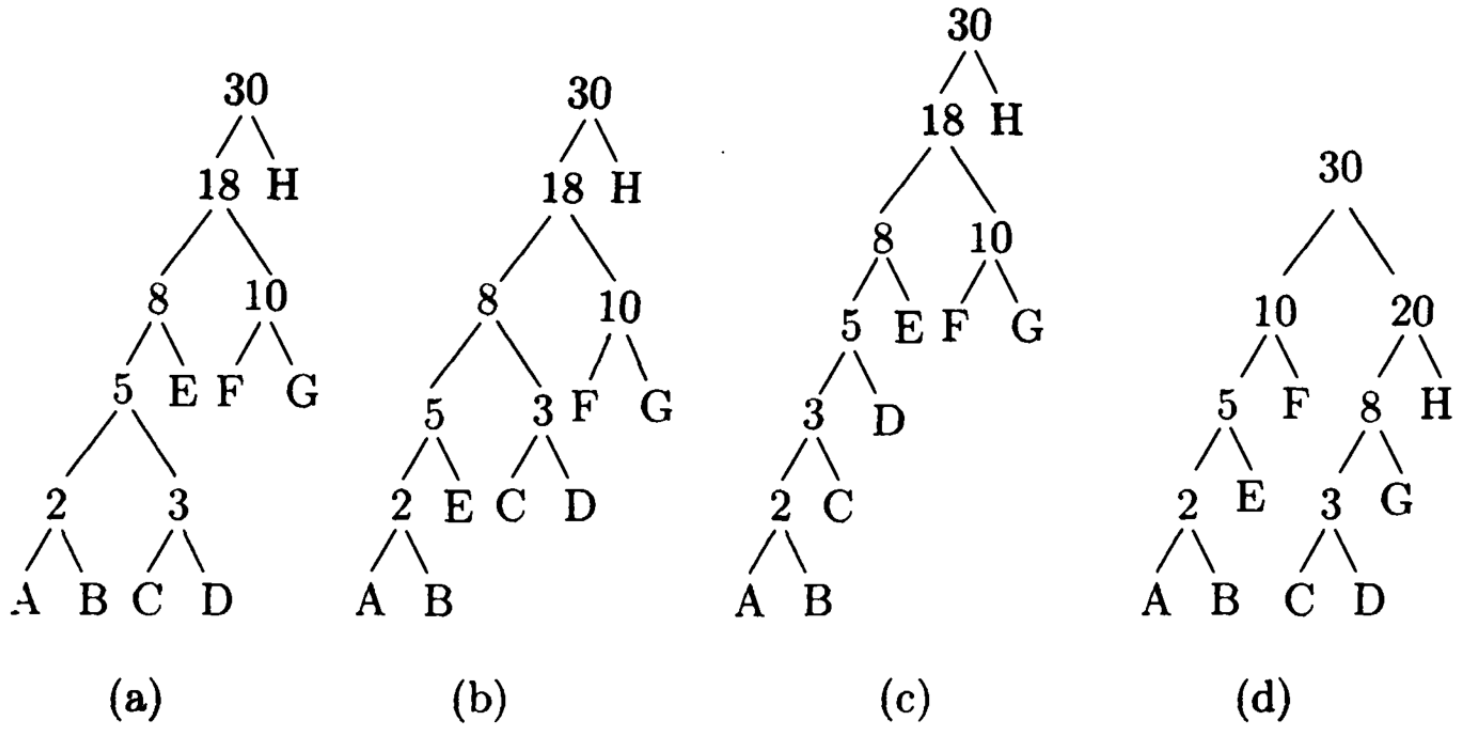
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

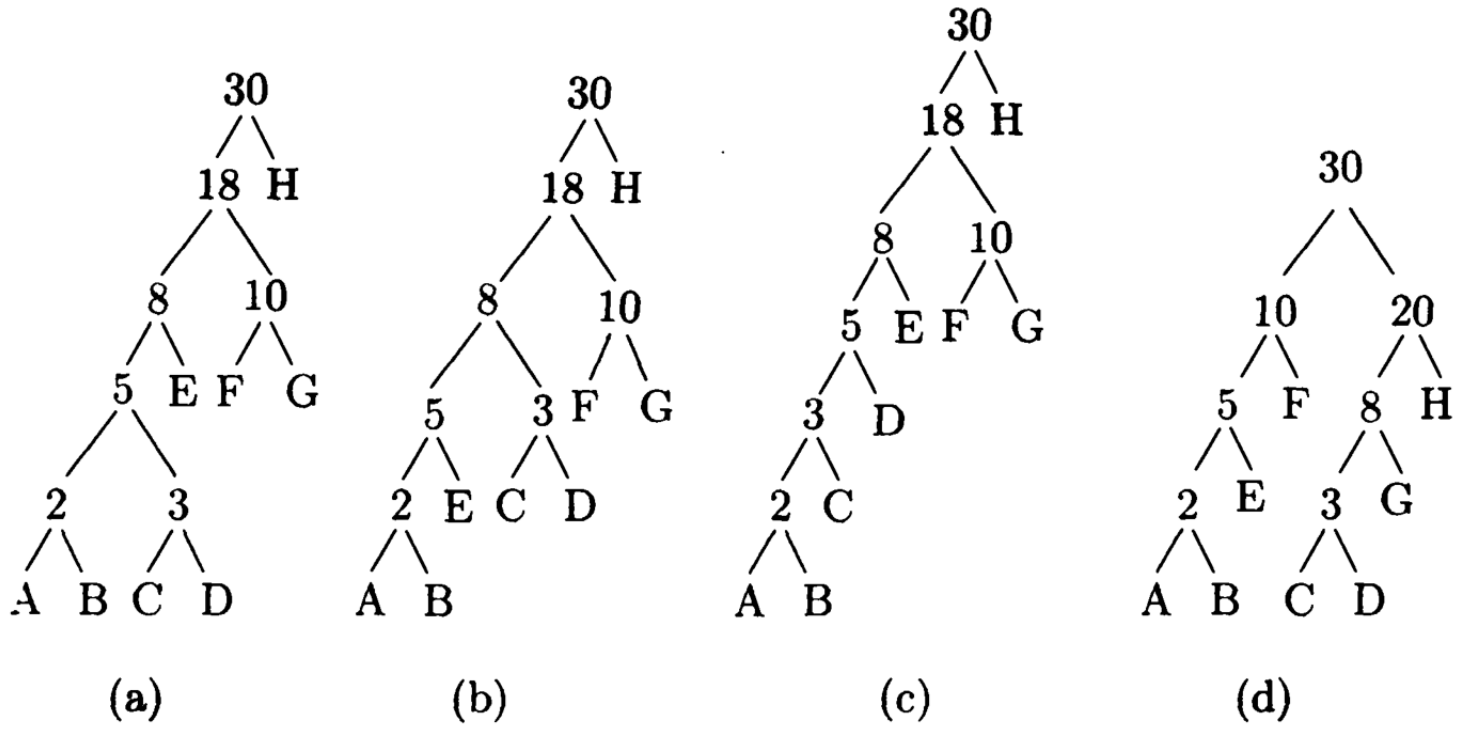
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

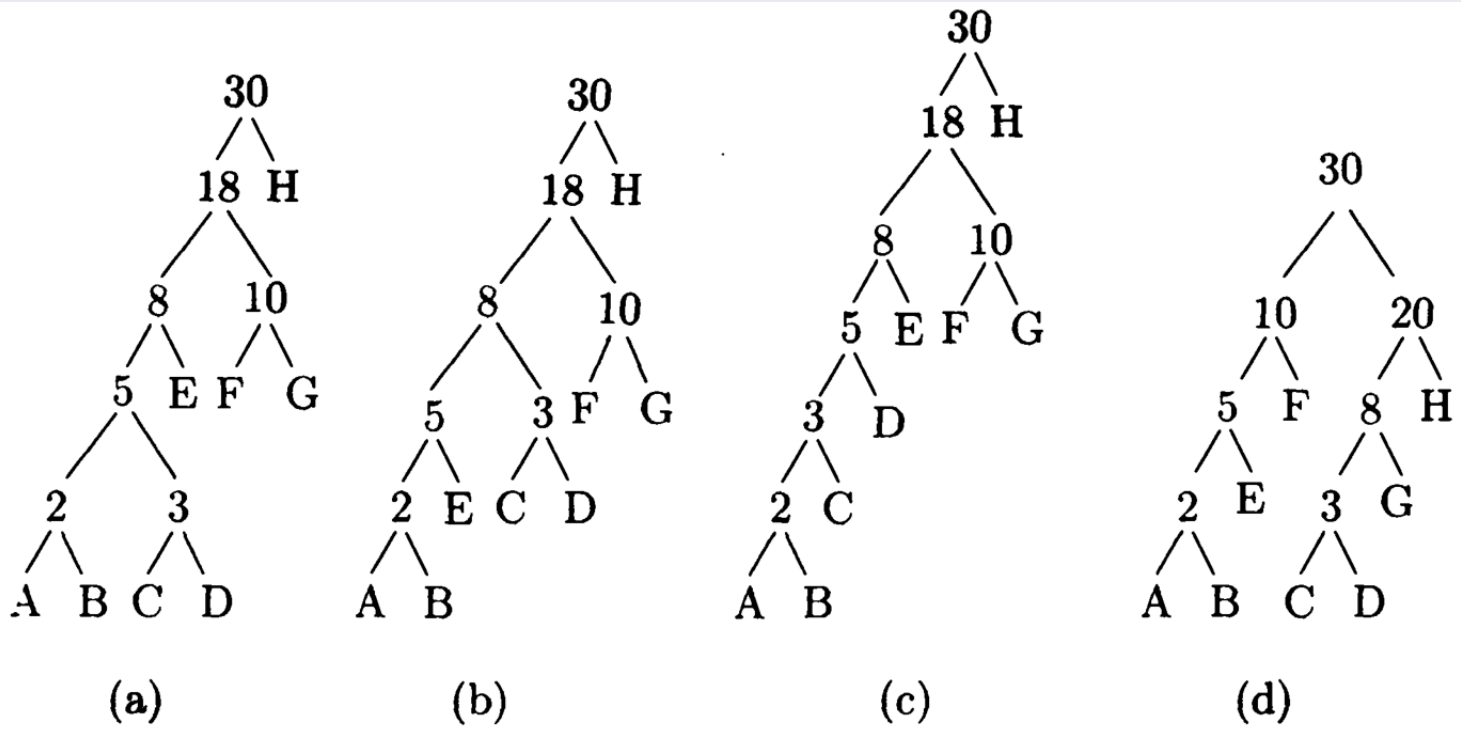
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

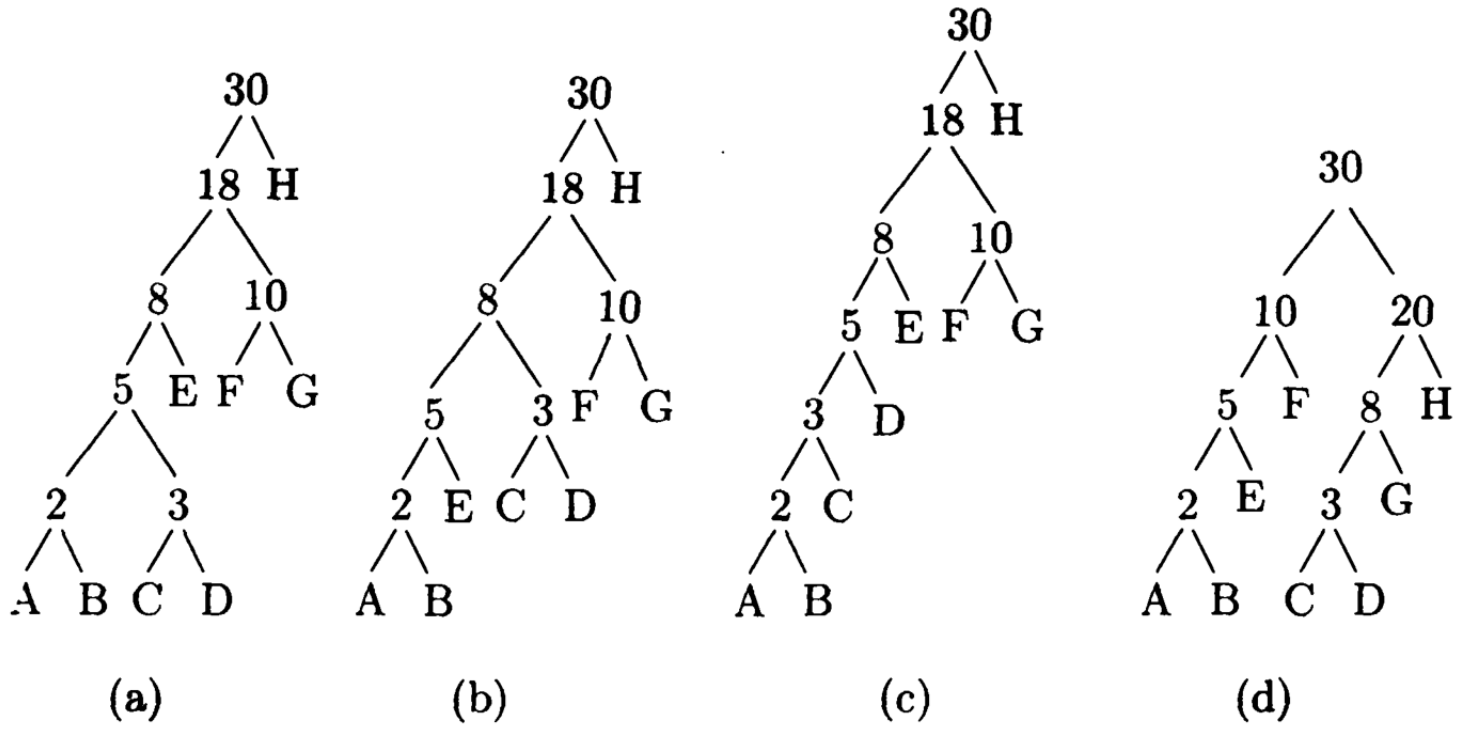
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

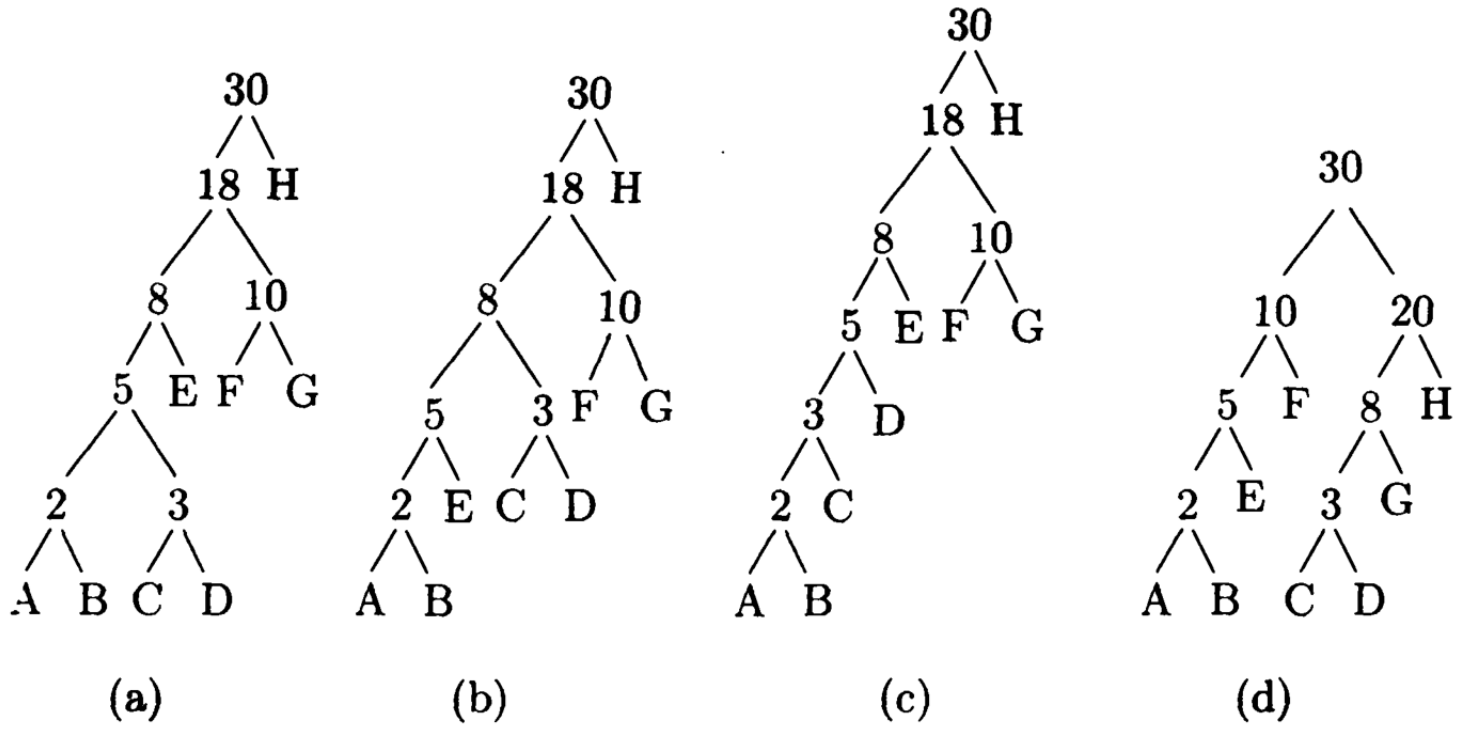
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

