МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт модуля «Статистичні методи стиснення інформації» з курсу «Сучасні методи стиснення інформації»

для студентів спеціальності 186 – Видавництво та поліграфія

Затверджено редакційно-видавничою
радою університету,
протокол № 1 від 30.01.18.

Харків
НТУ «ХПІ»
2018

Методичні вказівки до лабораторних робіт модуля «Статистичні методи стиснення інформації» з курсу «Сучасні методи стиснення інформації» для студентів спеціальності 186 – Видавництво та поліграфія. / Укладачі: В.О. Колбасін, Г.Ю. Сидоренко – Х.: НТУ «ХПІ», 2018. – 24 с.

Укладачі: В.О. Колбасін
Г.Ю. Сидоренко

Рецензент Л.М. Любчик

Кафедра системного аналізу та інформаційно-аналітичних технологій

ВСТУП

Статистичні методи стиснення даних відносяться до найбільш поширених класичних алгоритмів стиснення інформації, що використовуються майже повсюди. У тому числі вони застосовуються у видавництві, як самостійно – для стиснення текстової та бінарної інформації, так і у складі комплексних методів стиснення зображень та інших мультимедійних даних.

Дані методи стиснення основані на усуненні надлишку інформації, яка міститься у вихідних даних, у сенсі інформаційної ентропії Шеннона. Тобто ці методи виконують заміну кодів символів таким чином, щоб символи, які зустрічаються часто, були закодовані кодами меншої довжини. При такому підході до кодування вихідне повідомлення може мати будь-який формат і будь-яке походження та бути відтвореним (декодованим) без спотворень.

Серія лабораторних робіт модуля присвячена вивченню таких статистичних методів стиснення даних, як стиснення за допомогою кодів постійної довжини, метода Хафмана та арифметичного кодування. За час, відведений для виконання лабораторних робіт (2 або 4 академічні години), студент повинен:

1. Відповісти на контрольні запитання для отримання допуску до виконання роботи.
2. Написати програму для стиснення та розтиснення даних з використанням будь-якої мови програмування і середовища розробки (MS VC++, C#, Java тощо).
3. Виконати стиснення та розтиснення даних та проаналізувати характеристики метода стиснення.
4. Оформити звіт до лабораторної роботи.
5. Показати результати викладачу з можливою модифікацією програми у відповідності з додатковими запитаннями.

Лабораторна робота 1
ВИКОРИСТАННЯ КОДІВ ПОСТІЙНОЇ ДОВЖИНИ ДЛЯ СТИСНЕННЯ ДАНИХ

Вступ

Одним з найбільш простих методів стиснення даних є використання кодів постійної довжини. Цей метод передбачає, що бітова довжина кодів всіх символів повідомлення є однаковою, а стиснення відбувається за рахунок обмеження кількості символів у повідомленні. Наприклад, якщо повідомлення складається виключно з цифр – розмір алфавіту для нього буде 10 і для кодування такого повідомлення буде достатньо 4 бітів на символ замість 8 для кодування довільного двоїчного повідомлення.

Метою даної лабораторної роботи є стиснення та розтиснення даних кодом постійної довжини за допомогою власноруч створеного застосування.

1.1. Теоретичні основи

1.1.1. Визначення довжини коду та створення таблиці кодів

Довжина коду повязана з кількістю N унікальних символів у повідомленні, що стискається. Щоб обчислити довжину коду, треба визначити N та знайти мінімально достатню кількість бітів k для представлення N числових кодів за допомогою наступної нерівності:

$$2^{k-1} < N \leq 2^k.$$

Припустимо, що стискається рядок «***мама мила раму***». У цьому повідомленні 7 унікальних символів. Для їх представлення мінімально достатньо 3 біта:

$$2^2 < 7 \leq 2^3.$$

Далі треба призначити кожному унікальному символу повідомлення свій код. Найпростіше це зробити, використовуючи порядковий номер символу у якості коду. Приклад призначення кодів унікальним символам повідомлення наведено у табл. 1.1.

Таблиця 1.1 – Таблиця кодів символів повідомлення «мама мила раму»

№	Символ	Код	№	Символ	Код
1	«_»	000	5	М	100
2	А	001	6	Р	101
3	И	010	7	У	110
4	Л	011			

1.1.2. Стиснення повідомлення

На перший погляд, для виконання стиснення достатньо просто замінити кожен символ повідомлення на його код, але, для того щоб потім можна було розтиснути повідомлення, цього буде недостатньо. Бо розтиснювач має використати таку ж саму таблицю кодів, яка була використана при стисненні повідомлення, та він має знати довжину коду, що використовується. Тому до результату стиснення має бути додано розмір алфавіту повідомлення та список унікальних символів.

Таким чином результат стиснення повинен мати структуру, що наведена на рис. 1.1:

N *(1 байт)* *[7]*	*Список символів* *(7 байт)* *[МА_ИЛРУ]*	*Стиснуте повідомлення* *(6 байт)* *[00001000, 00100010, 10001100, 01010000, 00000011, 00000011]*

Рис. 1.1. Структура стиснутого повідомлення

При розтисненні повідомлення на основі кількості унікальних символів та їх списку декодер відтворює кодову таблицю та за її допомогою відновлює повідомлення. Враховуючи те, що коди символам було призначено на основі їх порядку у списку символів, для декодування достатньо буде зчитати символ зі списку зі зміщенням, яке дорівнює коду символу.

