**Черкаський національний університет імені Богдана Хмельницького**

**Кафедра Інформаційних технологій**

**КУРСОВА РОБОТА**

***з дисципліни***

***«Програмування та алгоритмічні мови»***

***НА ТЕМУ «Реалізація архіватора файлів на основі кодів Гаффмана»***

Студентки 2 курсу, групи\_КН-21\_

напряму підготовки «Інформаційні технології»

спеціальності «Комп’ютерні науки»

\_\_\_\_\_\_\_\_\_Погорілої К. С.\_\_\_\_\_\_\_

**(прізвище та ініціали)**

Керівник викладач Воєводін Є. В.

**(посада, вчене звання, прізвище та ініціали)**

Національна шкала: \_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_ Оцінка: ECIS\_\_

Члени комісії:

\_\_\_\_\_\_\_ Гребенович Ю.Є.\_

**(підпис) (прізвище та ініціали)**

\_\_\_\_\_\_\_ Любченко К. М.\_

**(підпис) (прізвище та ініціали)**

\_\_\_\_\_\_\_ Царик Т. Ю.\_\_\_\_

**(підпис) (прізвище та ініціали)**

**м. Черкаси – 2022 рік**

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ ЧЕРКАСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ім. Б. ХМЕЛЬНИЦЬКОГО**

**Кафедра інформаційних технологій**

ЗАТВЕРДЖУЮ

Завідувач кафедри,

Веретельник В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

«\_\_\_» \_\_\_\_\_\_\_\_ 2022 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

**на курсову роботу**

студентки \_\_\_\_Погорілої Катерини Сергіївни\_\_\_\_\_\_ групи\_\_КН-21\_ другого курсу\_\_\_\_\_\_

**ТЕМА** \_\_\_\_\_Реалізація архіватора файлів на основі кода Гаффмана\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Вихідні дані до курсової роботи:** Розроблений програмний продукт повинен:\_\_\_\_\_\_\_\_\_\_

1. Реалізувати алгоритм стиснення даних в основі якого лежить використання кодів Гаффмана. Алгоритм кодування складається з двох етапів. Перший, формується дерево, де кожному символу з файлу (що архівується) відповідає набір нулів та одиниць. Другий, заміна символів в файлі послідовностями бітів, базуючись на сформованому в першому етапі дереві.

2. Надавати можливість архівувати індивідуальні файли. В результаті процесу архівації отримуємо один файл-архів.

3. Надавати можливість розархівувати файл, сформований процесом архівації. В результаті розархівації отримуємо оригінальний файл.

Зміст **Пояснювальної записки** до курсової роботи:

Індивідуальне завдання

Вступ

1. Огляд…

2. Розробка схеми алгоритму…

3. Розробка програми…

Висновки

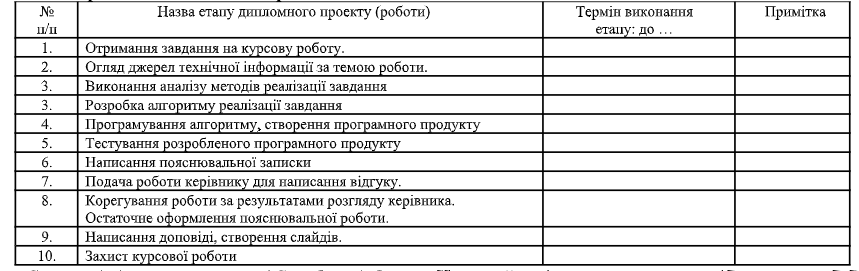
Список літератури

Додатки (за необхідності)

Перелік наочного матеріалу:

­­\_\_\_*Програмний продукт…, пояснювальна записка, презентація результатів роботи*.\_\_\_

**Календарний план виконання роботи:**



Студент(ка) \_\_\_\_\_ / Погоріла К.С. Науковий керівник\_\_\_\_\_\_\_\_ / Воєводін Є.В.

(підпис) (Прізвище, ініціали)

Завдання видане «21» «\_\_жовтня\_\_» 2022 р.