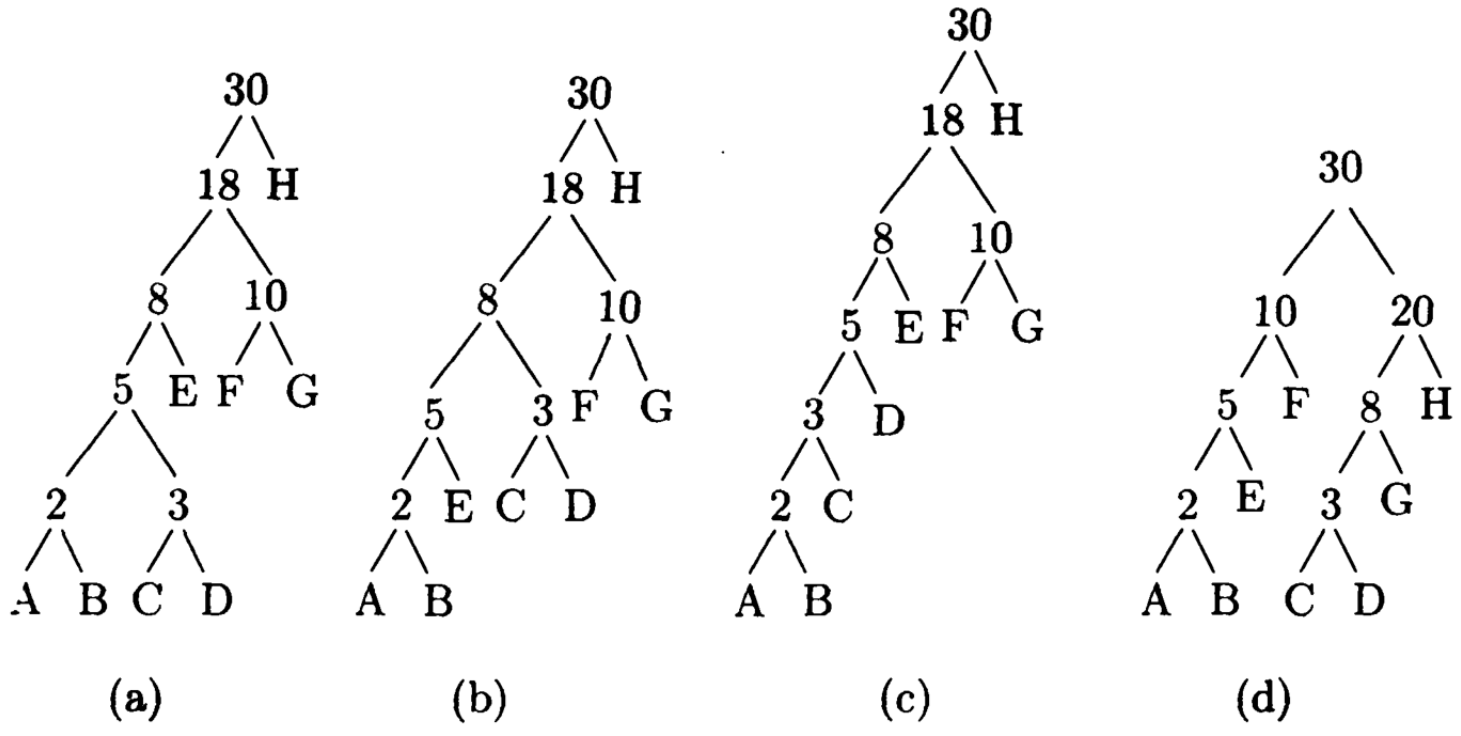
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

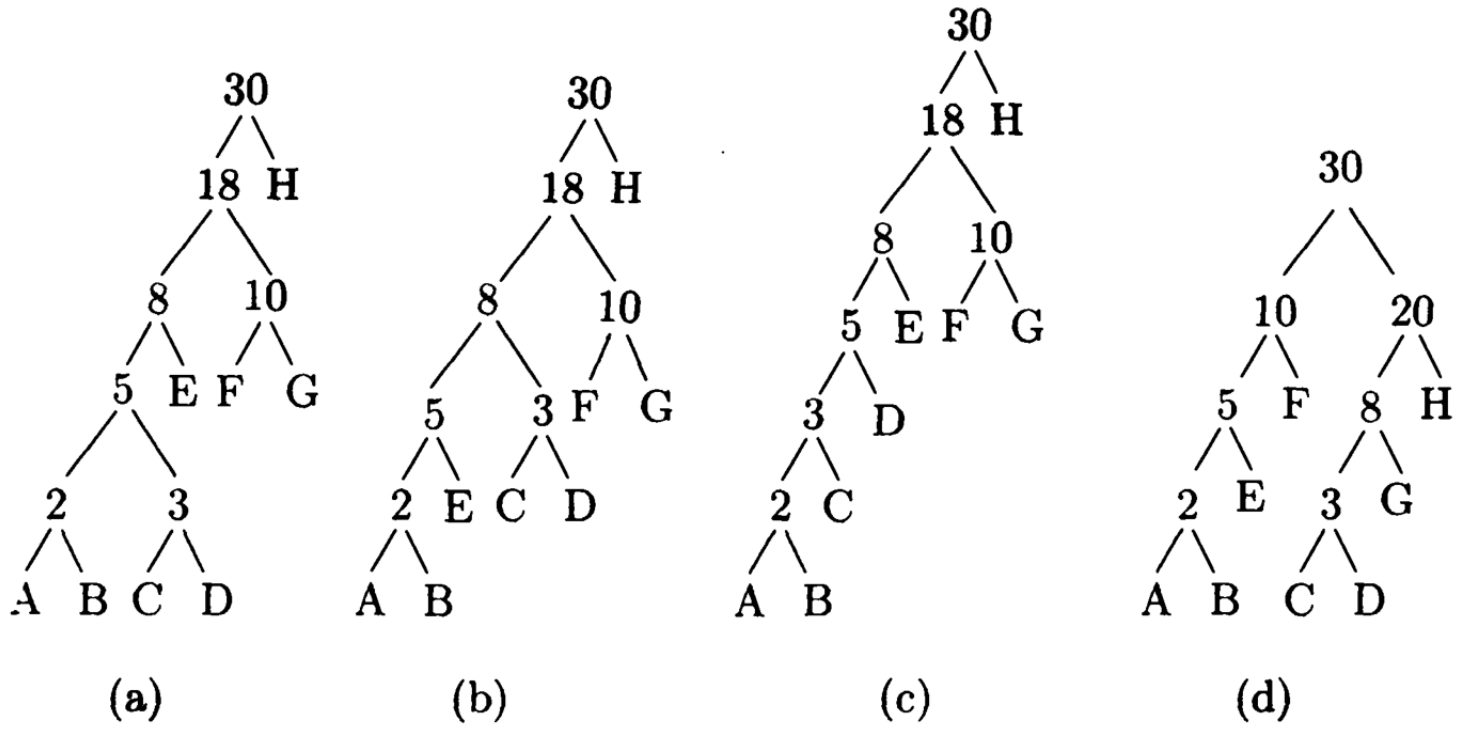
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

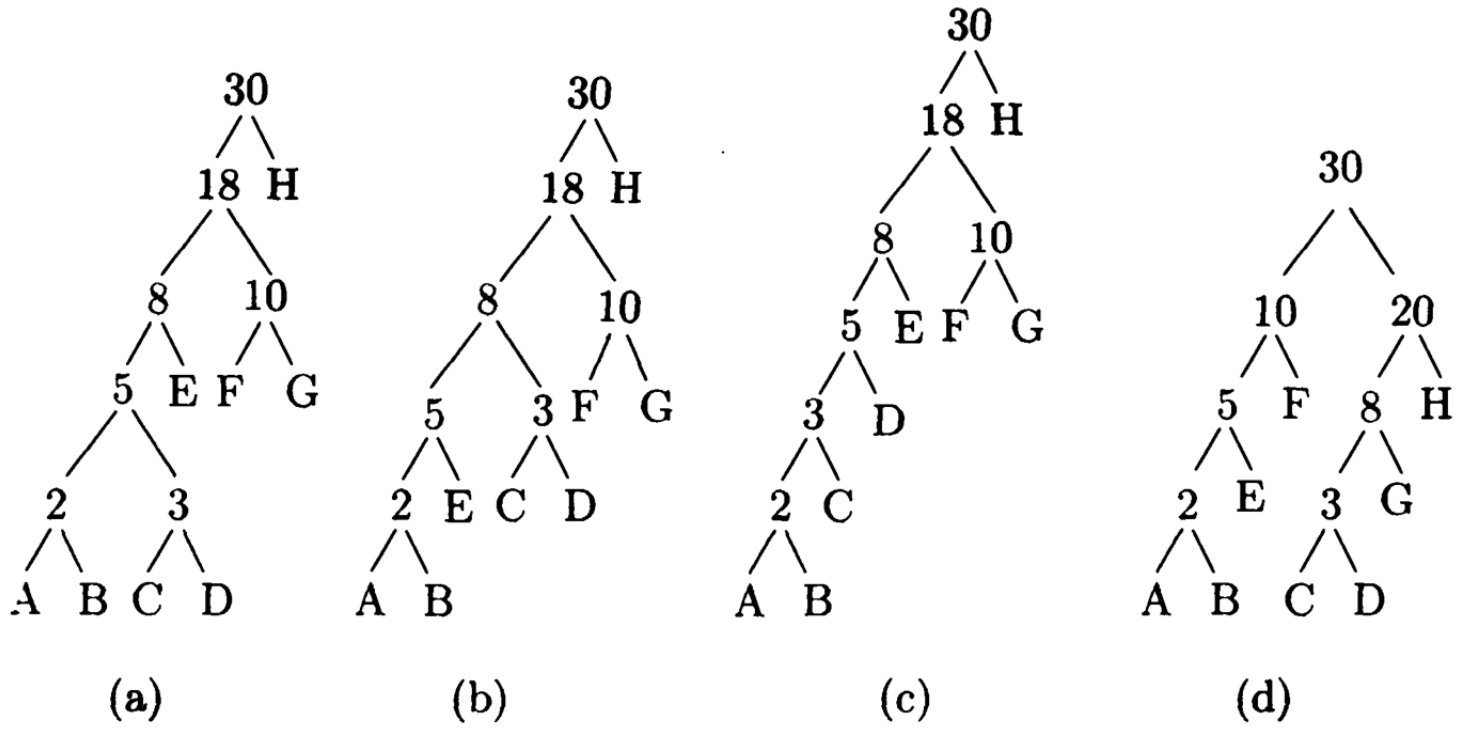
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

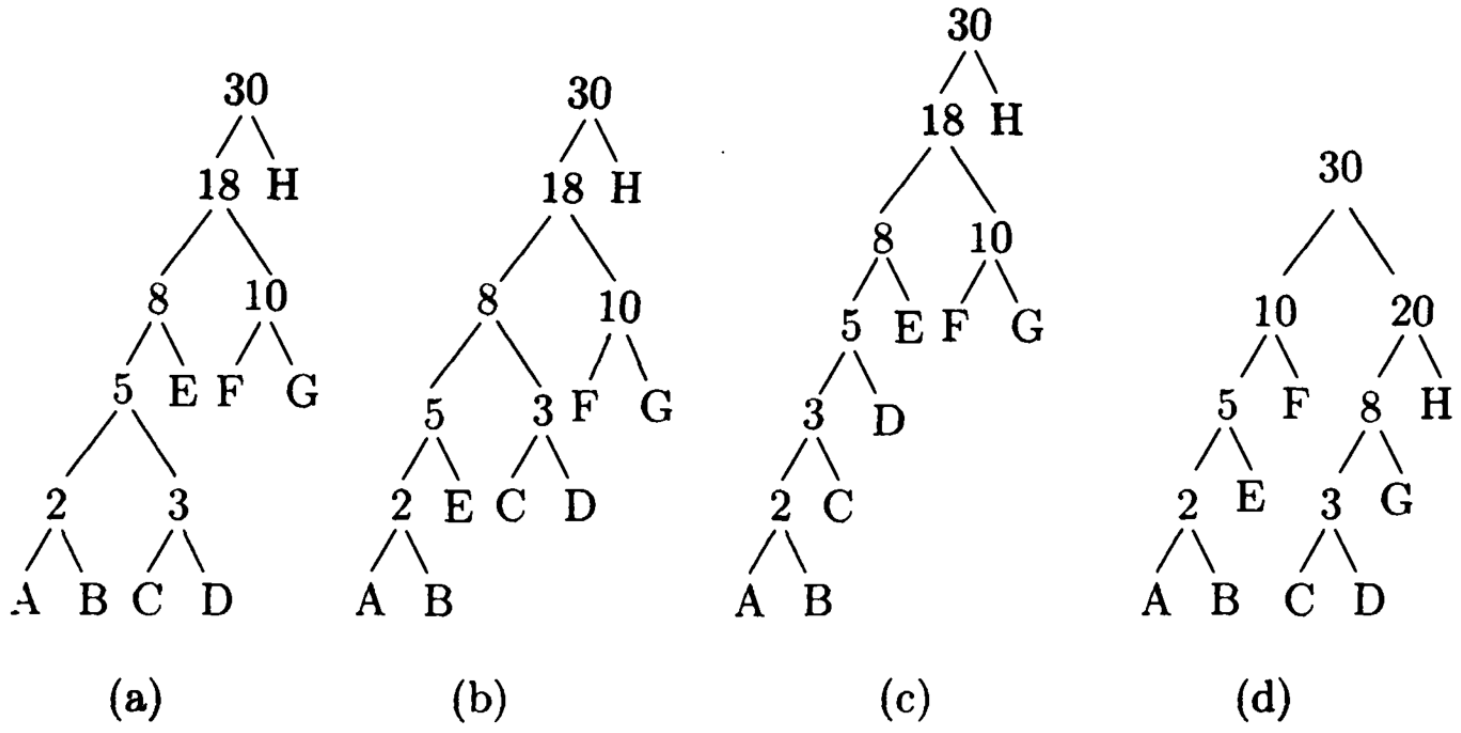
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

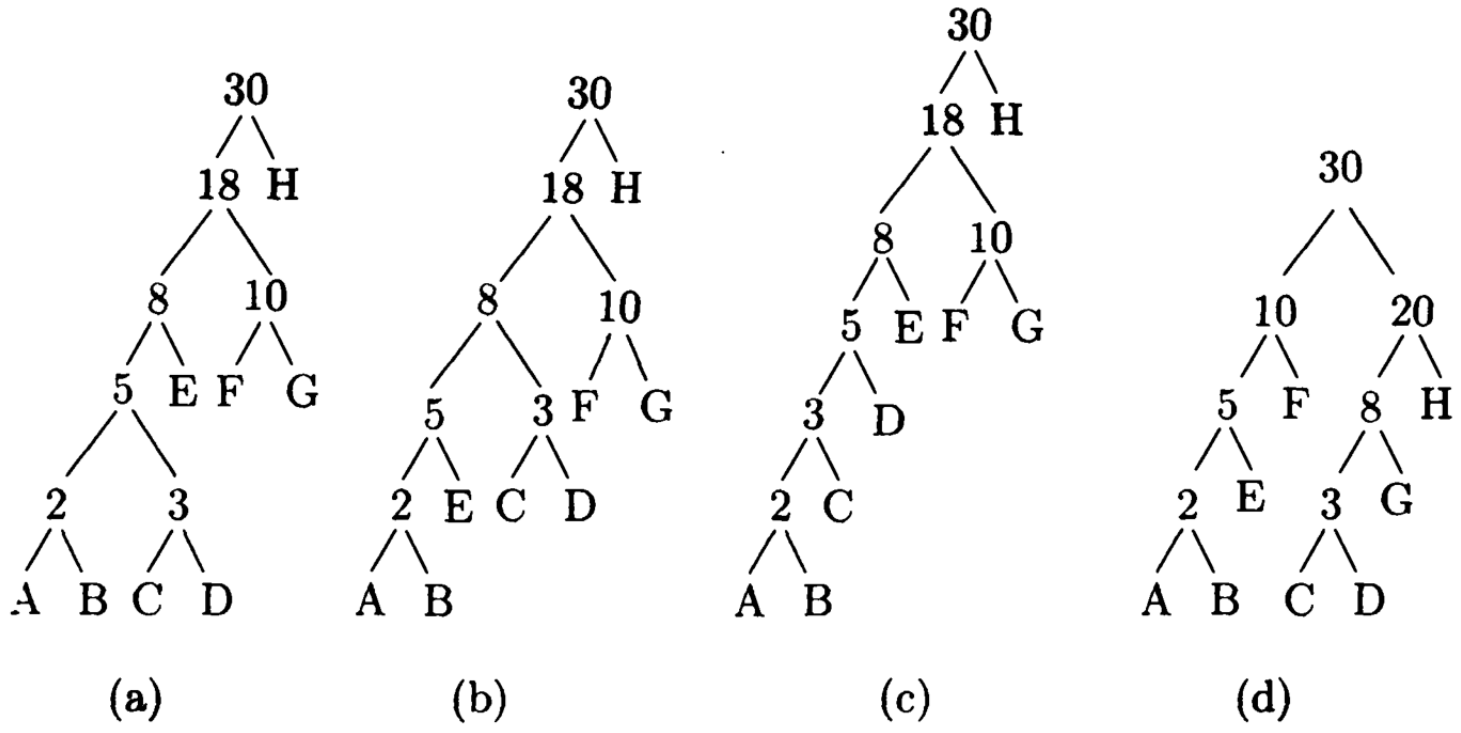
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

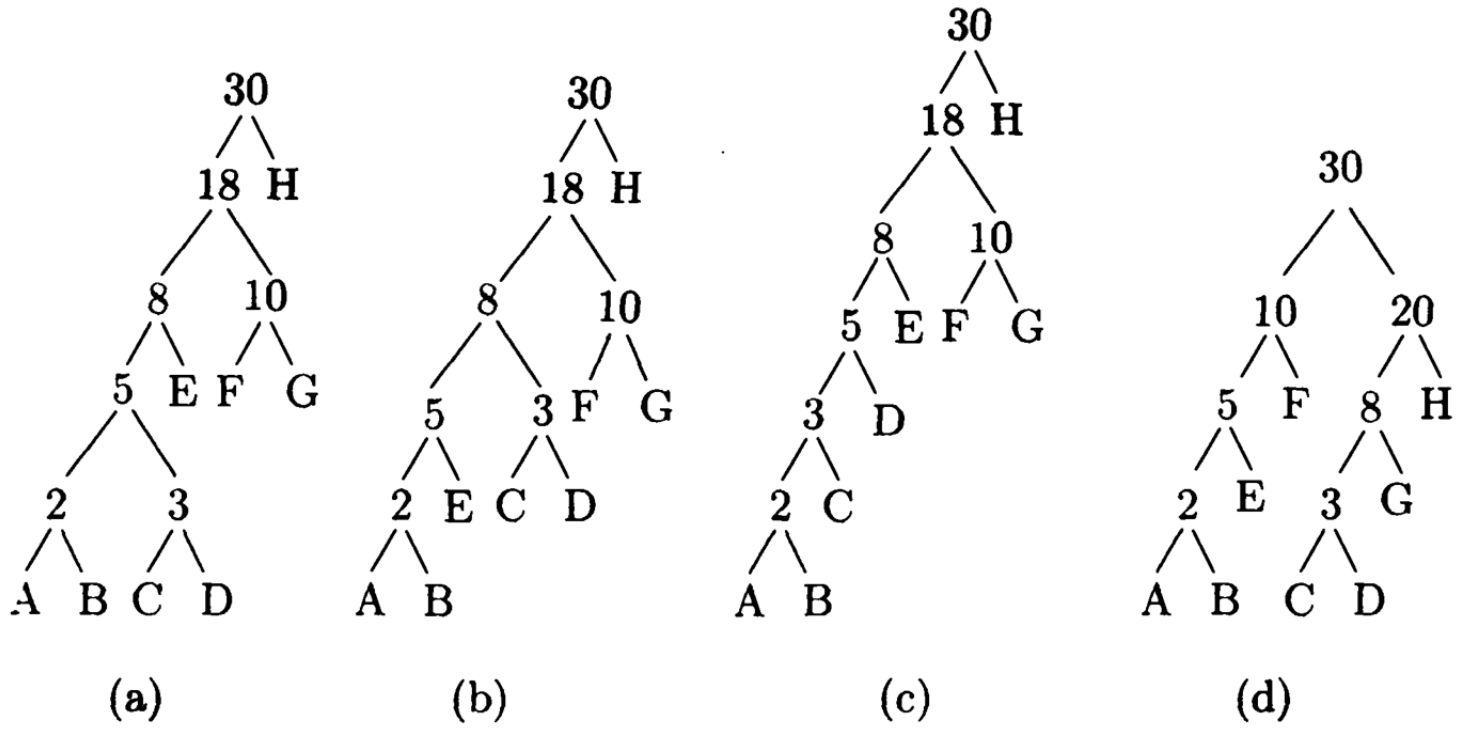
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