1.1.3. Програмування роботи з бітовими послідовностями

Розглянемо роботу з бітовими послідовностями у С-подібних мовах програмування, до яких відносяться такі популярні мови як C, C++, C#, Java, JavaScript. Ці мови не підтримують роботи з окремими бітами, тому

для роботи з бітовими послідовностями треба використовувати логічні оператори та оператори зсуву над цілими числами.

В рамках лабораторних робіт цього курсу передбачається, що розмір бітового коду буде не більше 20 бітів, тому у якості буферу для бітових послідовностей можна використати 32 бітовий цілий тип ***int***.

Важливо: слід зазначити, що таке припущення може не виконуватись для довільних вхідних даних, тому на практиці часто застосовують більш складні методи роботи з бітовими послідовностями.

Повна інформація про бітовий буфер складається з самого буфера та лічільника кількості бітів у ньому. Це можна об'явити наступним чином:

```
int buffer;         //буфер
int bitsInBuffer;   //лічільник буфера
```

Розглянемо додавання бітової послідовності до бітового буфера.

Щоб додати бітову послідовність до буфера, потрібно записати відповідні біти, починаючи з першого вільного біту у буфері. Для цього потрібно зсунути бітову послідовність вліво на кількість бітів, що дорівнює лічильнику буфера та поєднати її з буфером за допомогою логічної операції «АБО». Також на відповідну довжину слід збільшити лічильник буферу.

Після цього слід вивести у вихідний потік усі заповнені байти, якщо вони є. Щоб виділити заповнений байт, слід виконати логічну операцію «ТА» між буфером та маскою з 8-ма одиничними бітами. Результат операції треба вивести у вихідний потік, буфер зсунути вправо на 8 біт та відняти 8 з лічильнику буфера.

Ці операції виконуються наступним кодом:

```
void addBitsToBuffer(int bits, int bitsCount) {
    //Додаємо бітову послідовність до буферу
    buffer = buffer | (bits << bitsInBuffer);
    bitsInBuffer += bitsCount;
    //Вивід заповнених байтів у вихідний поток
    while(bitsInBuffer > 8) {
        output(buffer & 0xFF);
        buffer = buffer >> 8;
        bitsInBuffer = bitsInBuffer - 8;
    }
}
```

На рис. 1.2 наведено, що відбувається з бітовим буфером при додаванні бітового коду другої літери «м» при кодуванні рядка прикладу.

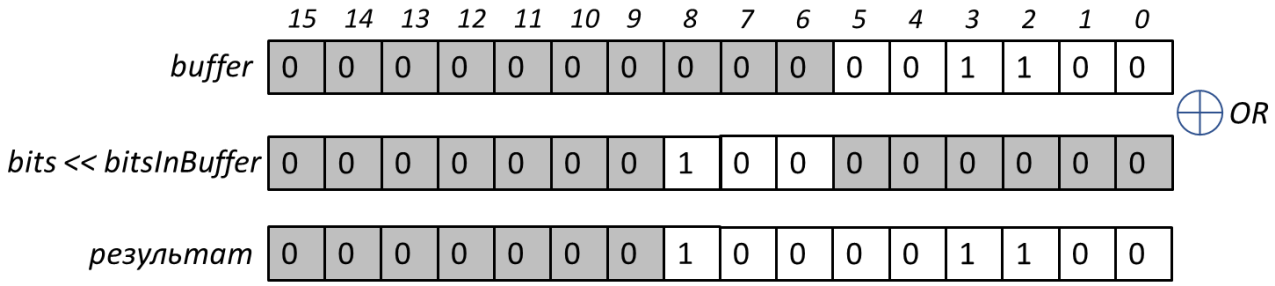

Рис. 1.2. Додавання коду літери «м» до буфера (бінарний код літери – «100»; лічильник буфера дорівнює 6; сірим кольором показані неінформативні біти)

Після додавання коду лічильник буфера буде дорівнювати 9, а сам буфер буде зберігати один повний байт. Тому далі виконується вивід заповнених байтів у вихідний потік, як показано на рис. 1.3.