**Зміст**

**Вступ ........................................................................................................................**1

**Розділ 1. Огляд та аналіз алгоритмів стиснення..............................................**2

**1.1 Описання встановленого завдання............................................................**2

**1.2 Методи стиснення інформації.....................................................................**2

**1.2.1 Алгоритми групи KWE ........................................................................**3

**1.2.2 Алгоритм Lossless JPEG .......................................................................**4

**1.2.3 Алгоритм Гаффмана..............................................................................**4

**1.3 Висновок до першого розділу ....................................................................**5

**Розділ 2. Розробка програми ..............................................................................**6

**2.1 Блок-схема ..................................................................................................**6

**2.2 Огляд складових компонентів...................................................................**7

**2.3 Розгляд методів програми .........................................................................**8

**2.4 Висновок до другого розділу ...................................................................**12

**Розділ 3. Реалізація архіватора файлів ...........................................................**13

**3.1 Переваги алгоритму Гаффмана .............................................................**13

**3.2 Реалізація методів кодування та декодування ....................................**14

**3.3 Висновок до третього розділу ................................................................**21

**Загальний висновок ...........................................................................................**22

**Використані джерела ........................................................................................**23

**Вступ**

У даній курсовій роботі розроблено архіватор файлів методом Гаффмана мовою С#.

**Актуальність архіватора:** в роботі з комп’ютером дуже часто використовують програми архівації, задля зменшення розміру файлу.

**Тема:** архіватор файлів методом Гаффмана.

**Мета курсової роботи:** реалізація архіватора файлів на основі даного алгоритму Гаффмана.

**Задача курсової роботи**:

1. Аналіз алгоритму для стиснення даних;
2. Побудова блок-схеми алгоритму.

**Об’єкт:** процес кодування та декодування файлу.

**Предмет роботи:** програмний продукт архіватора.

За останні роки у світі зросла тенденція на економію всього чого тільки можна. Це стосується всіх сфер життя. Люди намагаються заощадити якомога більше часу, місця, пам’яті. Розглянемо більш детально саме економію пам’яті.

Архіватор – це програма, якa виконує архівацію файлів, щоб їх можна було компактно зберігати в зовнішній пaм’яті, a також був можливий процес їх розархівування. Створення архіву відбувається шляхом стиснення даних.

Стиснення – це процес перекодування інформації, який зменшує об’єм файлу. Стиснення є двох видів – із втратами та без втрат. Якщо відбувається стиснення із втратами, то при відновленні файлу з архіву можуть бути невеликі втрати якості початкового файлу, а якщо без втрат – якість незмінна.

**Розділ 1. Огляд та аналіз алгоритмів стиснення**

**1.1 Описання встановленого завдання**

Задача роботи:

1. Аналіз методу.
2. Порівняння з іншими методами стиснення інформації.
3. Складання схеми для даного алгоритму.
4. Реалізація архіватора.

Основним завданням було реалізація архіватора файлів. У сучасному світі доволі гостро стоїть питання збереження та передачі інформації. Інколи потрібно зберегти велику кількість даних на сховище з невеликим обсягом пам’яті (наприклад, флешка). Для цього використовують алгоритми стиснення.

Отже, за мету було поставлено створити та реалізувати архіватор файлів на основі методу Гаффмана, для найбільшої економії місця.

**1.2 Методи стиснення інформації**

Методи стиснення файлів бувають «відкритими» та «комерційними». У першому випадку алгоритм можна вільно досліджувати та використовувати, оскільки він сам по собі є цінністю. У другому випадку використання методу є закритим: алгоритм засекречується через те, що він застосовується лише в окремих програмних продуктах, і несанкціоноване використання може викликати подальші розгляди авторських прав, що власне неодноразово відбувалося в історії розвитку архіваторів.

В даний час існує багато алгоритмів стиснення даних без втрат. Найбільш поширеними це Lossless JPEG, алгоритм Гаффмана, алгоритми групи KWE:

- алгоритм LZ (Лемпеля-Зіва );

- алгоритм LZW (Лемпеля-Зіва-Велча);

**1.2.1 Алгоритми групи KWE**

Існує досить багато реалізацій цього алгоритму, серед яких найбільш поширеними є алгоритм LZ та його модифікація алгоритм LZW, і в результаті стислий файл містить не речення, а послідовність чисел, що суттєво скорочує його розмір.