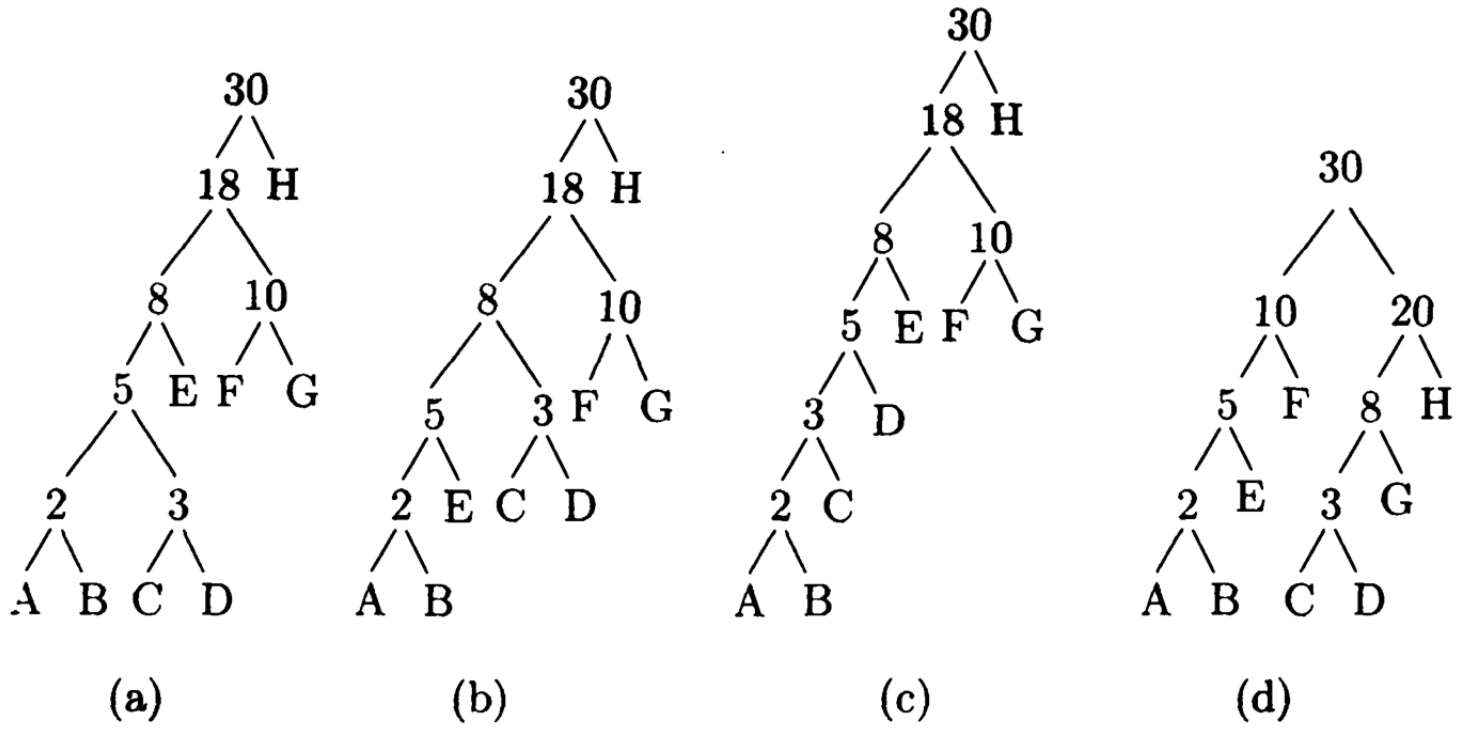
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

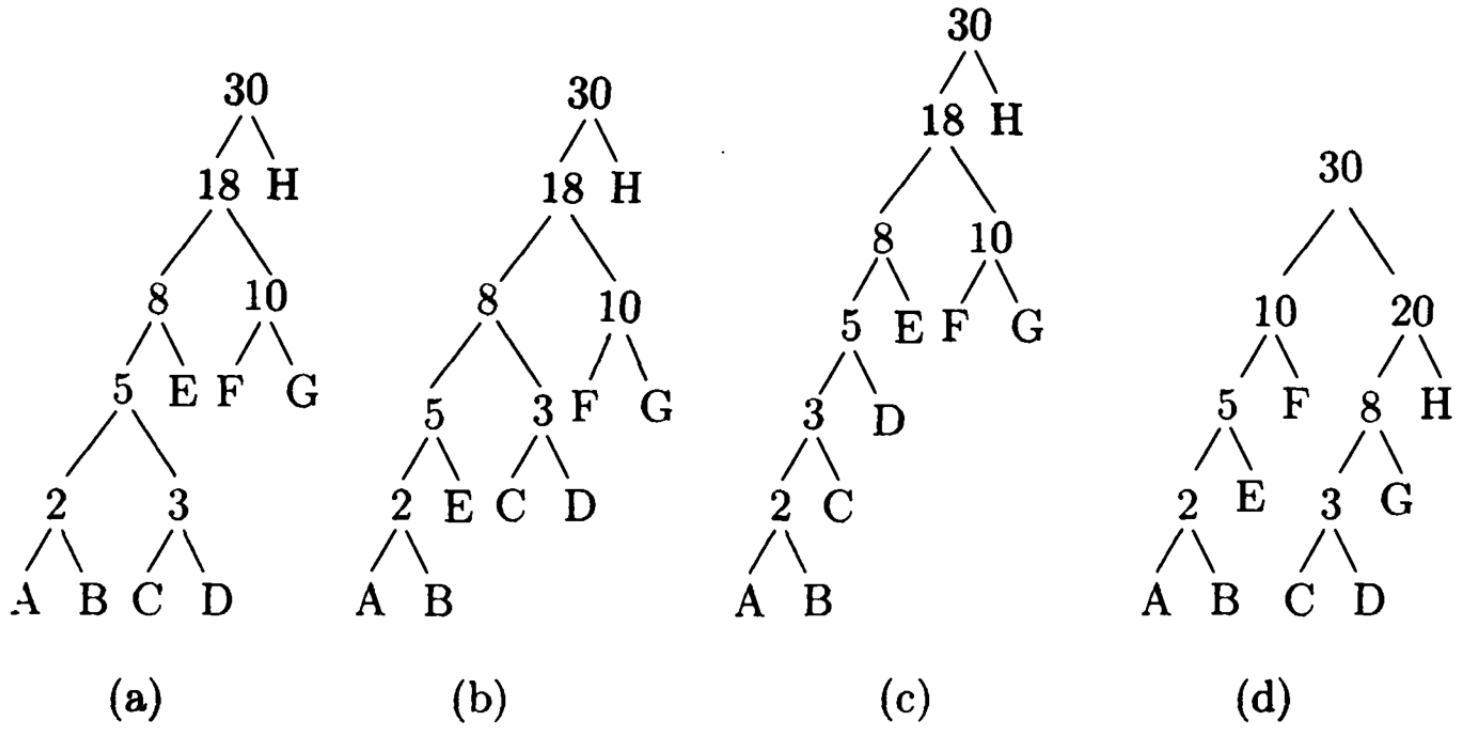
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

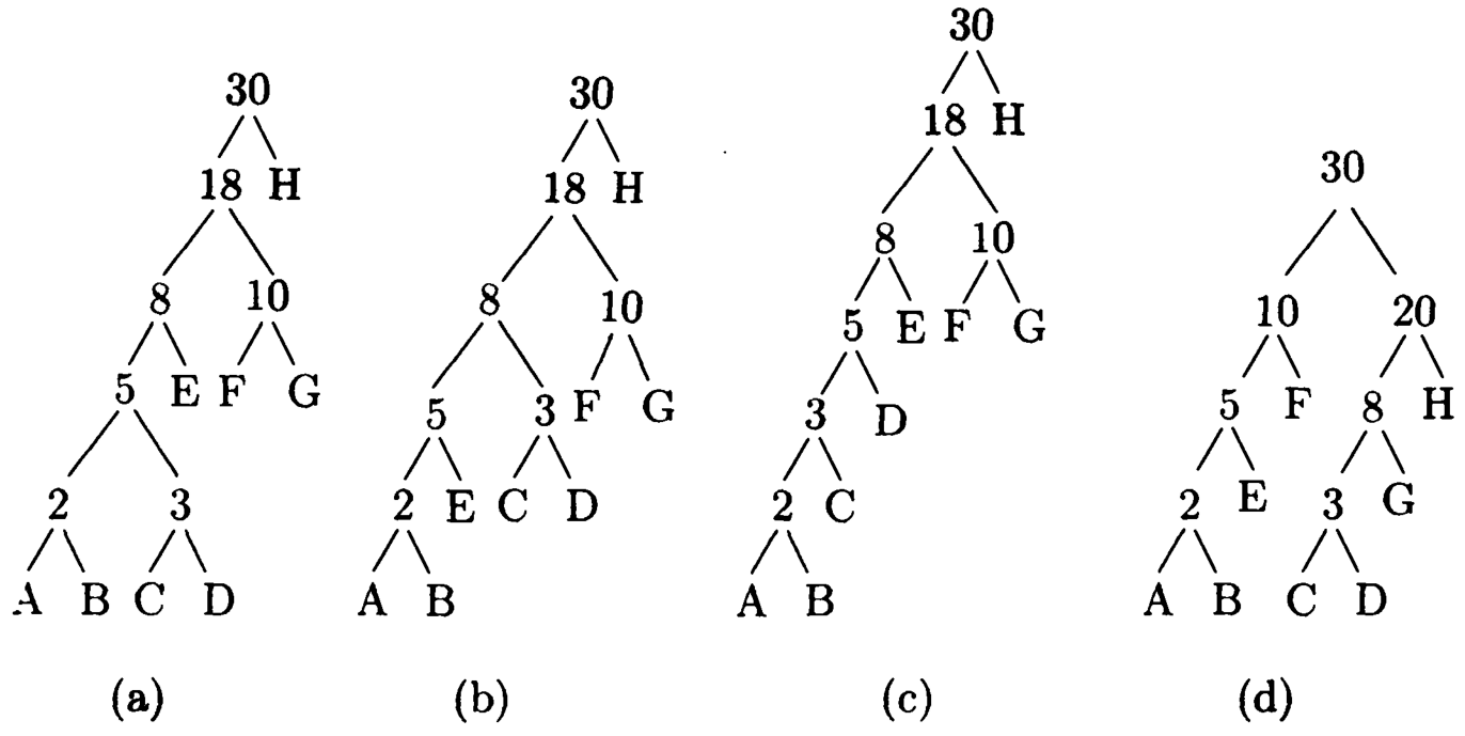
На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Приклад

На рис. (d) зображено інше дерево висоти 4 для восьми символів попереднього прикладу. Наступний аналіз стверджує, що відповідний код змінної довжини — поганий, хоча його довжина менше 4.



Аналіз. Після об'єднання символів A, B, C, D, E, F і G залишаються символи $ABEF$ (з ймовірністю $10/30$), CDG (з ймовірністю $8/30$) та H (з ймовірністю $12/30$). Символи $ABEF$ і CDG мають найменшу ймовірність, а тому їх необхідно було злити в один, але натомість були об'єднані символи CDG і H. Отримане дерево не є деревом Гаффмана.

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

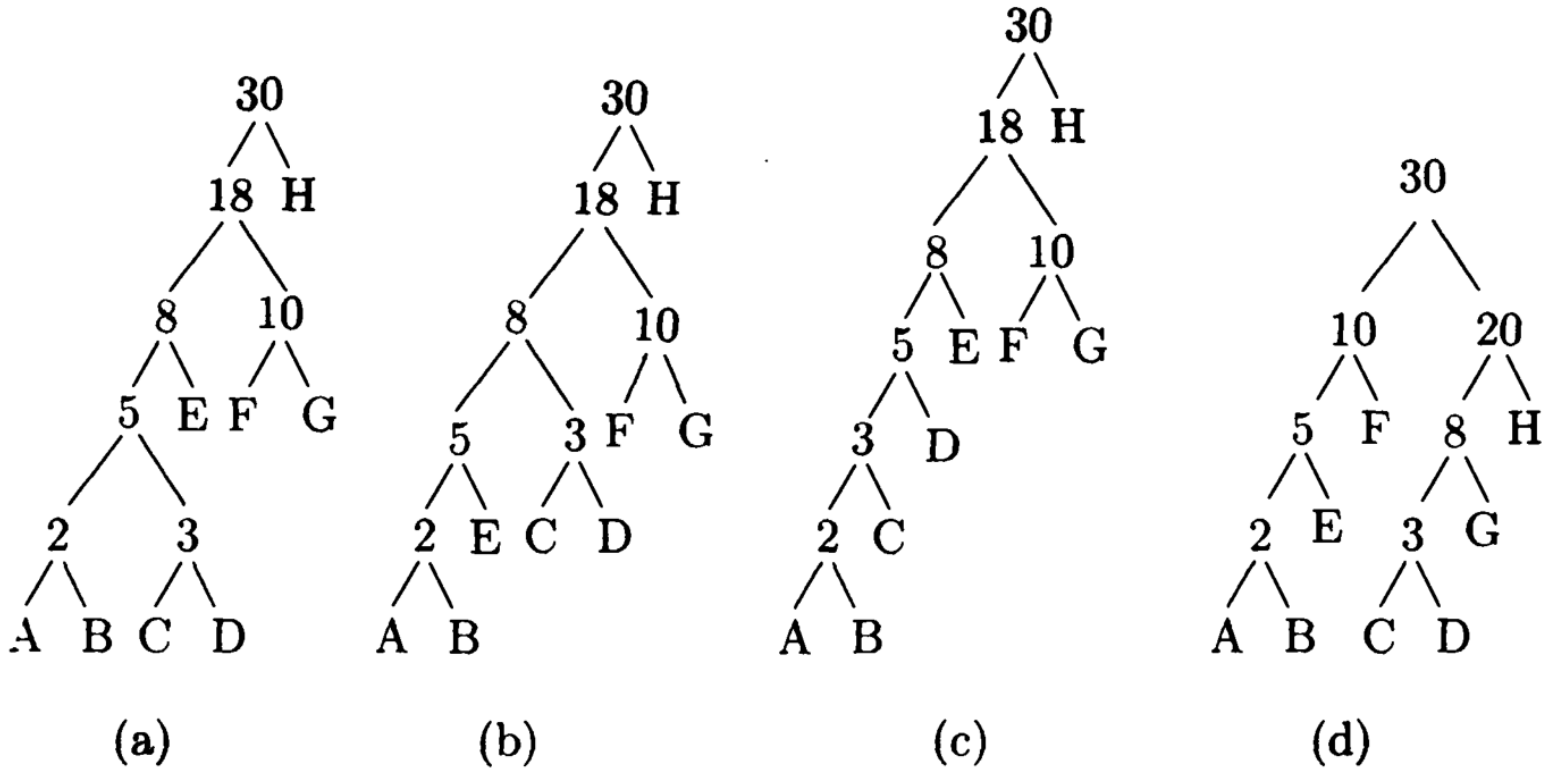
Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

Отже, деяке свавілля у побудові дерева дозволяє отримувати різні коди Гаффмана з однаковою середньою довжиною. Напрошується питання: "Який код Гаффмана, побудований для даного алфавіту, є найкращим?" Відповідь буде простою, хоча й неочевидною: найкращим буде код із найменшою *дисперсією*.

Дисперсія показує, наскільки сильно відхиляються довжини індивідуальних кодів від їхньої середньої величини (це поняття роз'яснюється в будь-якому підручнику зі статистики).

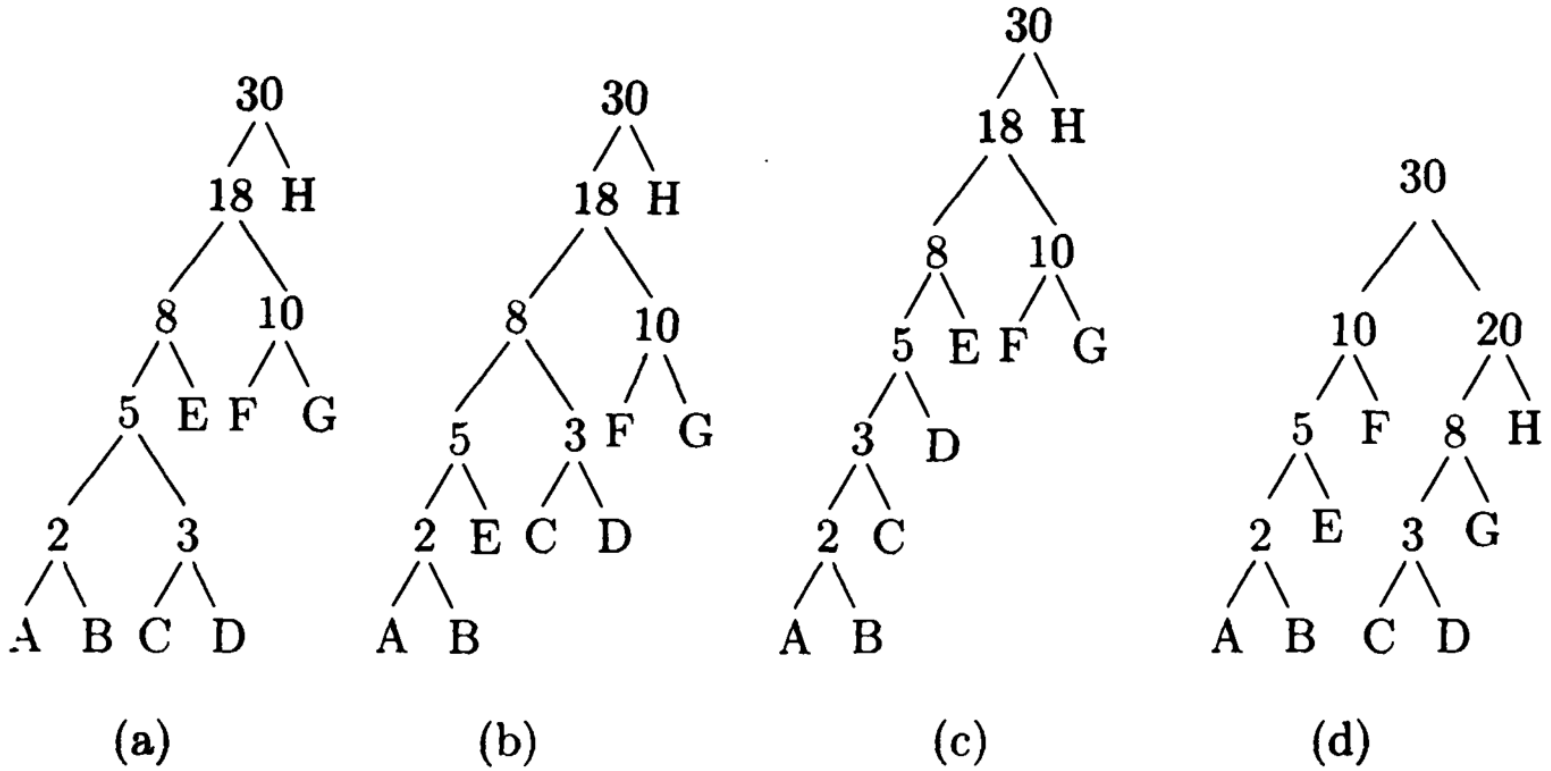


Дисперсія коду з рис. (a) дорівнює

$$0.4(1-2.2)^2+0.2(2-2.2)^2+0.2(3-2.2)^2+0.1(4-2.2)^2+0.1(4-2.2)^2 = 1.36,$$

коду з рис. (b)