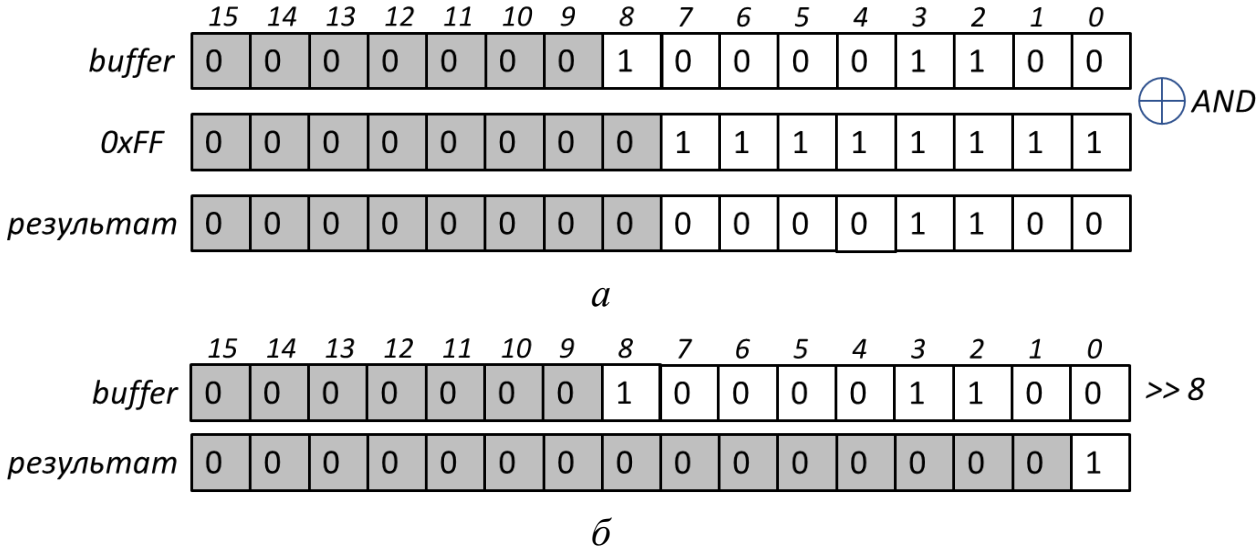

Рис. 1.3. Виділення байту для виводу *(а)* та вилучення його з буферу *(б)*; (лічильник буфера дорівнює 9; сірим кольором показані неінформативні біти)

Далі розглянемо отримання бітової послідовності з буфера.

Для отримання бітової послідовності слід переконатися, що буфер достатньо заповнено та при необхідності заповнити його. Після цього слід маскою виділити необхідну кількість бітів з буфера та зсувом вліво видалити їх з буфера. Також треба відняти відповідну кількість бітів від лічильника буфера.

Ці операції виконуються наступним кодом:

```
int readBitsFromBuffer(int bitsCount) {
    //Зчитуємо необхідну кількість байтів в буфер
    while(bitsInBuffer < bitsCount) {
        buffer = buffer | (input() << bitsInBuffer);
        bitsInBuffer = bitsInBuffer + 8;
    }
    //Отримуємо бітову послідовність
    int mask = (1 << bitsCount) - 1;
    int result = buffer & mask;
    //Видаляємо з буферу зчитані біти
    buffer = buffer >> bitsCount;
    bitsInBuffer = bitsInBuffer – bitsCount;
    return result;
}
```

Маска формується таким чином. Зсувом вліво формується бітова послідовність з однієї одиниці та ***bitsCount*** нулів. Після віднімання одиниці ця бітова послідовність перетворюється на ***bitsCount*** одиниць, що і треба від маски для виділення ***bitsCount*** бітів. Детально цей процес наведено на рис. 1.4.

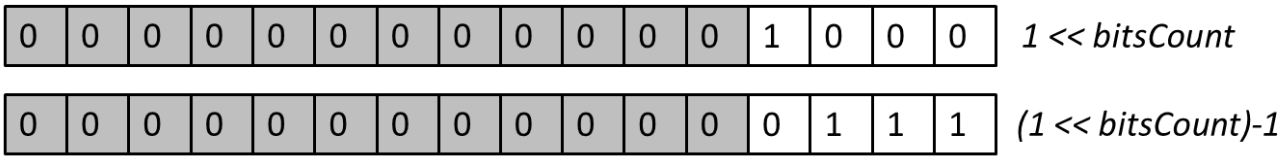

Рис. 1.4. Обчислення маски для виділення 3 наймолодших бітів

1.2. Завдання на лабораторну роботу

За час виконання лабораторної роботи (2 академічні години) студент повинен розробити програму для стиснення та розтиснення текстового файлу за допомогою кодів постійної довжини та з використанням цієї програми виконати стиснення та розтиснення текстового файлу.

Вихідні дані для виконання завдання требя взяти у відповідності до табл. А.1.

Контрольні запитання

1. Якою буде довжина коду постійної довжини, якщо у повідомленні є 8 унікальних символів?
2. Якою буде довжина коду постійної довжини, якщо у повідомленні є 31 унікальний символ?
3. Якою буде ступінь стиснення, якщо у повідомленні є 50 унікальних символів та довжина повідомлення – 200 літер? (При обчисленні слід враховувати службові дані)
4. Якою буде ступінь стиснення, якщо у повідомленні є 50 унікальних символів та довжина повідомлення – 10 літер? (При обчисленні слід враховувати службові дані)
5. Які є засоби для роботи з бітовими послідовностями у C-подібних мовах?
6. Як додати бітову послідовність до буфера?
7. Як витягнути бітову послідовність з буфера?
8. Як сформувати маску для отримання N молодших бітів?
9. Коли можна використовувати для зберігання бітового буферу байт?
10. Коли розмір повідомлення після стиснення буде більшим ніж у вихідного повідомлення?

Лабораторна робота 2
МЕТОД ХАФФМАНА

Вступ

Метод Хаффмана є класичним методом стиснення даних кодами змінної довжини, які мають найменшу середню довжину. Метод виконує ідеальне стиснення (тобто стиснення до рівня, що задається формулою інформаційної ентропії Шеннона), якщо частоти символів дорівнюють ступіням 2. Цей метод широко використовується для стиснення даних як самостійно, так у складі комплексних методів стиснення даних.

Метою даної лабораторної роботи є стиснення та розтиснення даних кодом Хаффмана за допомогою власноруч створеного застосування.

2.1. Теоретичні основи

2.1.1. Створення таблиці кодів

Алгоритм створення кодів Хаффмана є ітеративним. Після його виконання буде створено дерево кодів Хаффмана, за яким будуть побудовані коди Хаффмана символів.

Алгоритм починається з того, що з символів повідомлення формується перелік вільних вузлів дерева. Спочатку до нього додаються листи дерева, що містять символи повідомлення та їх частоту.

Далі з переліку обираються два вузла з найменшою частотою та поєднуються у новий вузол дерева, частота якого буде сумою частот його елементів. Вузли, що були поєднані у новий, видаляються з переліку вільних, а новий вузол до нього додається. Ця операція проводиться доки у переліку не залишиться один вузол. Процес побудови дерева кодів Хафмана для тестового повідомлення «мама мила раму» наведений на рис. 2.1.

Слід зазначити, що на кожному кроку алгоритму можна обирати будь-які елементи переліку, але вони мають бути з найменшою частотою. В іншому разі код буде сформований невірним чином, що приведе до зниження ступіня стиснення.

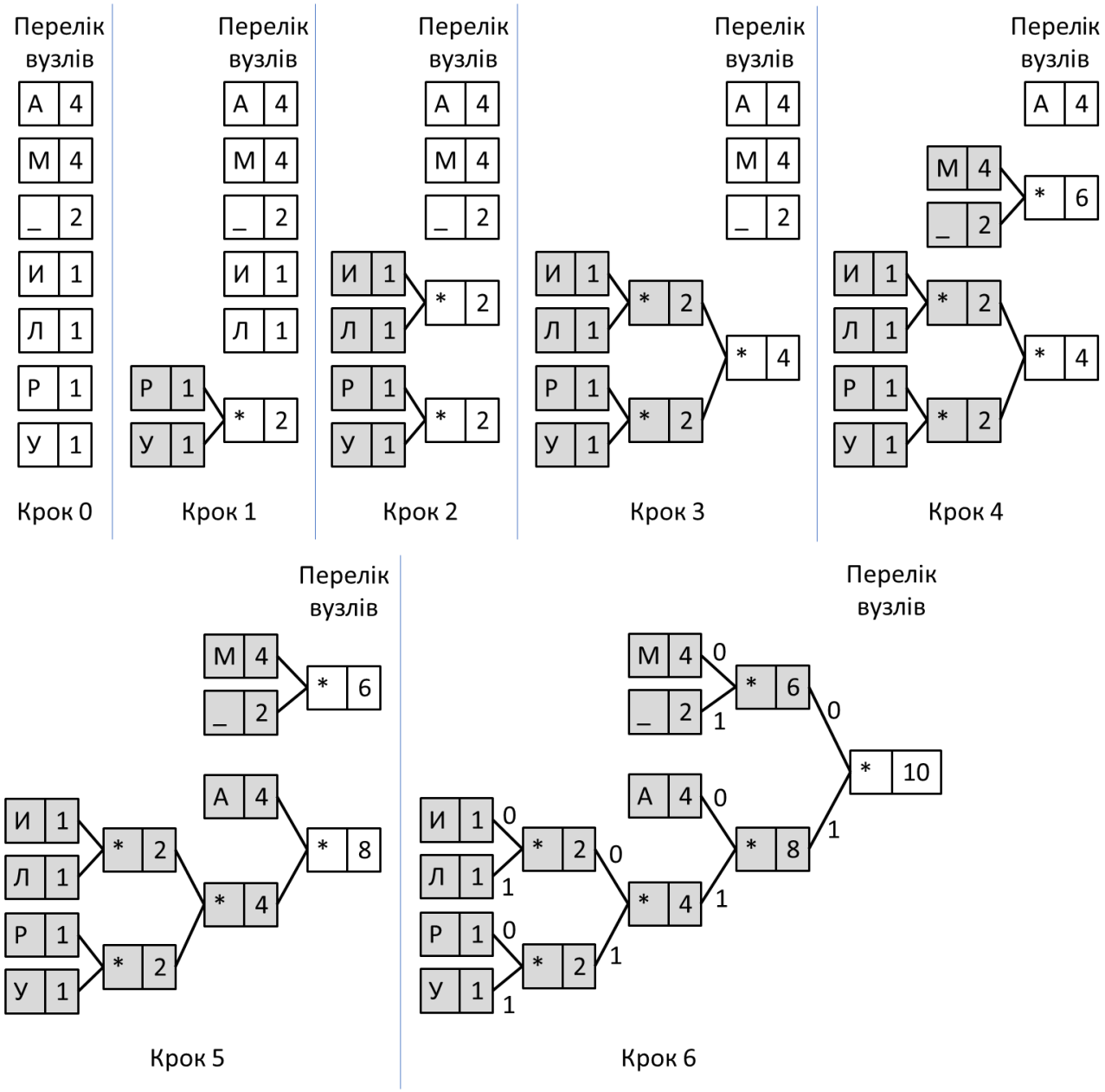

Рис. 2.1. Побудова дерева кодів Хаффмана

Після того, як дерево кодів буде побудовано, гілкам дерева призначають біти «0» або «1» за обраним правилом. На наведеному на рис. 2.1 прикладі верхній гілці призначається «0», а нижній – «1».

Правила поєднання вільних вузлів дерева та призначення бітів гілкам дерева мають бути однаковими для кодера та декодера.

Коди символів будуються обходом дерева зверху-донизу. Кодом Хаффмана для символа буде конкатенація бітів, які призначені гілкам дерева, що були пройдені при спуску з корня дерева до листа з

відповідним символом. Таблиця кодів Хаффмана для рядка прикладу наведена у таблиці 2.1.