$$0.4(2-2.2)^2+0.2(2-2.2)^2+0.2(2-2.2)^2+0.1(3-2.2)^2+0.1(3-2.2)^2 = 0.16.$$

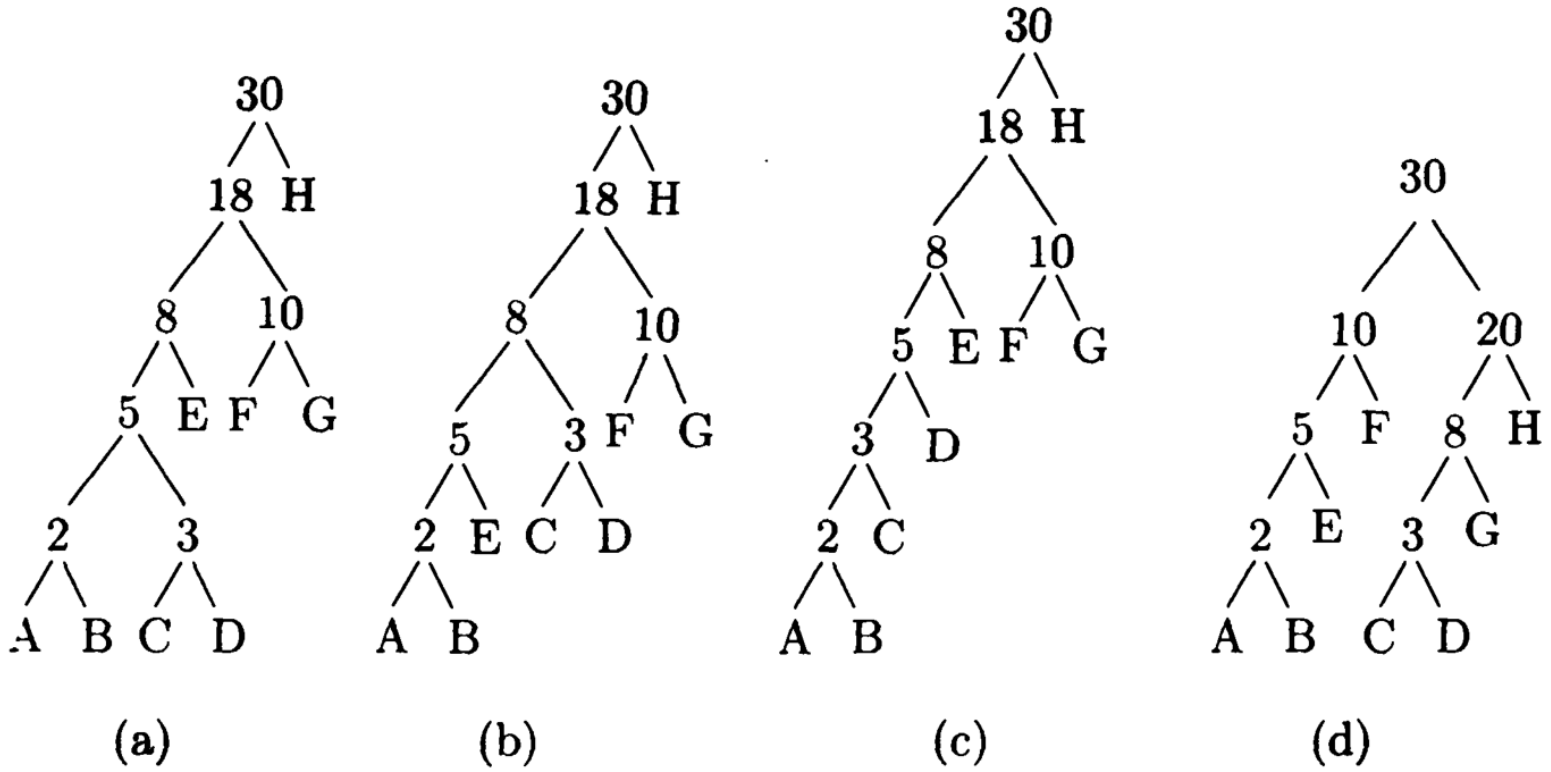


Дисперсія коду з рис. (a) дорівнює

$$0.4(1-2.2)^2+0.2(2-2.2)^2+0.2(3-2.2)^2+0.1(4-2.2)^2+0.1(4-2.2)^2 = 1.36,$$

коду з рис. (b)

$$0.4(2-2.2)^2+0.2(2-2.2)^2+0.2(2-2.2)^2+0.1(3-2.2)^2+0.1(3-2.2)^2 = 0.16.$$

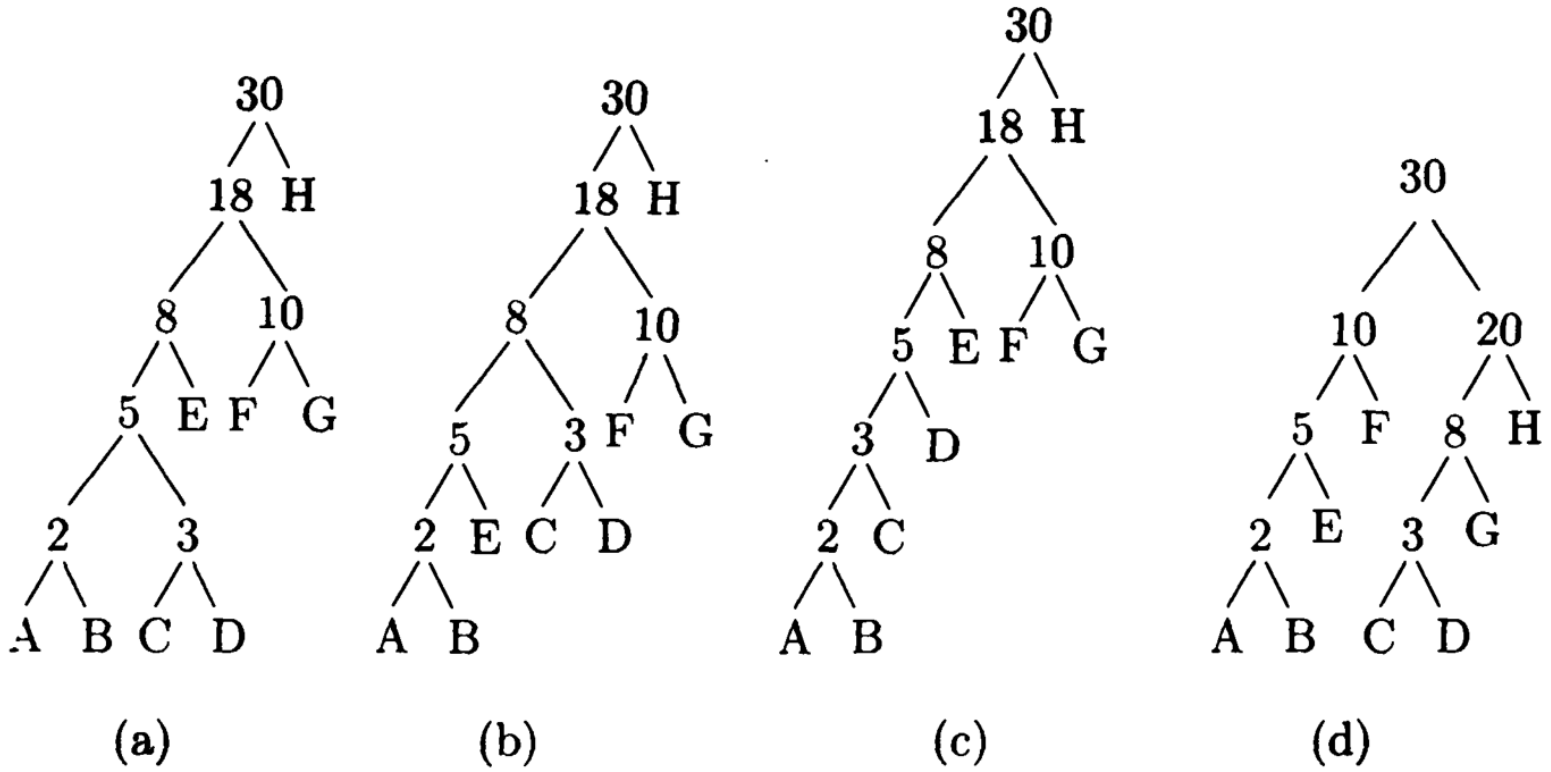


Дисперсія коду з рис. (a) дорівнює

$$0.4(1-2.2)^2+0.2(2-2.2)^2+0.2(3-2.2)^2+0.1(4-2.2)^2+0.1(4-2.2)^2 = 1.36,$$

коду з рис. (b)

$$0.4(2-2.2)^2+0.2(2-2.2)^2+0.2(2-2.2)^2+0.1(3-2.2)^2+0.1(3-2.2)^2 = 0.16.$$

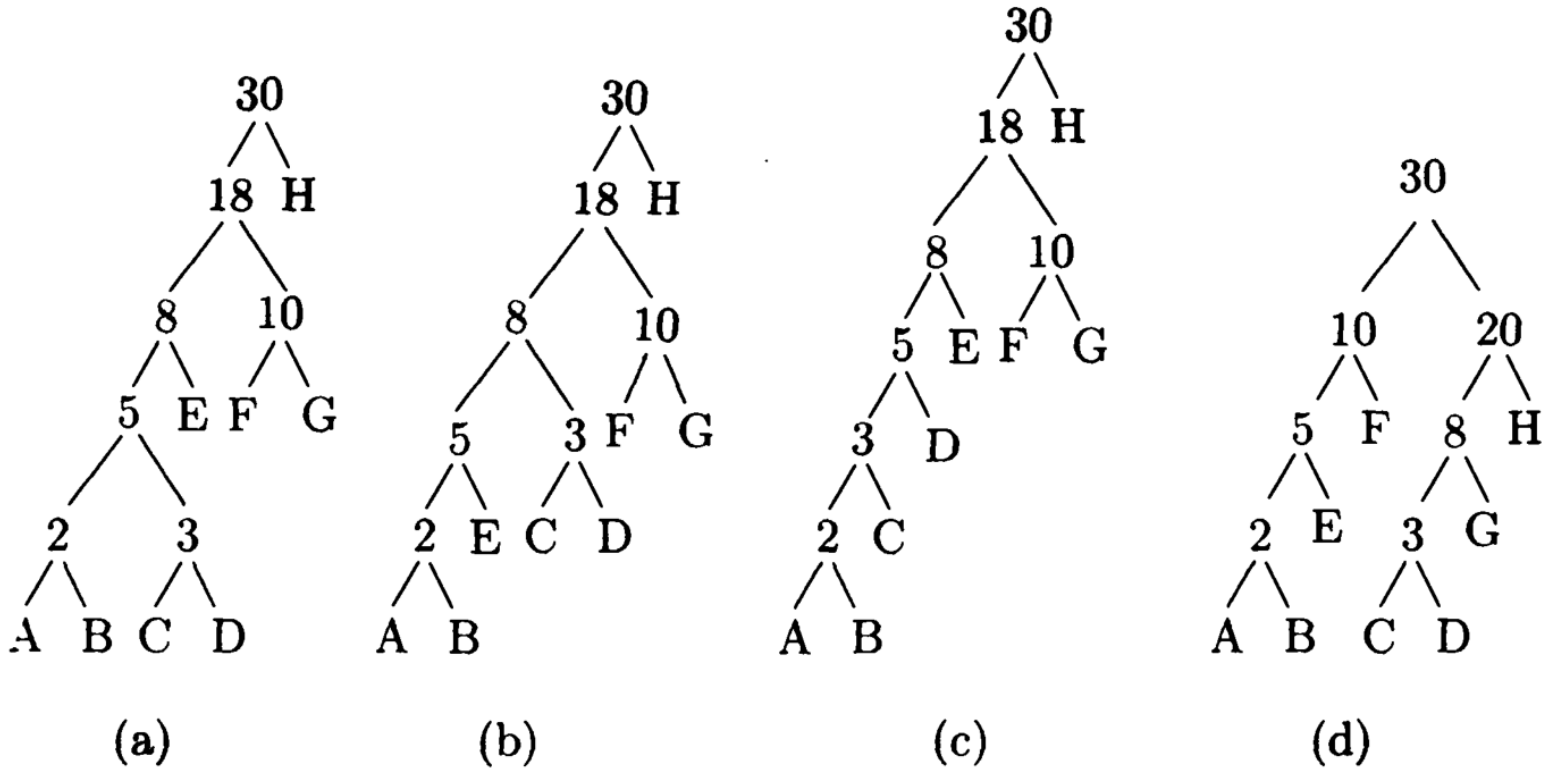


Дисперсія коду з рис. (a) дорівнює

$$0.4(1-2.2)^2+0.2(2-2.2)^2+0.2(3-2.2)^2+0.1(4-2.2)^2+0.1(4-2.2)^2 = 1.36,$$

коду з рис. (b)

$$0.4(2-2.2)^2+0.2(2-2.2)^2+0.2(2-2.2)^2+0.1(3-2.2)^2+0.1(3-2.2)^2 = 0.16.$$

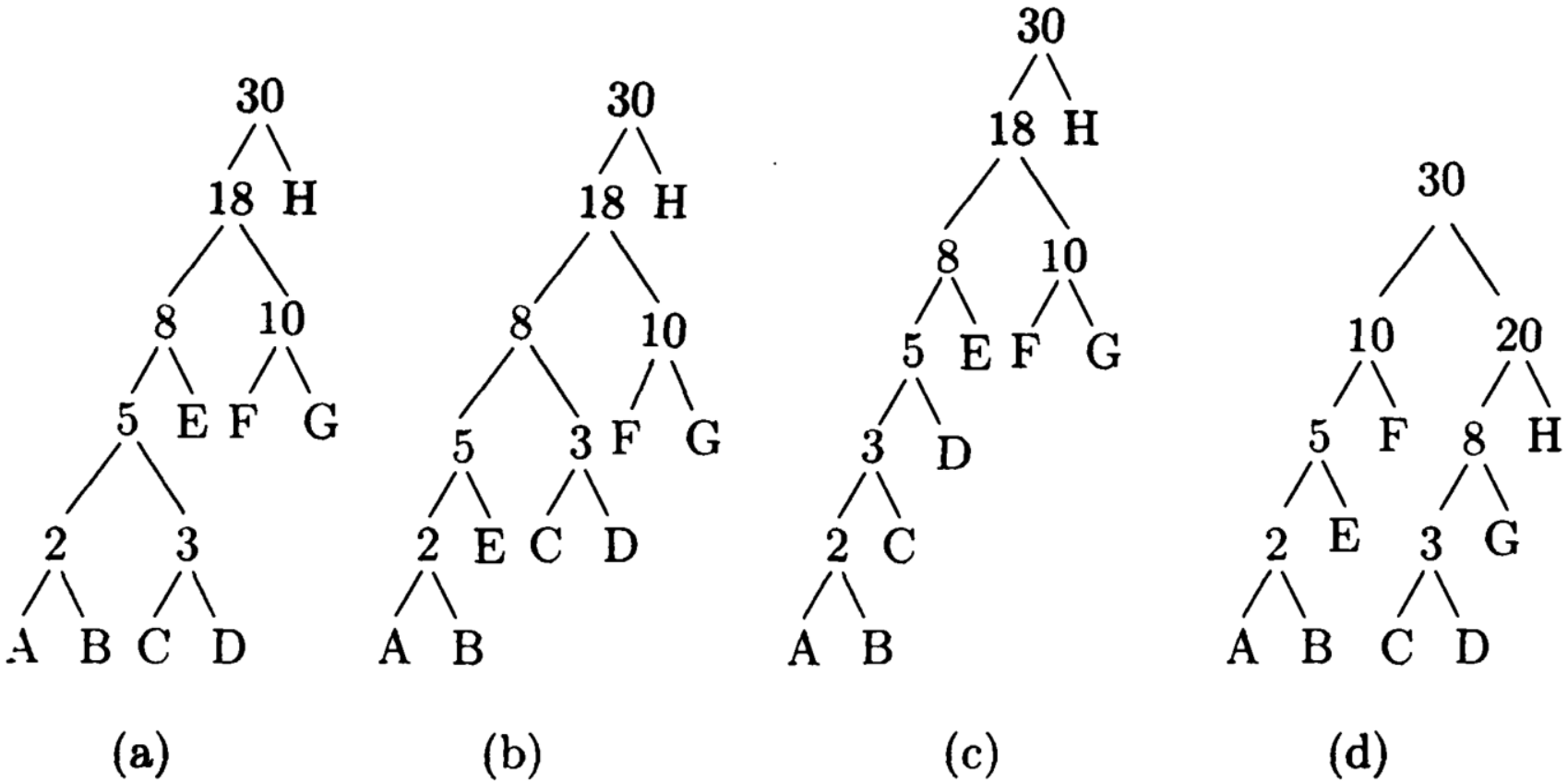


Дисперсія коду з рис. (a) дорівнює

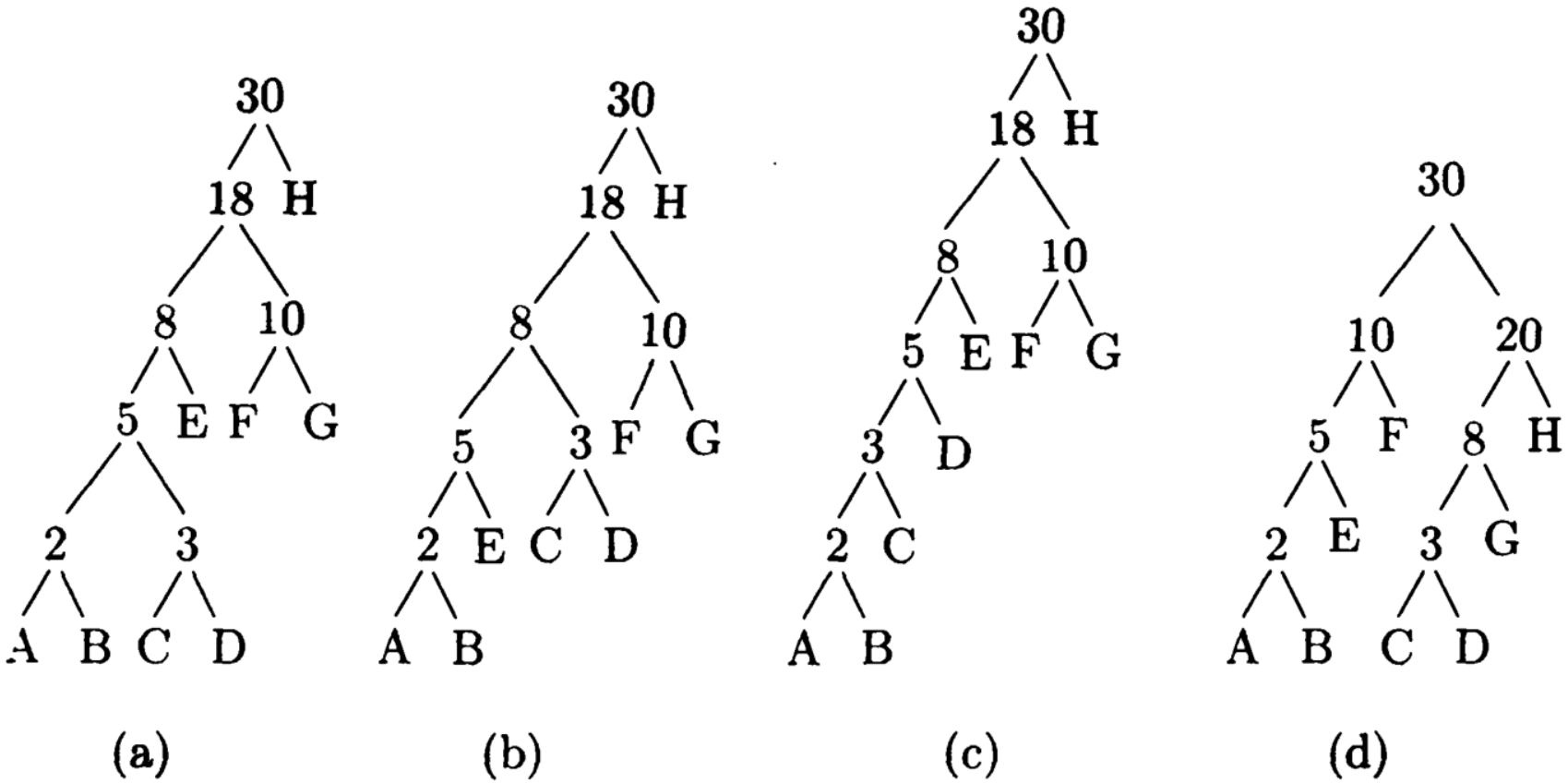
$$0.4(1-2.2)^2+0.2(2-2.2)^2+0.2(3-2.2)^2+0.1(4-2.2)^2+0.1(4-2.2)^2 = 1.36,$$