Таблиця 1.1 – Таблиця кодів символів повідомлення «мама мила раму»

№	Символ	Код	№	Символ	Код
1	« »	01	5	М	00
2	А	10	6	Р	1110
3	И	1100	7	У	1111
4	Л	1101			

Коди Хаффмана є префіксними кодами. Це означає, що немає такого коду, який би був префіксом іншого коду. Ця властивість кодів дозволяє однозначно декодувати кодоване повідомлення. Також її можна використовувати для перевірки побудованих кодів.

2.1.2. Кодування та декодування

Для успішного розтиснення даних декодер повинен використовувати таке саме дерево кодів, що і кодер. Найбільш просто це можна забезпечити за рахунок додавання таблиці частот символів до стиснутого повідомлення. У цьому разі декодер почне роботу з побудови дерева кодів Хаффмана на основі зчитаної зі стиснутого повідомлення таблиці частот. Для виконання лабораторної роботи можна зберігати частоти символа як 16-бітове ціле число.

Процес декодування символу, закодованого методом Хаффмана, виглядає як спуск по дереву до відповідного листа. У цьому процесі кожен зчитаний біт визначає: по якій гілці треба переходити до наступного вузла (0 – по верхній, 1 – по нижній у прикладі). Після того, як буде досягнуто листа дерева, символ у цьому листі виводиться у вихідний потік. Далі виконується декодування наступного символу.

Може статися, що останній байт повідомлення буде частково заповнений кодами символів. Щоб уникнути помилкового декодування зайвого символу, слід передавати до декодера кількість символів у вихідному повідомленні. Тоді після декодування переданої кількісті символів декодер зупиниться незважаючи на те, що в буфері є необроблені біти.

Таким чином, результат стиснення повинен мати структуру, що наведена на рис. 2.2:

Символів таблиці (1 байт) [7]	*Таблиця частот (21 байт) [(М,4), (А,4), (_,2), (И,1),(Л,1),(Р,1),(У,1)]*	*Символів повідомлення (2 байта) [14]*	*Стиснуте повідомлення (5 байт) [01000100, 00110010, 10011011, 00010111, 00001111]*

Рис. 2.2. Структура стиснутого повідомлення

Процес кодування виглядає простішим ніж процес декодування. Кодер має записати дані про таблицю частот символів, кількість символів повідомлення та для кожного символу повідомлення записати його код.

2.1.3. Робота з бітовими послідовностями

Для роботи з бітовими послідовностями при кодуванні та декодування кодів Хаффмана можна використовувати методи, які було створено у лабораторній роботі 1, ***addBitsToBuffer*** та ***readBitsFromBuffer***.

При цьому треба приділити увагу формуванню бітових послідовностей кодів Хаффмана у кодері. Код слід записувати у порядку з молодших до старших бітів. Такий порядок потрібен, бо декодер не знає, який символ він декодує, тому після останнього біта символа k він має зчитати перший біт символа $k + 1$.

Для формування кодів також можна використовувати метод ***addBitsToBuffer***. Наприклад, код Хаффмана для символа «Р» можна отримати, викликавши цей метод на кожній гілці дерева:

```
addBitsToBuffer(1,1);
addBitsToBuffer(1,1);
addBitsToBuffer(1,1);
addBitsToBuffer(0,1);
```

Після цього значення буфера можно записати до таблиці та використовувати при кодування повідомлення.

Слід пам’ятати, що, якщо бітовий буфер зберігає хоча б один біт коду після закінчення кодування, його молодший байт має бути збережений у вихідній потік.

2.2. Завдання на лабораторну роботу

За час виконання лабораторної роботи (4 академічні години) студент повинен розробити програму для стиснення та розтиснення текстового файлу за допомогою кодів Хаффмана та з використанням цієї програми виконати стиснення та розтиснення текстового файлу.

Вихідні дані для виконання завдання необхідно взяти у відповідності до табл. А.1.

Контрольні запитання

1. Яка довжина буде у кодів Хаффмана для повідомлення з 8 унікальних символів, якщо всі символи мають одну частоту – 10?
2. Яке розподілення частот символів призведе до формування найбільш довгого коду Хаффмана?
3. Якою може бути максимально можлива довжина коду Хаффмана на алфавіті з 256 символів? При якому розподілі частот символів її можна досягнути?
4. Що таке «префіксність» кодів Хаффмана та навіщо ця властивість потрібна?
5. Чому при створенні дерева Хаффмана потрібно поєднувати вузли з мінімальною частотою?
6. Чому у вихідній потік треба записувати розмір вихідної послідовності?
7. Як можна прискорити кодування символів уникаючи виконання спуску по дереву кодів?
8. Навіщо кодеру та декодеру мати однакові дерева кодів Хаффмана та що буде, якщо цю вимогу буде порушено?
9. Який найбільш можливий ступінь стиснення при використанні кодів Хаффмана?
10. Який найменш можливий ступінь стиснення при використанні кодів Хаффмана?

Лабораторна робота 3
АРИФМЕТИЧНЕ КОДУВАННЯ

Вступ

Метод арифметичного кодування є ще одним підходом до статистичного стиснення інформації. Він дозволяє обійти обмеження в один біт на символ кодованого повідомлення, що діє для метода Хаффмана, та дозволяє суттєво підвищити ступінь стиснення однородних даних у випадку, якщо частота символу не є ступінню двійки (тобто у більшості практично важливих випадків). Арифметичне кодування також широко використовується як самостійний метод стиснення даних, так і у складі комплексних методів стиснення.

Метою даної лабораторної роботи є стиснення та розтиснення даних методом арифметичного кодування за допомогою власноруч створеного застосування.

3.1. Теоретичні основи

3.1.1. Головна ідея арифметичного кодування

Метод арифметичного стиснення представляє стиснуте повідомлення у вигляді дуже довгого дрібу.

Щоб зрозуміти головну ідею метода, візьмемо інтервал [0,1) та розіб'ємо його на інтервали пропорційні до частоти появи символів. Для однозначності розташуємо ці інтервали в порядку зменшення частоти символів. Далі для кожного вихідного символу будемо обирати відповідний символу інтервал (робочій інтервал) та розбивати його знов на підінтервали. Після того, як останній символ повідомлення буде оброблено, будь-яке число у відповідному робочому інтервалі може бути використане для кодування всього повідомлення.

Схема розбиття інтервалів та кодування перших трьох символів «abb» повідомлення, з частотами символів: «a» – 5, «b» – 3, «c» – 2, наведена на рис. 3.1. Будь-яке число всередені робочого інтервалу [0.325, 0.37) може бути використане як результат кодування вихідної послідовності. Наприклад, як результат кодування цих трьох символів, може бути використане число 0.36.

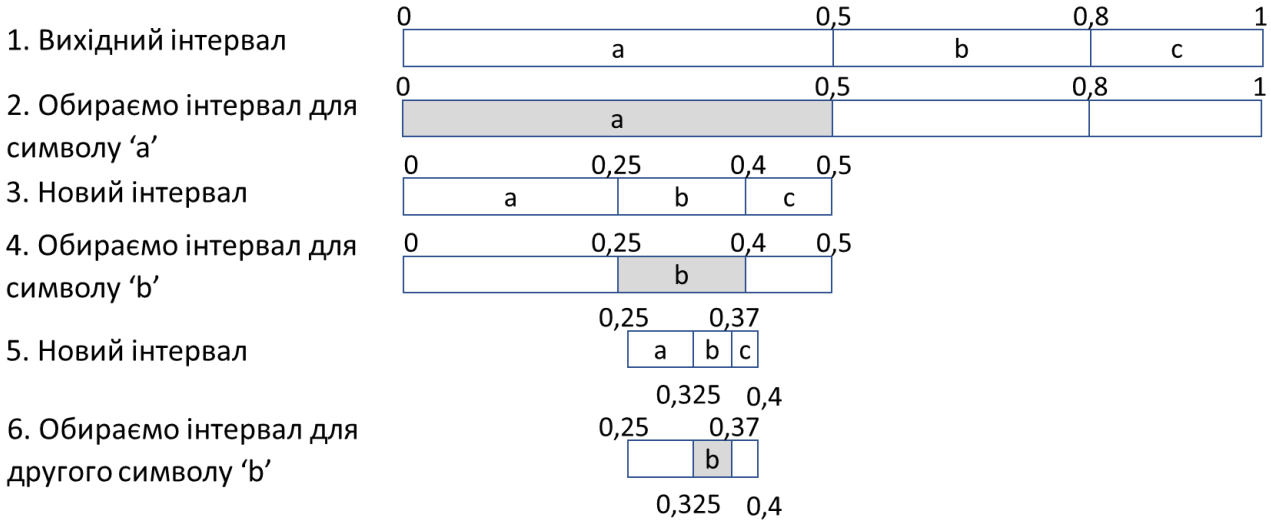

Рис. 3.1. Кодування символів «abb»

Декодування виконується подібно до кодування. За вихідним інтервалом визначається, якому символу належить дріб, що є результатом кодування. Далі символ виводиться до потоку виводу, а відповідний підінтервал обирається робочим та розбивається на інтервали. Ці кроки виконуються доки не будуть відновлені всі символи повідомлення.

Цю ідею в такому виді майже неможливо реалізувати на практиці, тому що дріб буде надмірно довгим та буде величезна втрата точності при виконанні операцій з дійсними числами, що представлені типами з рухомою крапкою. Тому на практиці використовують іншу реалізацію алгоритму на базі цілих чисел з постійною крапкою.

3.1.2. Арифметичне кодування з постійною комою

Цей алгоритм використовує постійну крапку, що розташована перед старшим бітом цілого беззнакового типу. При такому розташуванні діапазон значень цього типу, як раз представляє інтервал [0,1).

Щоб уникнути втрати точності, після кожної зміни робочого інтервалу виконується нормалізація. Головна мета нормалізації – забезпечити, щоб між межами робочого інтервалу було більше ніж половина діапазону значень цілого типу. Розглянемо, як виконується нормалізація.

Якщо робочій інтевал цілком знаходиться у нижній або верхній половині діапазону значень цілого типу, то межі інтервалу мають однакові старші біти. Це означає, що можна записати значення цього біту у

вихідний потік, а залишок помножити на два, збільшуючи робочій інтервал у два рази.

Якщо межі робочого інтервалу наближаються до середини діапазону цілого типу і знаходяться у його 2-ій та 3-ій чвертях, тоді віднімається чверть діапазону від меж інтервалу. У цьому випадку не можливо нічого відразу записати у вихідний потік, тому ця чверть інтервалу «позичається» і відповідні біти будуть виведені пізніше.