коду з рис. (b)

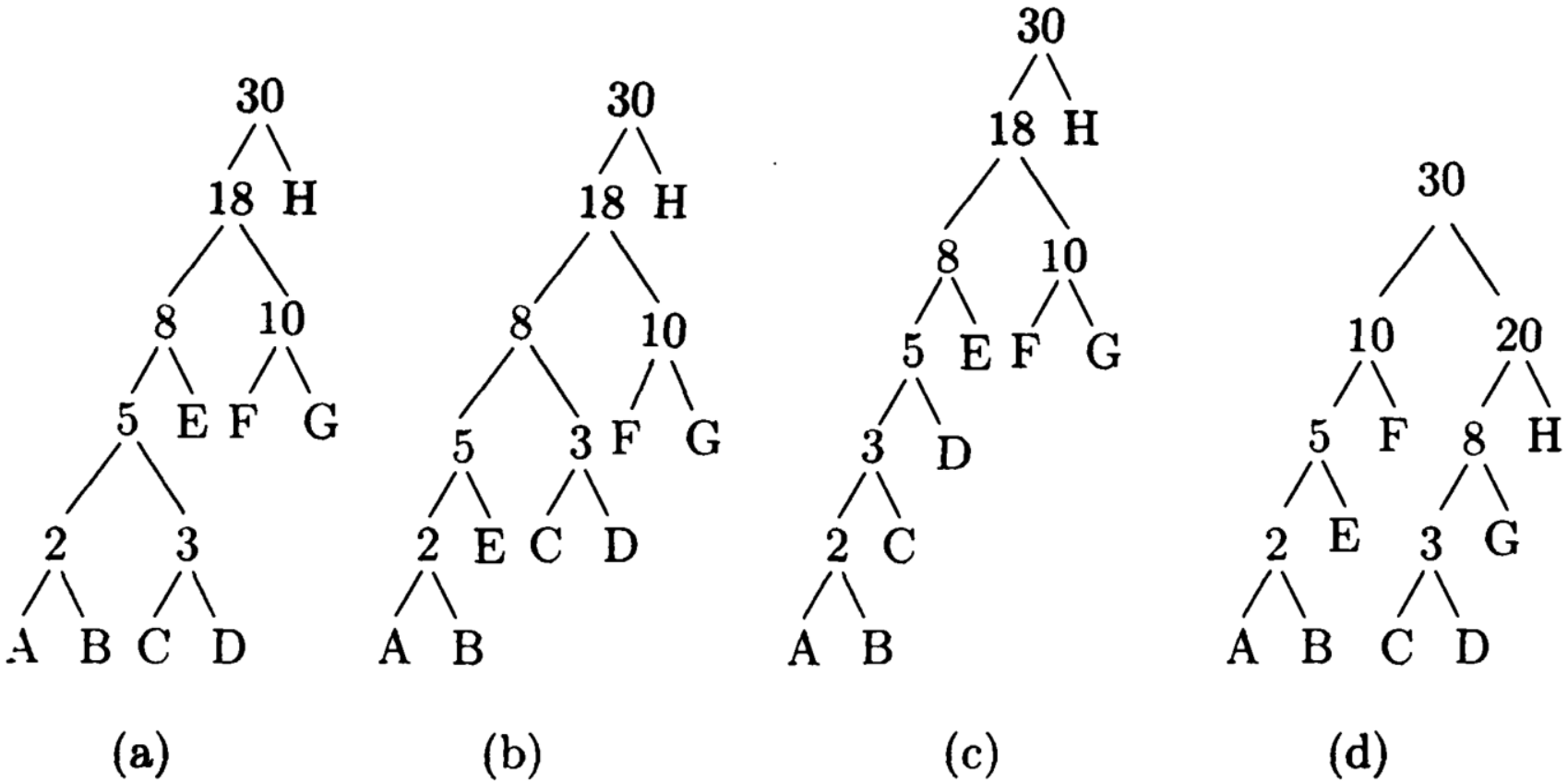
$$0.4(2-2.2)^2+0.2(2-2.2)^2+0.2(2-2.2)^2+0.1(3-2.2)^2+0.1(3-2.2)^2 = 0.16.$$



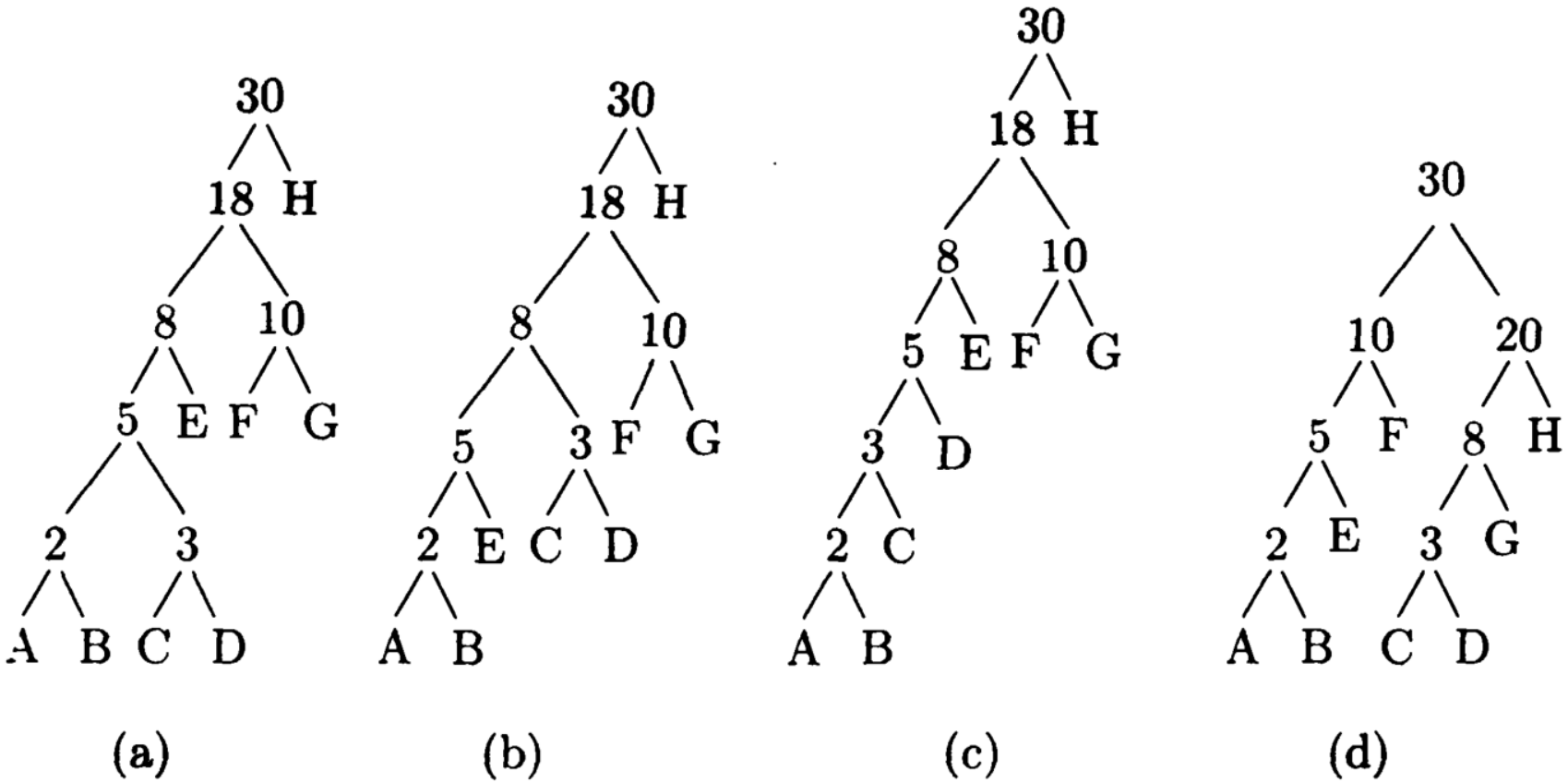
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



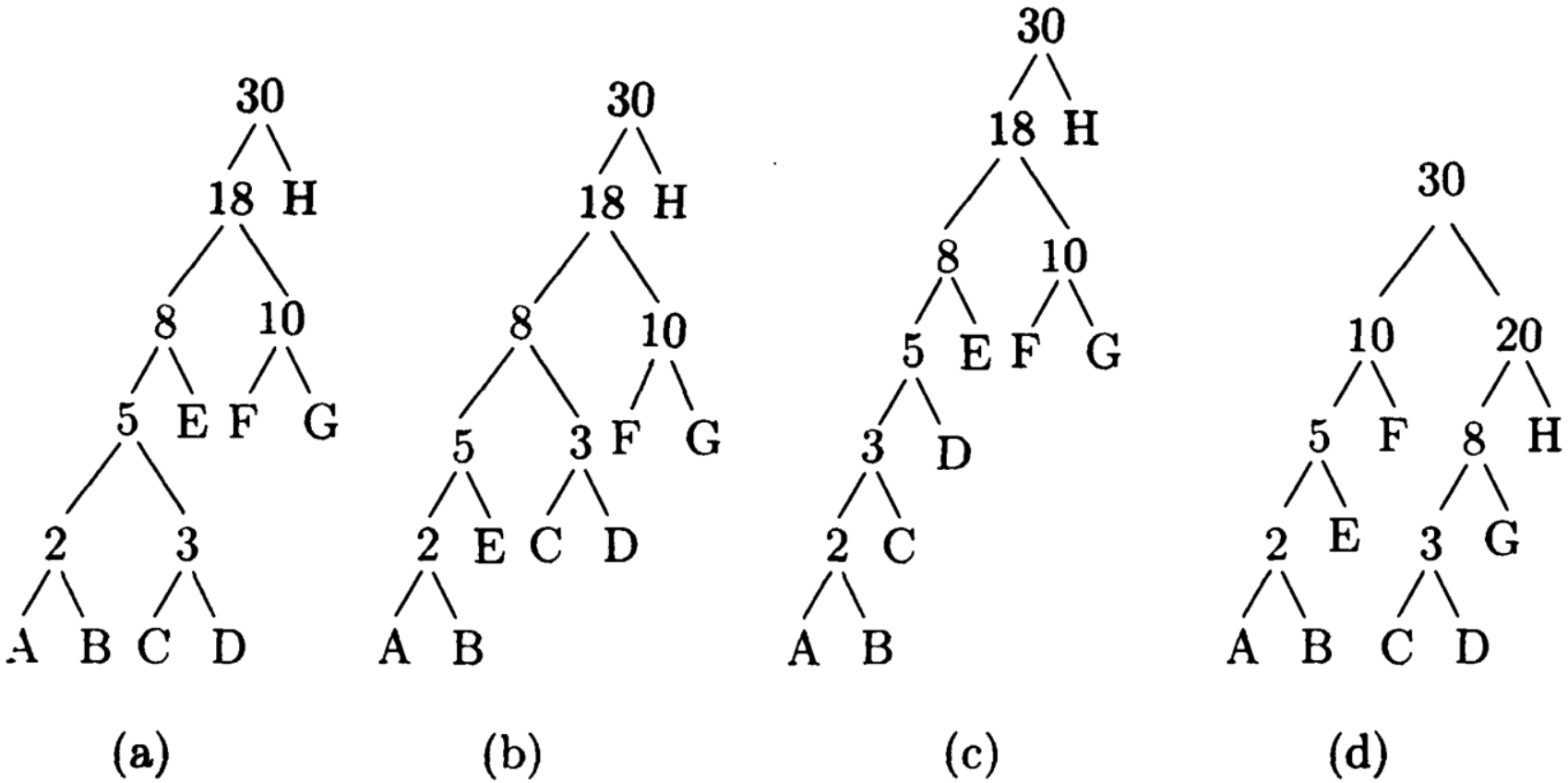
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



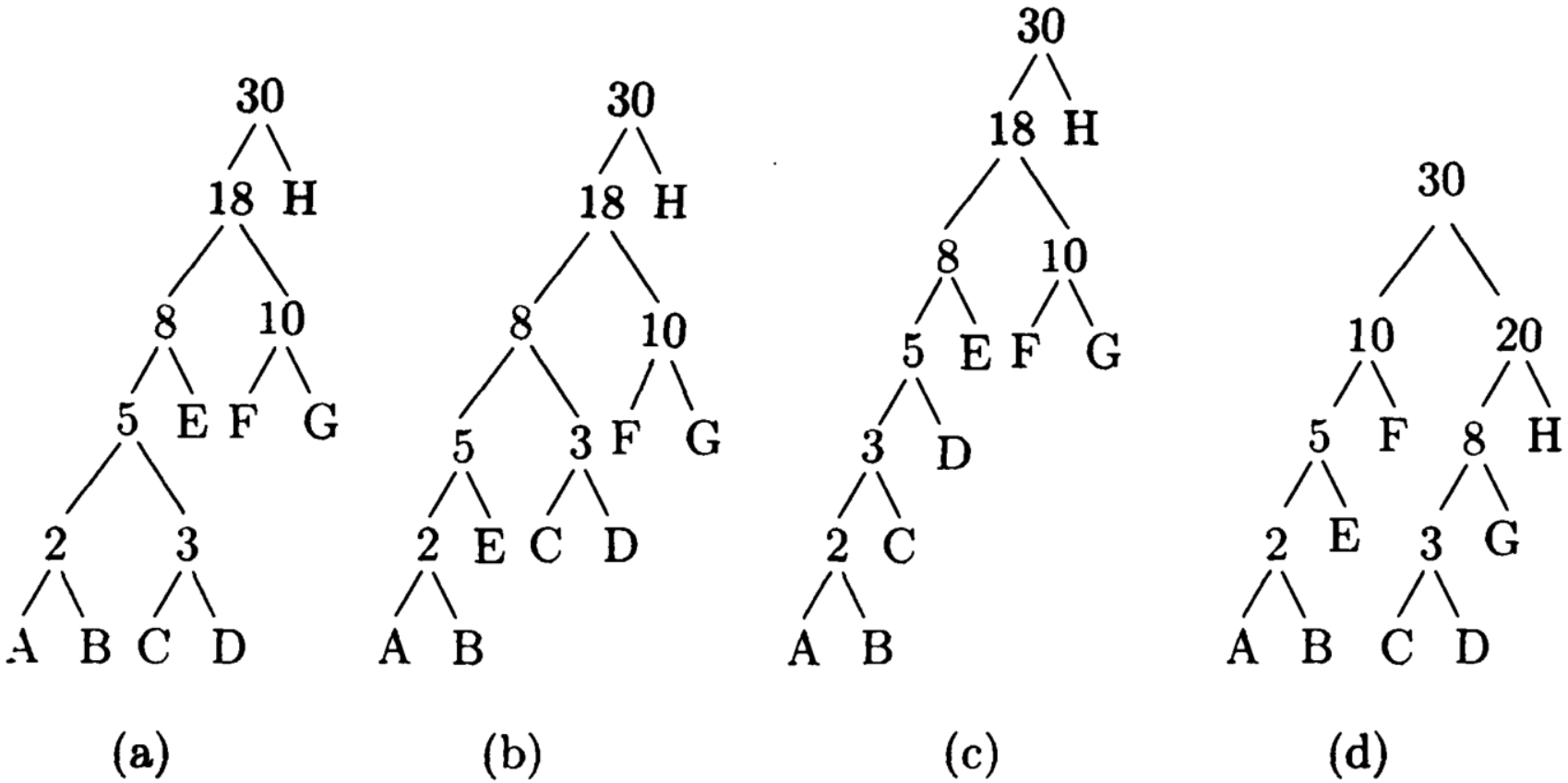
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



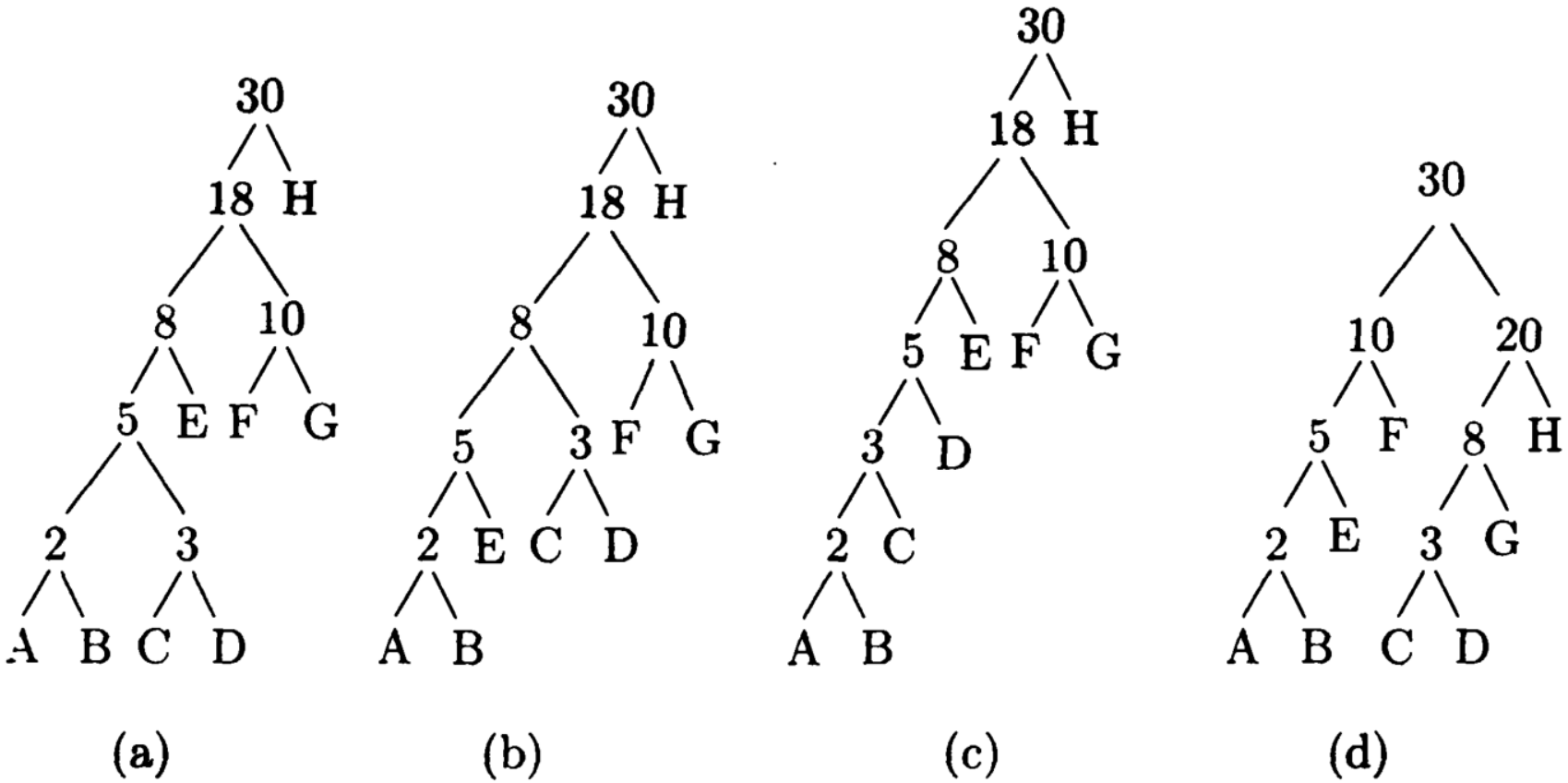
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



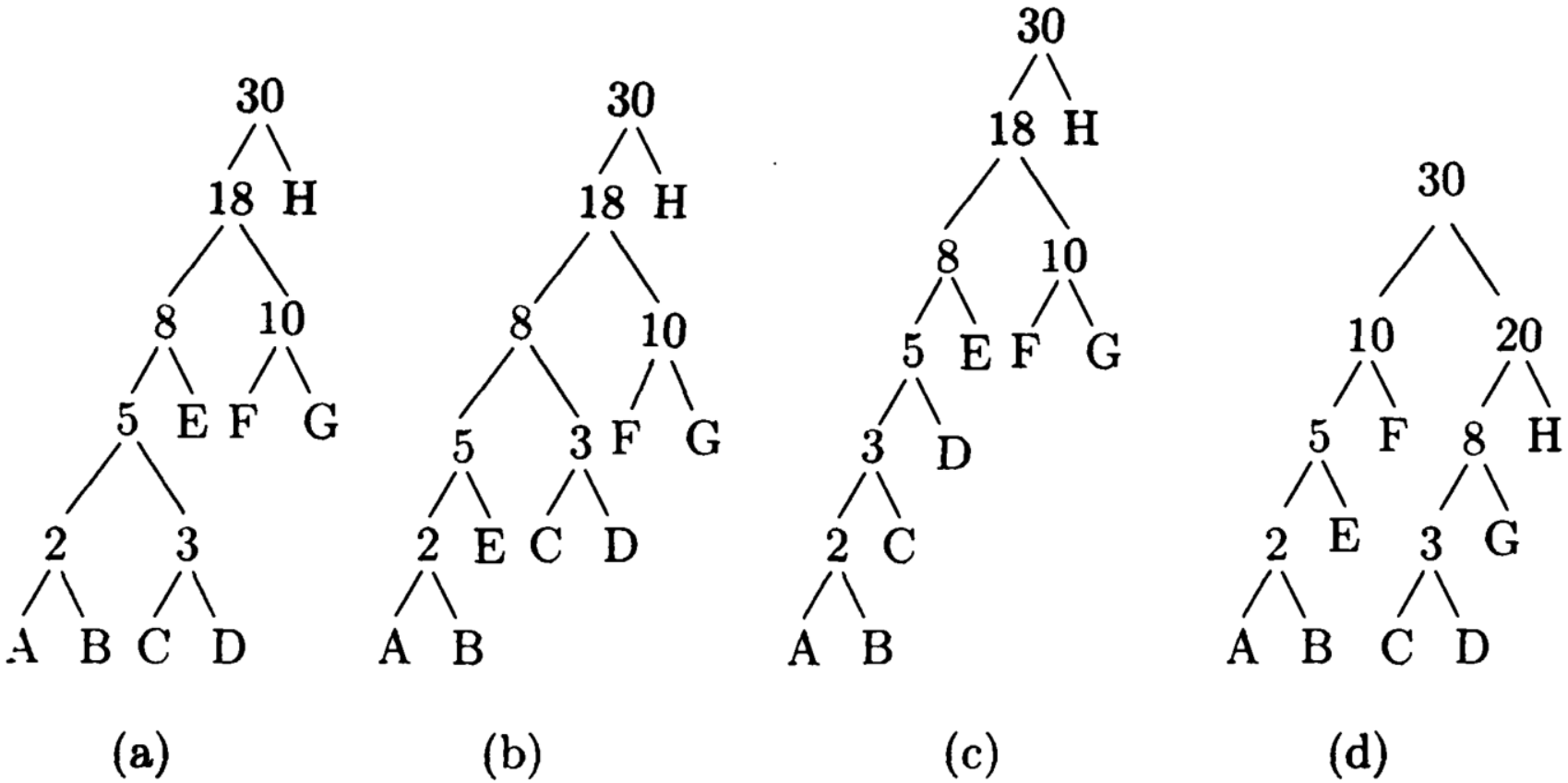
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



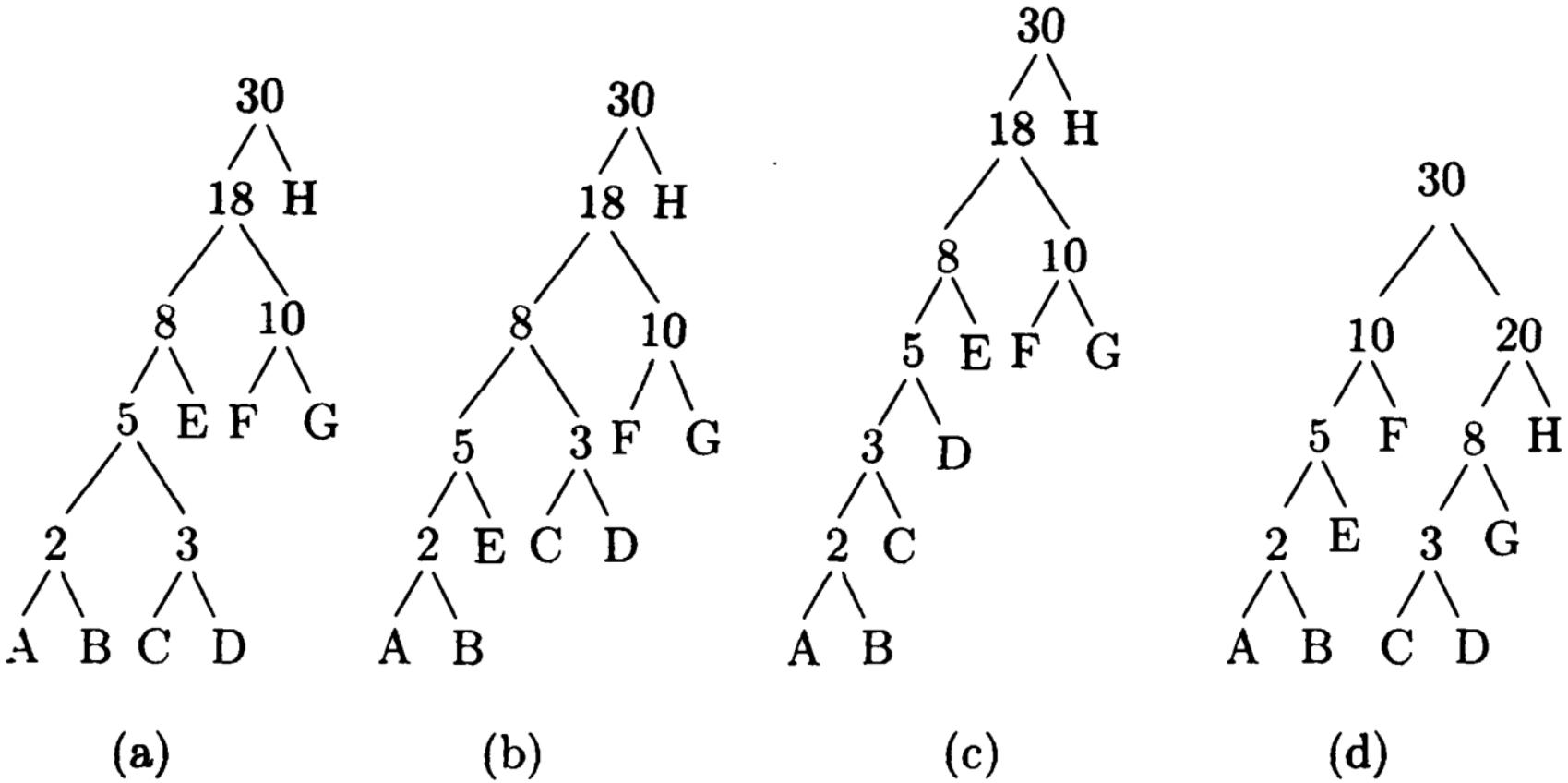
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



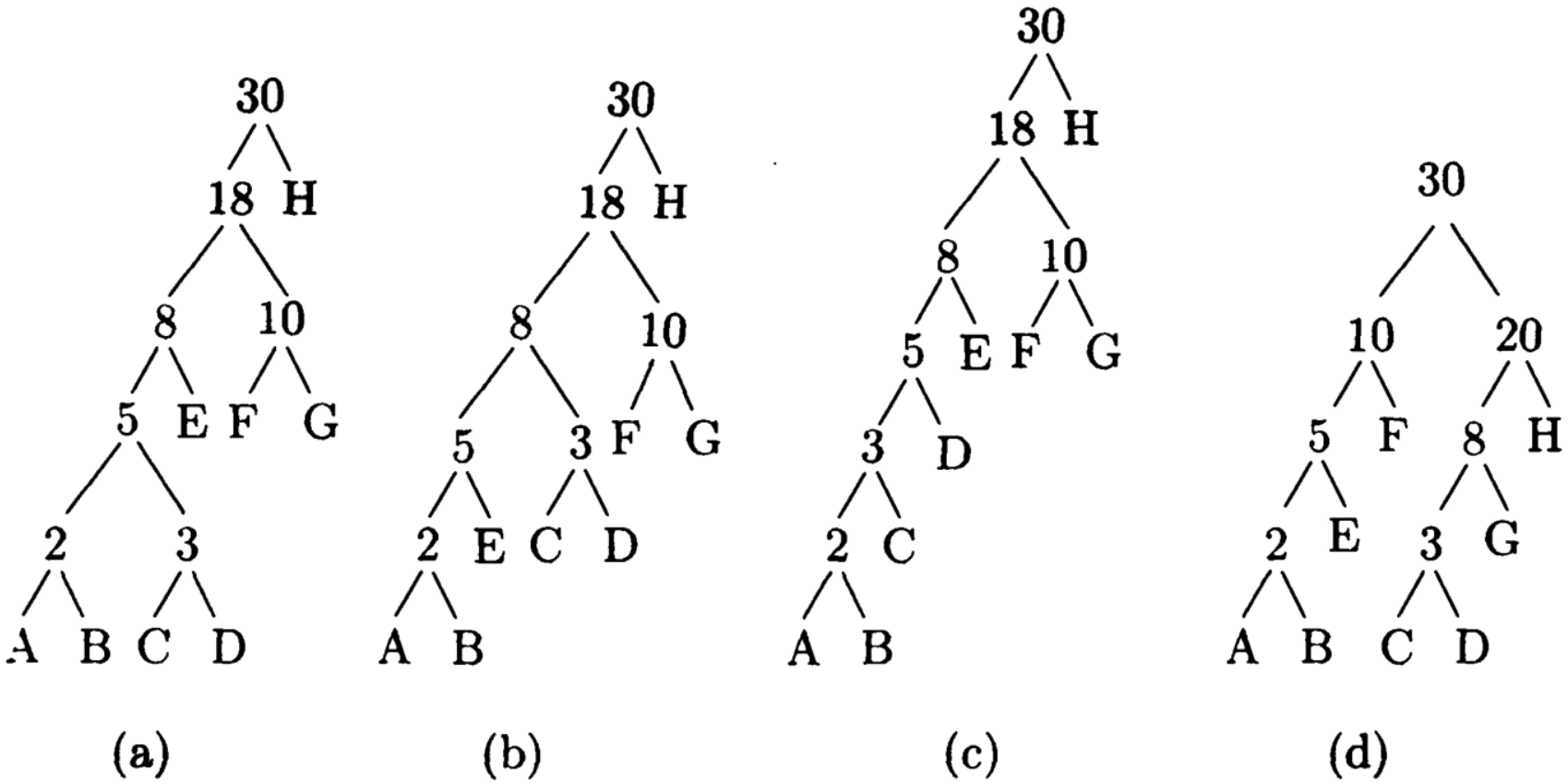
Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.



Код з рис. (b) є кращим (це буде пояснено нижче). Уважний погляд на дерева показує, як вибрати одне, потрібне нам. На дереві рис. (a) символ a_{45} зливається із символом a_3, тоді як на рис. (b) він зливається з a_1. Правило буде таке: коли на дереві є більше двох вузлів із найменшою ймовірністю, то слід поєднувати символи з найбільшою та найменшою ймовірністю; це скорочує загальну дисперсію коду.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Якщо кодер просто записує стислий файл на диск, то дисперсія коду не має значення. Коди Гаффмана з малою дисперсією кращі лише у випадку, якщо кодер буде передавати цей стислий файл по лініях зв'язку. У цьому випадку код з великою дисперсією змушує кодер генерувати біти зі змінною швидкістю. Зазвичай дані передаються каналами зв'язку з постійною швидкістю, тому кодер використовуватиме буфер. Біти стисненого файлу поміщаються в буфер у міру їхньої генерації та подаються в канал з постійною швидкістю передачі. Легко бачити, що код з нульовою дисперсією подаватиметься в буфер з постійною швидкістю, тому знадобиться короткий буфер, а велика дисперсія коду вимагатиме використання довгого буфера.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує* $\lceil \log_2 P_i \rceil$. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.

Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

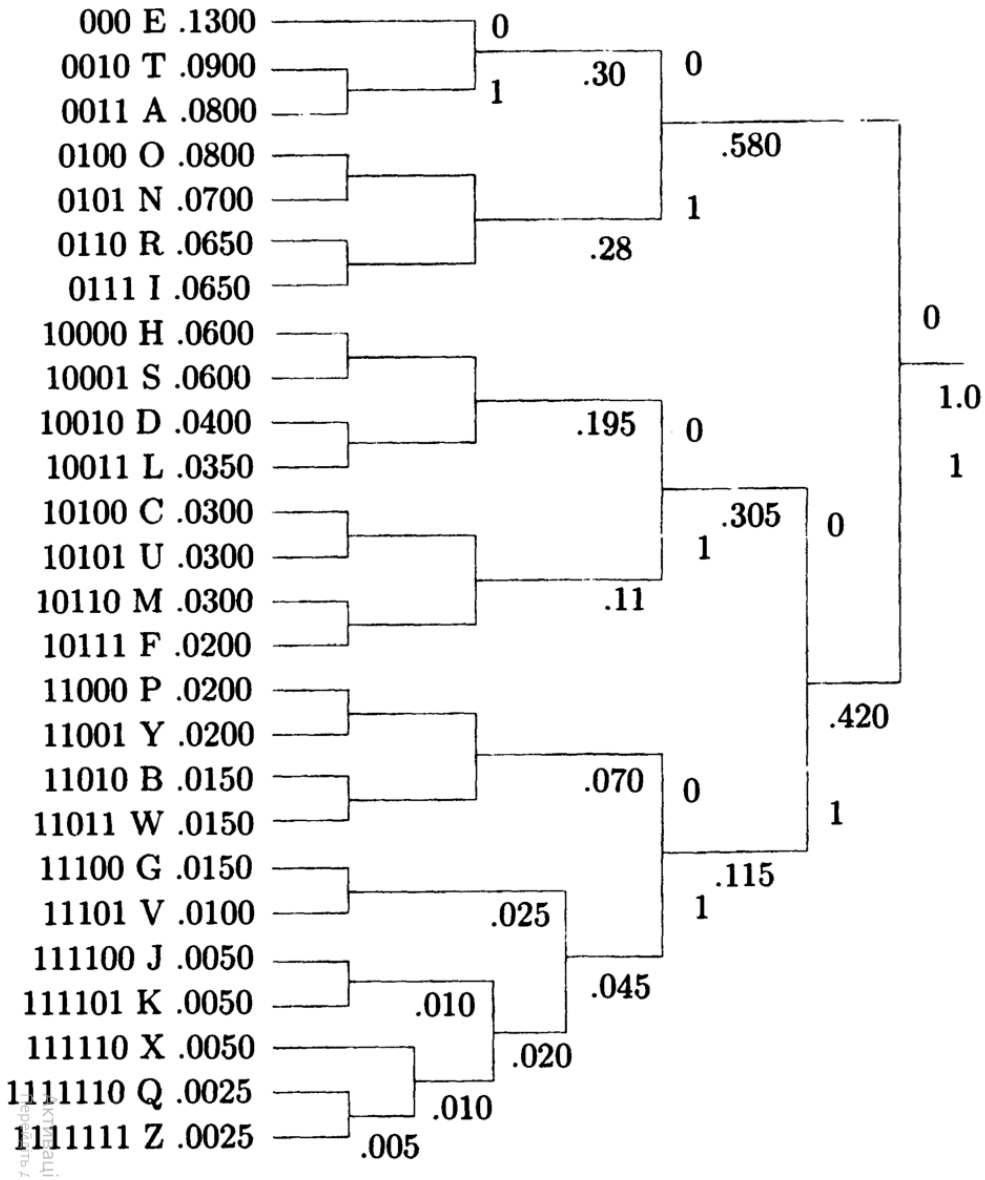
У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Таке твердження можна іноді знайти в літературі зі стиснення інформації: *довжина коду Гаффмана символу a_i з ймовірністю P_i завжди не перевищує $\lceil \log_2 P_i \rceil$*. Насправді, незважаючи на справедливість цього твердження в багатьох прикладах, у загальному випадку воно не є вірним. Гай Блелок навів приклад коду, наведеного в табл.