Кодування виконується наступним чином:

```
int l = 0, h = MAX_INT, half = MAX_INT/2+1;
int firstQtr = half/2, thirdQtr = half + firstQtr;
int intervalPoint = MAX_INT/2+1;
int bitsBorrow = 0;
while(source.hasNext()){
  char c = source.nextChar();
  int j = getIndexForSymbol(c);
  //отримуємо новий робочий інтервал
  int lNew = l + range[j  ]*(h-l+1)/intervalPoint;
  int hNew = l + range[j+1]*(h-l+1)/intervalPoint - 1;
  //виконуємо нормалізацію
  while(lNew > firstQtr || hNew < thirdQtr) {
    if (hNew < half) {
      writeBit(0, bitsBorrow);
    } else if (lNew > half) {
      writeBit(1, bitsBorrow);
      lNew = lNew - half; hNew = hNew - half;
    } else if (lNew > firstQtr && hNew < thirdQtr) {
      bitsBorrow++;
      lNew = lNew - firstQtr; hNew = hNew - firstQtr;
    }
    lNew = 2*lNew; hNew = 2*hNew + 1;
  }
  l = lNew; h = hNew;
}

//метод виводу біту з урахуванням позичених бітів
void writeBit(int bit, int borrowCnt){
  addBitsToBuffer(bit, 1);
  while(borrowCnt-- > 0){ addBitsToBuffer(~bit, 1); }
}
```

Метод ***getIndexForSymbol*** повертає індекс нижньої межі інтервалу відповідного символу. Оскільки інтервали розташовані один за одним –

якщо від нижній межі наступного інтервалу відняти одиницю, то отримаємо верхню межу поточного інтервалу.

Межі інтервалу також зберігаються в форматі з постійною крапкою. Позиція крапки позначається константою ***intervalPoint***. Це ж саме значення слід використовувати при побудові масиву меж інтервалів ***range***. Також слід переконатися, що обчисленні межі інтервалів не мають жодного з нульовою довжиною. Якщо такий інтервал з'являється, слід збільшити частоту символу, це призведе до зниження ступеня стиснення, але забезпечить коректність декодування повідомлення.

Важливо: у залежності від мови програмування, що використовується, можлива втрата точності при обчисленні нового робочого інтервалу. Переконайтеся, що відповідні арифметичні операції виконуються з використанням подвійної точності (тип ***long***) та за необхідністю змініть відповідний код.

Після закінчення обробки усіх символів повідомлення треба записати у вихідний потік біти, що дозволять однозначно визначити координати нижньої межі робочого інтервалу. Для цього треба визвати метод ***writeBit(l<firstQtr?0:1, 1)***.

Декодування виконується наступним чином:

```
int l = 0, h = MAX_INT, half = MAX_INT/2+1;
int firstQtr = half/2, thirdQtr = half + firstQtr;
int intervalPoint = MAX_INT/2+1;
int value = readIntBits();
int count = 0;
while(count < dataSize){
  int pos = (value - l)*intervalPoint - 1)/(h - l + 1);
  int j = getIndexForPosition(pos);
  output(symbol[j]);
  //отримуємо новий робочій інтервал
  int lNew = l + range[j  ]*(h-l+1)/intervalPoint;
  int hNew = l + range[j+1]*(h-l+1)/intervalPoint - 1;
  //виконуємо нормалізацію
  while(lNew > firstQtr || hNew < thirdQtr) {
    if (lNew > half) {
      lNew = lNew - half; hNew = hNew - half;
    } else if (lNew > firstQtr && hNew < thirdQtr) {
      lNew = lNew - firstQtr; hNew = hNew - firstQtr;
    }
    lNew = 2*lNew; hNew = 2*hNew + 1;
    value = (value<<1) | readBitsFromBuffer(1);
  }
```

```
  l = lNew; h = hNew; count++;
}
```

Функція ***readIntBits*** зчитує біти з кодованого повідомлення так, щоб заповнити ними змінну цілого типу ***int***. Слід зазначити, що заповнення змінної слід виконувати у напрямку зі старшого біту до молодшого. При використанні метода ***readBitsFromBuffer*** – це потребує написання циклу, та використання маски з одиничним бітом, що зсувається з позиції старшого біту до молодшого.

Функція ***getIndexForPosition*** виконує пошук індексу в масиві меж інтервалів символів для заданого числа. Функція повертає індекс найбільшого елементу масиву, який є меншим за параметр. Відповідно до мови програмування, що використовується, при її реалізації можна застосувати бібліотечні функції.

Порівнюючи реалізацію кодування та декодування легко помітити, що кодер та декодер використовують однаковий код для обчислення нового робочого інтервалу та виконання нормалізації. Це є вкрай необхідною деталлю реалізацій, бо інакше накопичення помилок округлення може призвести до розбіжностей у межах робочого інтервалу між кодером та декодером, що призведе до помилкового розтиснення повідомлення.

3.1.3. Кодування та декодування

Для успішного розтиснення даних декодер має використовувати ті самі межі інтервалів символів, що і кодер. Найбільш простим способом реалізувати це є передача до декодера таблиці частот символів з стисненим повідомленням. За нею декодер відтворить масив меж інтервалів символів, а потім виконає декодування за наведеним вище алгоритмом.