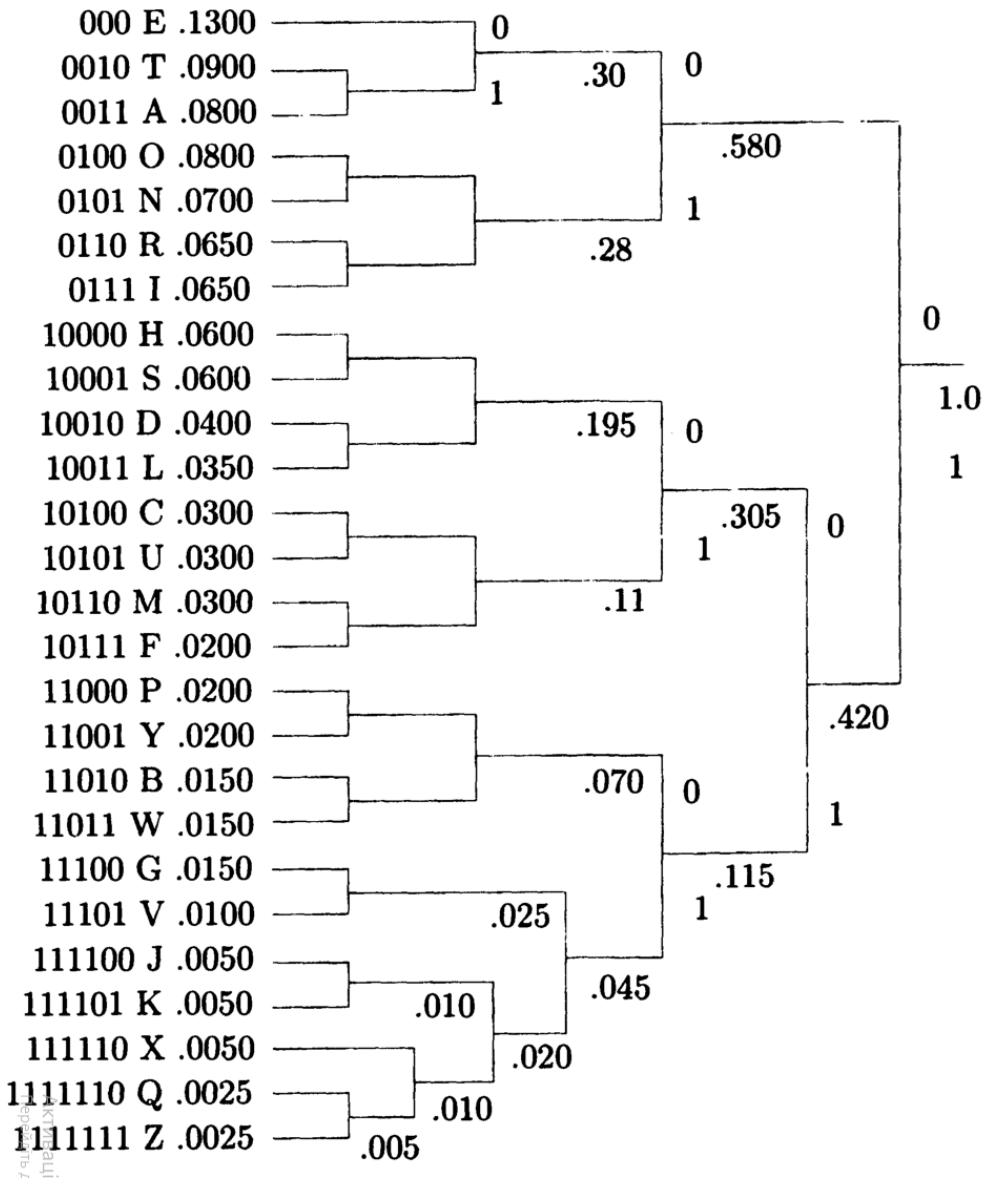
	P_i	Код	$-\log_2 P_i$	$\lceil -\log_2 P_i \rceil$
	0.01	000	6.644	7
*	0.30	001	1.737	2
	0.34	01	1.556	2
	0.35	1	1.515	2

У другому рядку цієї таблиці стоїть символ із кодом довжини 3 біти, у той час як $\lceil -\log_2 0.3 \rceil = \lceil 1.737 \rceil = 2$.
Довжина коду символу a_i, звичайно, залежить від його ймовірності P_i. Однак вона також залежить від розміру алфавіту. У великому алфавіті ймовірності символів малі, тому коди Гаффмана мають більшу довжину. У маленькому алфавіті спостерігається зворотна картина. Інтуїтивно це зрозуміло, оскільки для малого алфавіту потрібно лише декілька кодів, тому всі вони короткі, а великому алфавіту необхідно багато кодів і деякі з них мають бути довгими.

Статистичні методи стиснення зображень. Кодування Гаффмана

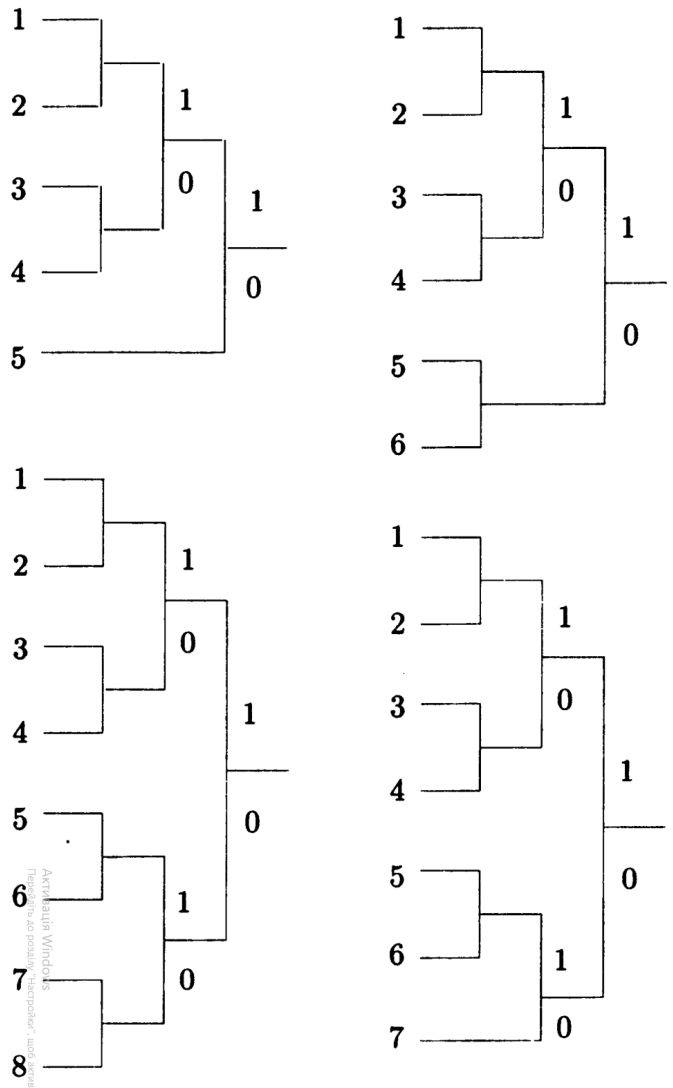


На рис. зображено код Гаффмана для всіх 26 літер латинського алфавіту.

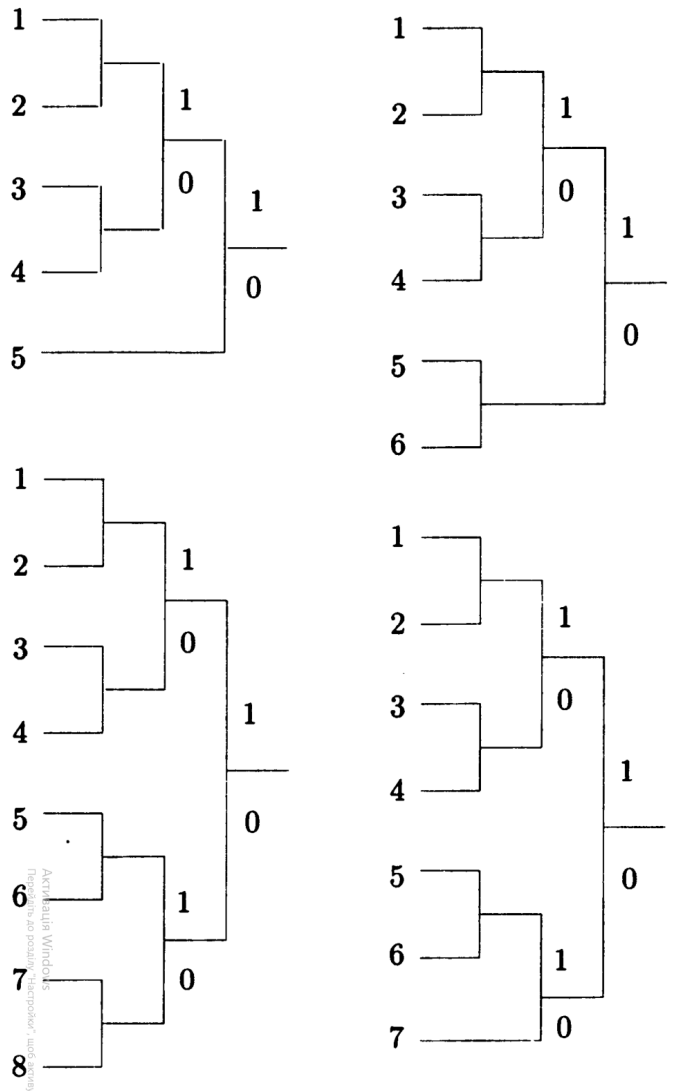


На рис. зображено код Гаффмана для всіх 26 літер латинського алфавіту.

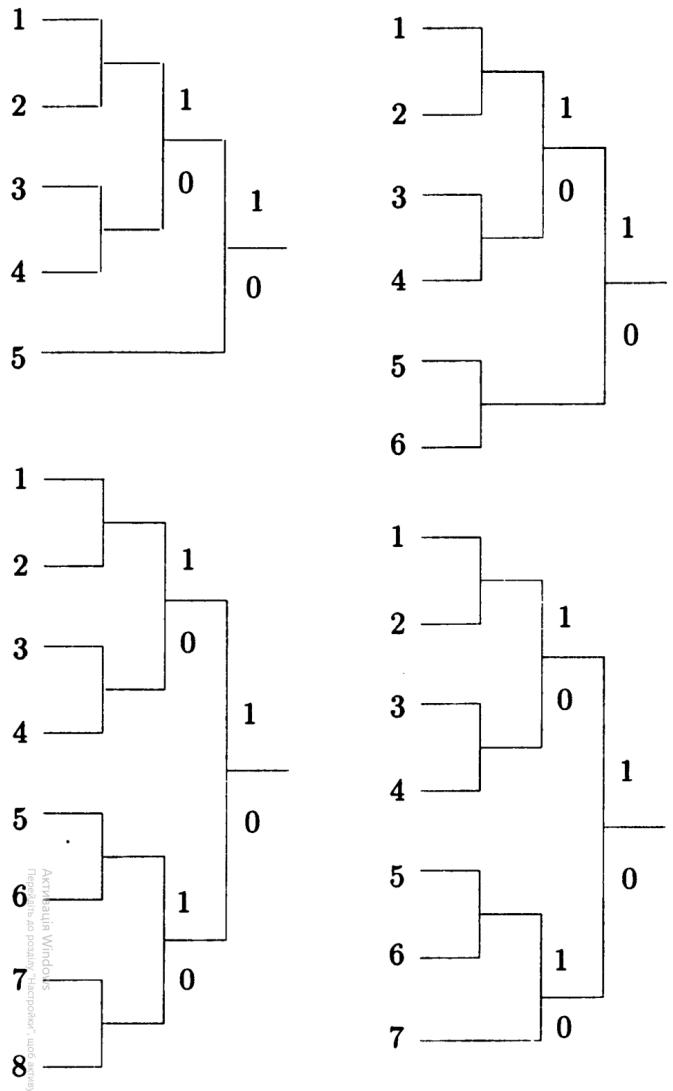
Статистичні методи стиснення зображень. Кодування Гаффмана



Випадок алфавіту, в якому символи мають однакову ймовірність, особливо цікавий. На рис. наведено коди Гаффмана для алфавіту з 5, 6, 7 та 8 символами, які мають однакову ймовірність.



Випадок алфавіту, в якому символи мають однакову ймовірність, особливо цікавий. На рис. наведено коди Гаффмана для алфавіту з 5, 6, 7 та 8 символами, які мають однакову ймовірність.



Випадок алфавіту, в якому символи мають однакову ймовірність, особливо цікавий. На рис. наведено коди Гаффмана для алфавіту з 5, 6, 7 та 8 символами, які мають однакову ймовірність.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Якщо розмір алфавіту число n є степенем числа 2, то отримуємо просто коди фіксованої довжини. В інших випадках коди дуже близькі до кодів із фіксованою довжиною. Це означає, що використання кодів змінної довжини не дає жодних переваг. У табл.

n	p	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Сер.довж.	Дисперсія
5	0.200	111	110	101	100	0				2.6	0.64
6	0.167	111	110	101	100	01	00			2.672	0.2227
7	0.167	111	110	101	100	011	010	00		2.86	0.1226
8	0.167	111	110	101	100	011	010	001	000	3	0

наведено коди, їхні середні довжини та дисперсії.

Той факт, що дані з рівноймовірними символами не стискаються методом Гаффмана, може означати, що рядки таких символів є цілком випадковими. Однак, є приклади рядків, у яких всі символи рівноймовірні, але не є випадковими, і їх можна стискати. Хорошим прикладом є послідовність

$$a_1a_1\cdots a_1a_2a_2\cdots a_2a_3a_3\cdots,$$

у якій кожен символ зустрічається довгими серіями. Такий рядок можна стиснути методом RLE, але не методом Гаффмана. Букосполучення RLE означає “run-length encoding”, тобто “кодування довжин серій”. Цей простий метод сам по собі мало ефективний, але його можна використовувати в алгоритмах стиснення з багатьма етапами.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

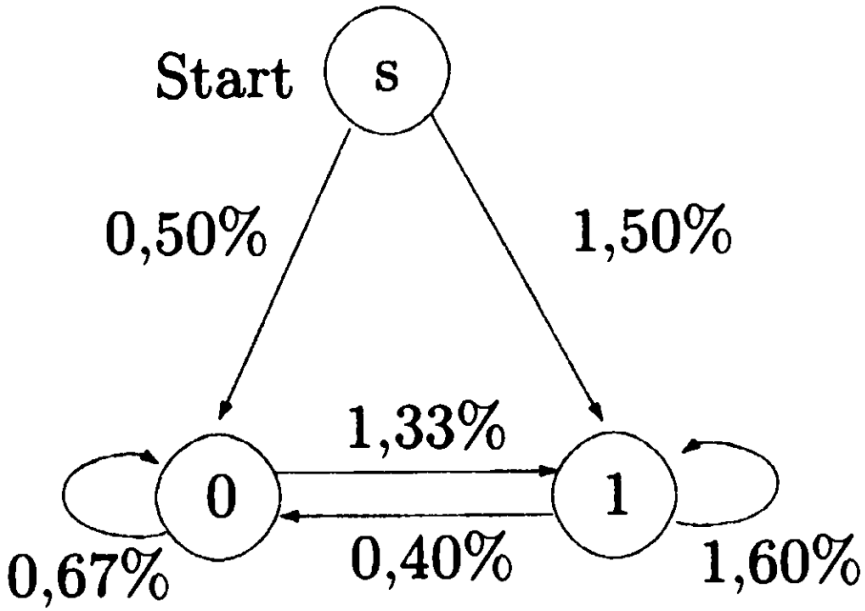
Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

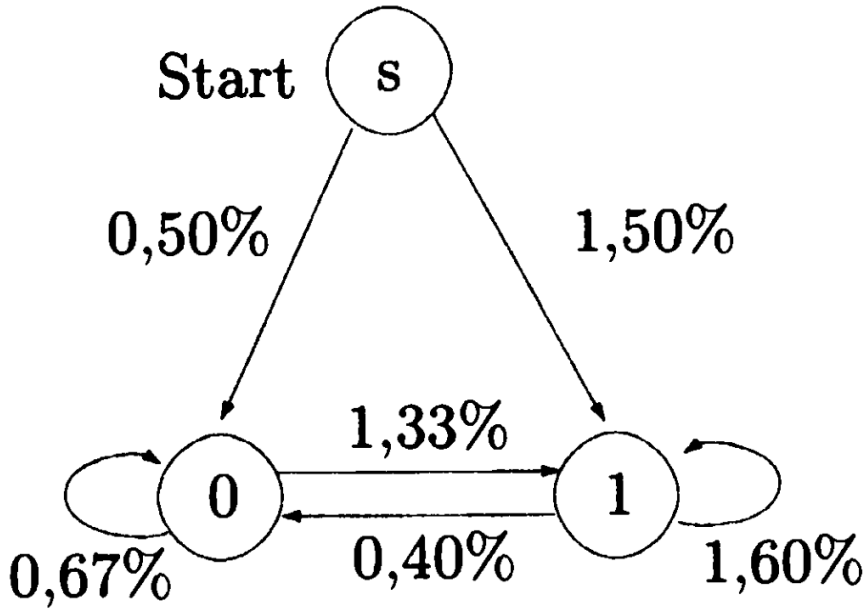
Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.

Зауважимо, що метод Хаффмана не працює у разі двосимвольного алфавіту. У такому алфавіті одному символу доведеться присвоїти код 0, а іншому — 1. Метод Гаффмана не може присвоїти одному символу код коротше одного біта. Якщо вихідні дані (джерело) складаються з індивідуальних бітів, як у випадку двохрівневого (монохромного) зображення, то можливе уявлення кількох біт (4 або 8) у вигляді одного символу нового недвійкового алфавіту (з 16 або 256 символів). Проблема при такому підході полягає в тому, що вихідні бітові дані могли мати певні статистичні кореляційні властивості, які були втрачені при об'єднанні бітів у символи. При скануванні монохромного малюнка або діаграми рядками пікселі частіше зустрічатимуться довгими послідовностями одного кольору, а не швидким чергуванням чорних і білих. У результаті вийде файл, що починається з 1 або 0 (з ймовірністю $1/2$). За нулем з великою ймовірністю слідує нуль, а за одиницею — одиниця.