Реалізація декодера, що розглядається, не використовує спеціальних символів для позначення того, що повідомлення вже декодовано. Тому, для уникнення помилкового декодування зайвого символу слід передавати до декодера кількість символів у вихідному повідомленні.

Зважаючи на усі перелічені передумови, результат стиснення повинен мати структуру, що наведена на рис. 3.2:

Символів таблиці (1 байт) [7]	*Таблиця частот (21 байт) [(М,4), (А,4), (_,2), (И,1),(Л,1),(Р,1),(У,1)]*	*Символів повідомлення (2 байта) [14]*	*Стиснуте повідомлення (5 байт) [01010000, 00110101, 01101110, 00101000, 00110000]*

Рис. 3.2. Структура стиснутого повідомлення

3.2. Завдання на лабораторну роботу

За час виконання лабораторної роботи (2 академічні години) студент повинен розробити програму для стиснення та розтиснення текстового файлу за допомогою арифметичного кодування та з її використанням виконати стиснення та розтиснення текстового файлу.

Вихідні дані для виконання завдання требя взяти у відповідності до табл. А.1.

Контрольні запитання

1. Який максимальний ступінь стиснення може бути досягнутий при використанні метода арифметичного кодування?
2. Чому неможливо використовувати дійсні числа з рухомою крапкою для реалізації метода арифметичного кодування?
3. Яке число з результуючого робочого інтервалу може бути використано як результат стиснення?
4. Яке найменше ненульове число може бути представлено числом з нерухомою комою, якщо позиція коми – 4-й біт?
5. Навіщо потрібна операція нормалізації?
6. Чому, якщо обидві межі робочого інтервалу є меншими ніж половина діапазону, у вихідний потік записується 0?
7. Що станеться, якщо у декодері використати цілий тип меншої розмірності? Наприклад, кодер використовує 32-розрядне ціле, а декодер – 16-ти.
8. Що станеться, якщо кодер не передав декодеру кількість символів у повідомлені?
9. Чому арифметичне кодування може забезпечити більший ступінь стиснення ніж метод Хаффмана?
10. Що станеться, якщо у вихідному повідомленні частота символів однакова?

СПИСОК ЛІТЕРАТУРИ

1. Сэломон Д. Сжатие данных, изображений и звука. М.:Техносфера, 2004. 368с.
2. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. М.:ДИАЛОГ-МИФИ, 2003. 384 с.
3. Mahoney M. Data Compression Explained. Dell Inc., 2013. Режим доступу: http://mattmahoney.net/dc/dce.html. Дата звертання: 17.11.2017.

ДОДАТОК А. Завдання для лабораторних робіт

Таблиця А.1 – Варіанти даних для завдання до лабораторних робіт

1. Прилетіли горобці – говорили про крупці; не про крупці, не про крупицю, а про крупячко.
2. Шило шубку Шурі шило, шовком, шерстю, шви обшило. Вийшла шубка прехороша нашій Шурі на порошу.
3. Перепілка – гарна птиця, та хлоп’ят вона боїться, бо хлоп’ята беруть гілку і лякають перепілку. Не потрібно так лякать, в неї п’ять перепелят.
4. Сторож на баштані кріт якось зготував обід. Лиш зачувши про обід, до крота примчався кіт. Кіт і кріт посмакували, кіт і кріт порозмовляли, трішки кіт і кріт поспали. Лихо! Кавуни покрали!
5. Котилася торба з високого горба. В торбі паляниця, ведмідь і лисиця. Лисиця пищить, ведмідь верещить, а торба тріщіть: трісь, трісь, трісь!
6. Качка чотири яєчка знесла, в червні вона каченят навела. Ходить тепер вона з ними на річку їсть по дорозі смачненьку суничку.
7. День народження у джунглях відзначає Джонатан. Одягнув нарядні джинси відбиває джаз у джбан.
8. Кукурудза на городі дивувалась своїй вроді: милувалась дуже-дуже наче в дзеркало в калюжу. Білі зуби, коса довгенька, та сукня гарна, зелененька.
9. Сім пухнастих кошеняток вийшли в поле пострибати, троє, причаївшись нишком, хочуть упіймати мишку. А найменше кошенятко по доріжці біжить з татком.
10. Кілька яблучок червоних принесла в гніздо ворона. Два дала вороненяті, два великих – його тату, а чотири – заховала.

ЗМІСТ

Навчальне видання

Методичні вказівки
до лабораторних робіт модуля
«Статистичні методи стиснення інформації»
з курсу «Сучасні методи стиснення інформації»
для студентів спеціальності 186 – Видавництво та поліграфія

Укладачі: КОЛБАСІН Вячеслав Олександрович
СИДОРЕНКО Ганна Юріївна

Відповідальний за випуск О.С. Куценко
Роботу до видання рекомендував проф. О.В. Горілий

В авторській редакції

План 2018 р., поз. 93

Підписано до друку 11.05.2018 р. Формат 60×84 1/16. Папір офсетний.
Друк – ризографія. Гарнітура Таймс. Ум. друк. арк. 1,2.
Наклад 50 прим. Зам. № 283-20. Ціна договірна

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2

Друкарня «ФОП Пісня О.В.»
Свідоцтво про державну реєстрацію ВО» № 249750 від 13.09.2007 р.
61002, Харків, вул. Гіршмана, 16а, кв. 21, тел. (057) 764-20-28