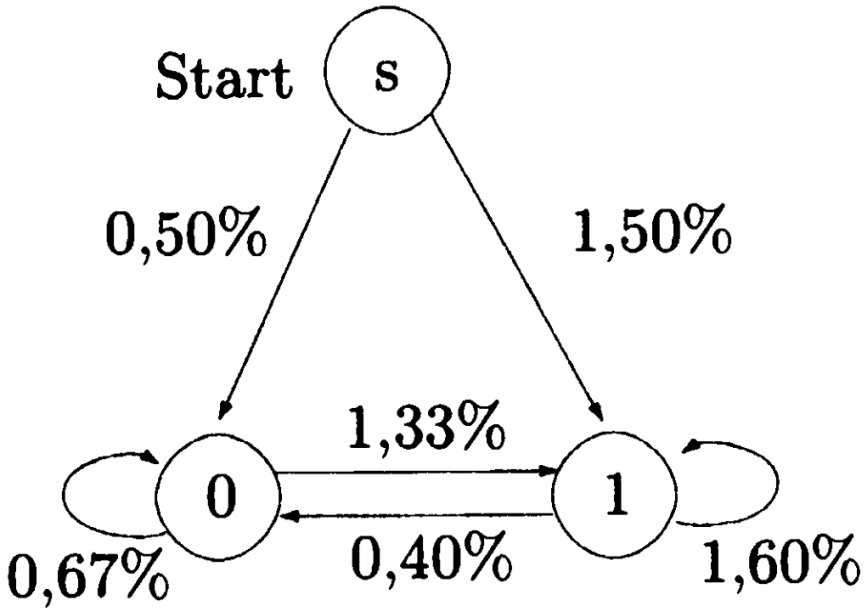
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



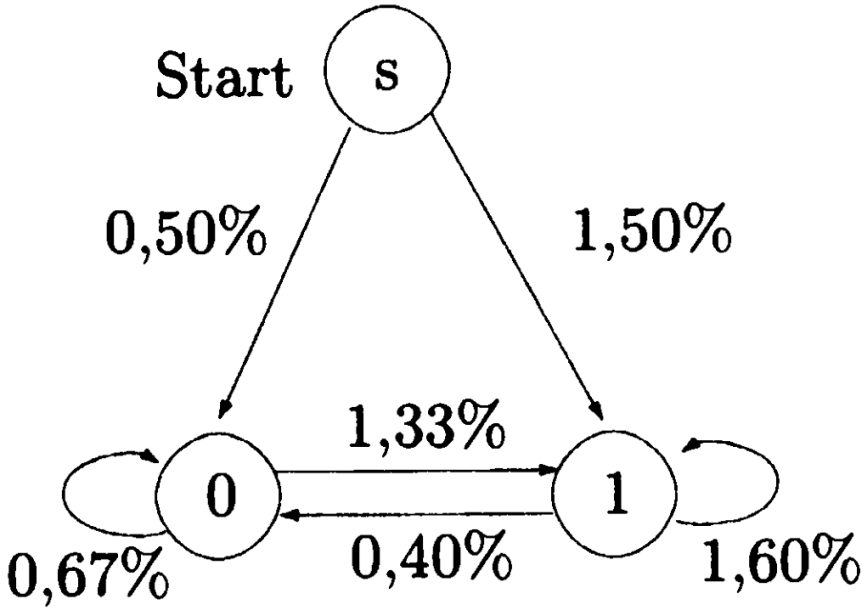
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



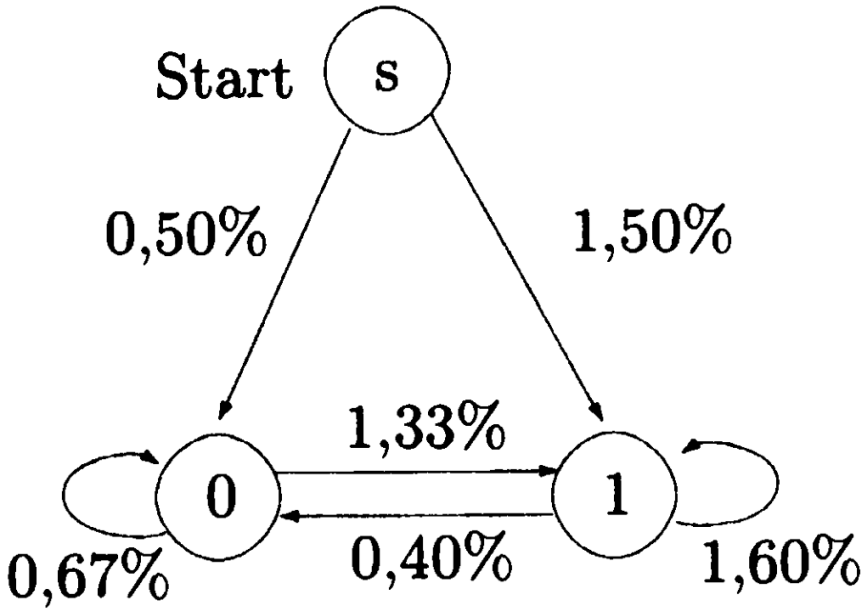
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



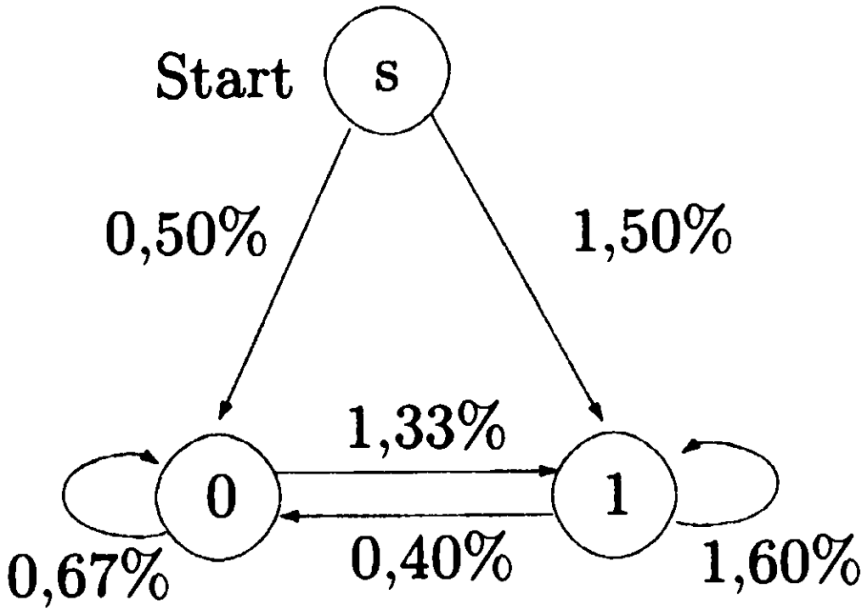
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



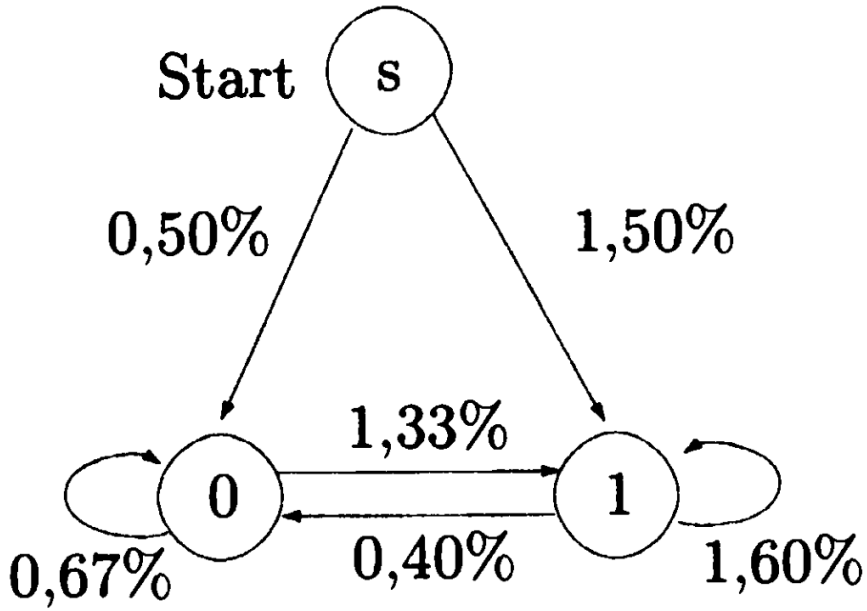
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



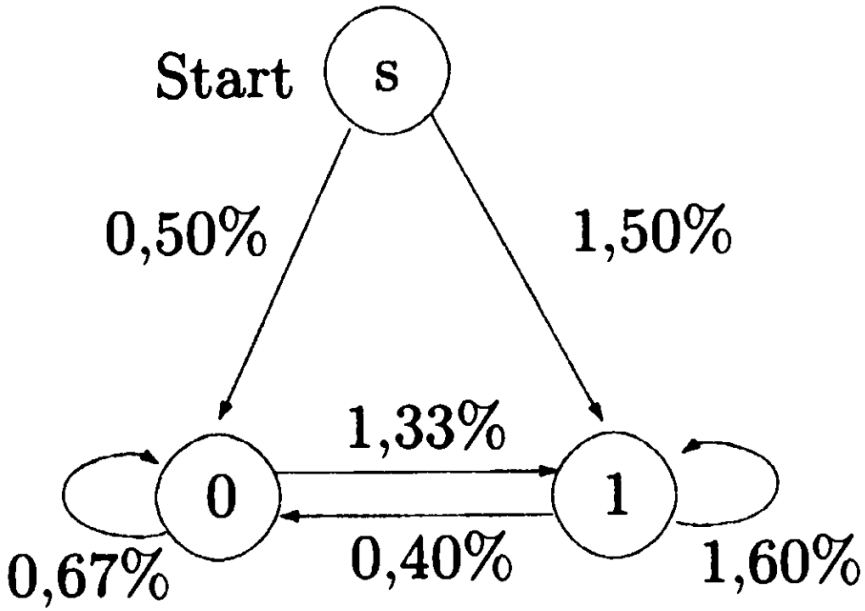
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



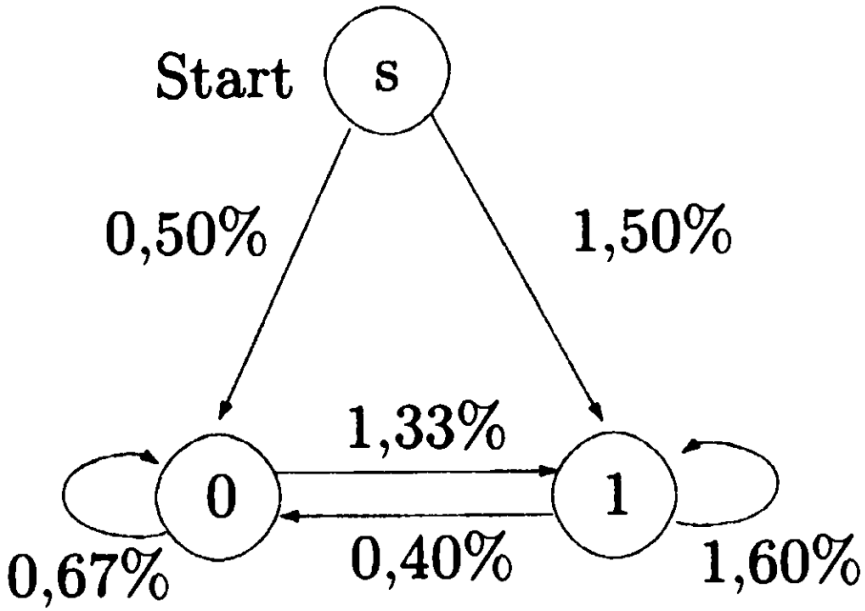
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



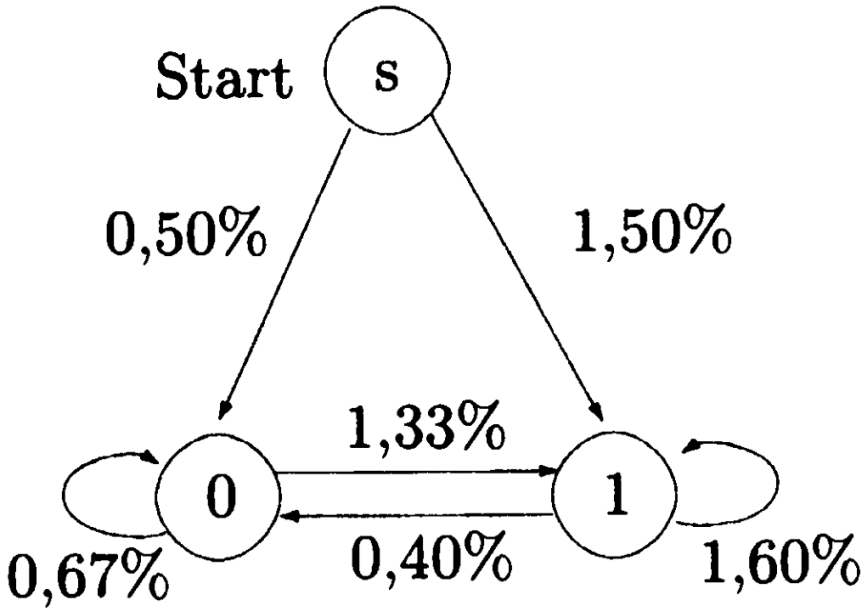
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



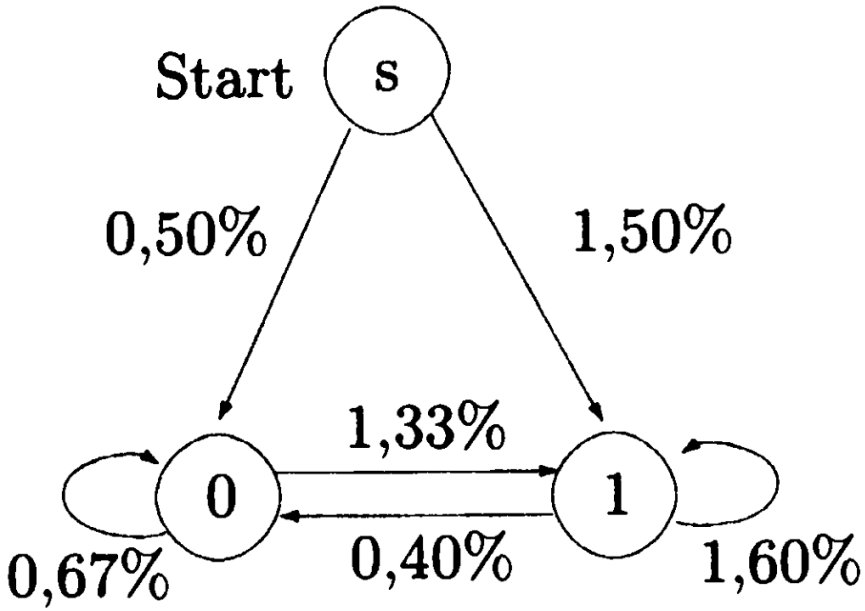
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



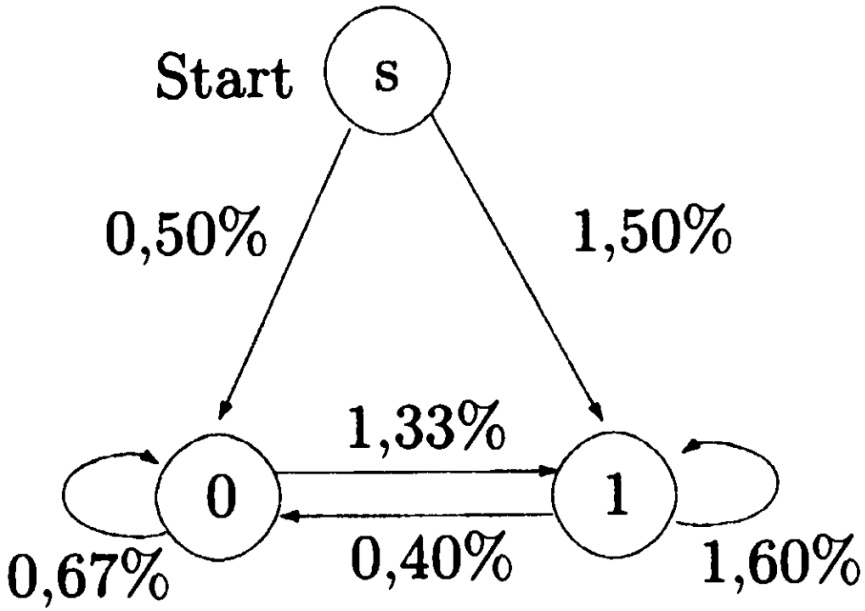
На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.



На рис. зображено скінченний автомат, що ілюструє цю ситуацію. Якщо біти об'єднувати у групи, скажімо, по 8 розрядів, то біти всередині групи будуть корелювати, але самі групи не матимуть кореляцію з вихідними пікселями. Якщо вхідний файл містить, наприклад, дві сусідні групи 00011100 і 00001110, то вони кодуватимуться незалежно, ігноруючи кореляцію останніх нулів першої групи і початкових нулів другої. Вибір довгих груп покращує положення, але збільшує число можливих груп, що тягне за собою збільшення пам'яті зберігання таблиці кодів і подовжує час створення цієї таблиці. [2]

[2] Нагадаємо, якщо група довжини s збільшується до довжини $s+n$, то кількість груп зростає експоненційно з 2^s до 2^{s+n}.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Статистичні методи стиснення зображень. Кодування Гаффмана

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Більш складний підхід до стиснення зображень за допомогою кодів Гаффмана заснований на створенні кількох повних множин кодів Хаффмана. Наприклад, якщо довжина групи дорівнює 8 біт, то породжується декілька сімей кодів розміру 256. Коли необхідно закодувати символ S вибирається одна сім'я кодів, і S кодується з цієї сім'ї.

Приклад

Уявімо зображення з 8-и бітових пікселів, у якому половина пікселів дорівнює 127, а інша половина має значення 128. Проаналізуємо ефективність методу RLE для індивідуальних бітових областей порівняно з кодуванням Гаффмана.

Аналіз. Двійковий запис числа 127 дорівнює 01111111, а числа 128 — 10000000. Половина пікселів поверхні буде нулем, а друга половина — одиницею. У найгіршому випадку область буде схожа на шахівницю, тобто мати багато серій довжини 1. У цьому випадку кожна серія вимагає коду в 1 біт, що веде до одного кодового біта на піксел на область, або 8 кодових бітів на піксел для всього зображення. Це призводить до повної відсутності стиснення. А коду Хаффмана для такого зображення знадобиться всього два коди (оскільки є всього два різних пікселі), і вони можуть бути довжини 1. Це призводить до одного кодового біта на піксел, тобто до восьмиразового стиснення.

Дякую за увагу!