Алгоритм починає роботу з майже порожнім словником, який містить один закодований рядок (null). Під час зчитування чергового символу вхідної послідовності даних, цей рядок додається до поточного рядку й так продовжується до тих пір, поки поточний ряд не перестане відповідати будь-якій фразі зі словника. Потім кодер видає код, який складається з індексу останнього збігу словника з поточним рядом, й наступного за ним символу, який порушив збіг рядків, а нова фраза, що складається з збігаючого індексу і наступного за ним символу – записується до словника. Якщо ця фраза ще раз з’являється, то її можна використати для побудови довшої фрази.

Алгоритм LZW побудований навколо таблиці фраз (словника), яка замінює рядки символів повідомлення, що стискається, в коди фіксованої довжини. Якщо всі частини словника повністю заповнені, кодування перестає бути адаптивним (кодування відбувається виходячи з фраз, що вже існують у словнику).

Процес стиснення наступним чином. Спочатку послідовно зчитуються символи вхідного потоку та виконується перевірка на наявність такого ж рядку у таблиці. Якщо він є, то зчитується наступний символ, якщо ні – код раніше знайденого рядка зноситься в потік, а в таблицю заноситься рядок й пошук починається знову.

Особливістю даного алгоритму є те, що він вимагає збереження таблиці рядків. Тому для її зберігання було б добре використовувати хеш-таблицю.

**1.2.2 Алгоритм Lossless JPEG**

Цей алгоритм розроблено групою експертів у галузі фотографії (Joint Photographic Expert Group). На відміну від JBIG, Lossless JPEG орієнтований на повнокольорові 24-бітові або 8-бітові у градаціях сірого зображення без палітри. Коефіцієнти стиснення: 20, 2, 1. Lossless JPEG рекомендується застосовувати в тих додатках, де потрібна побітова відповідність вихідного та декомпресованого зображень.

Цей формат розроблявся насамперед для зберігання зображень у медичних цілях, тобто для тих випадків, коли важливо мати велике зображення без найменших втрат якості.

**1.2.3 Алгоритм Гаффмана**

Алгоритм Гаффмана розробив у 1952 році, будучи аспірантом технічного інституту, Девід Гаффман під час того, як писав курсову роботу. В наш час його доволі часто використовують для зберігання фото, відео або тексту. Код Гаффмана дуже рідко використовують окремо від інших методів кодування,

Основна ідея даного методу полягає в тому, що елемент, який повторюється в даній послідовності багато разів отримує нову дуже маленьку послідовність бітів, а елемент, який навпаки зустрічається менше – довгий код (послідовність бітів). Таким чином, один символ може займати всього 1 біт, другий – 2 або 3, третій – 2, 3 або більше.

В даному методі можна виділити два основні етапи:

1. Побудовa дерева;
2. Побудовa відображення код-символ, якa ґрунтується на вже створеному дереві.

Не дивлячись на те, що дана архівація є дуже ефективною, існує проблема з декодуванням, а саме – отримання неоднозначного результату. Задля уникнення цієї проблеми, потрібно скористатись префіксним правилом. Його суть полягає в тому, що декодувати код можна тільки всього лише одним унікальним способом. Префіксний код має таку властивість – якщо в коді є слово А, то для іншого заповненого рядка В слово АВ не буде існувати.

Під час роботи алгоритму вага всіх вузлів дерева постійно зростає. З’являються деякі проблеми. Наприклад, коли вага кореня дерева збільшується й місця в комірці не вистачає, щоб зберігати його. Як правило, це 16-бітне значення, яке не може бути більше за 65535. Ще одна проблема, яка може виникнути це - розмір найдовшого коду буде більшим за вміст комірки, яка використовується для того, щоб передати його в вихідний потік. Так як декодування відбувається згори дерева кодування й рухається вниз, обираючи по одному біту, а кодування ж відбувається навпаки. Тому, коли довжина коду Гаффмана є більшою за розмір «цілого» в бітах, відбувається переповнення.

**1.3 Висновки до першого розділу**

В даному розділі ми більш детально розглянули існуючі методи стиснення інформації, визначили основні задачі даної роботи та створили конкретний план її виконання. Ми з’ясували чим відрізняються більш популярні методи архівації, які є у кожного з них переваги, які недоліки.

**Розділ 2. Розробка програми**

**2.1 Блок-схема**

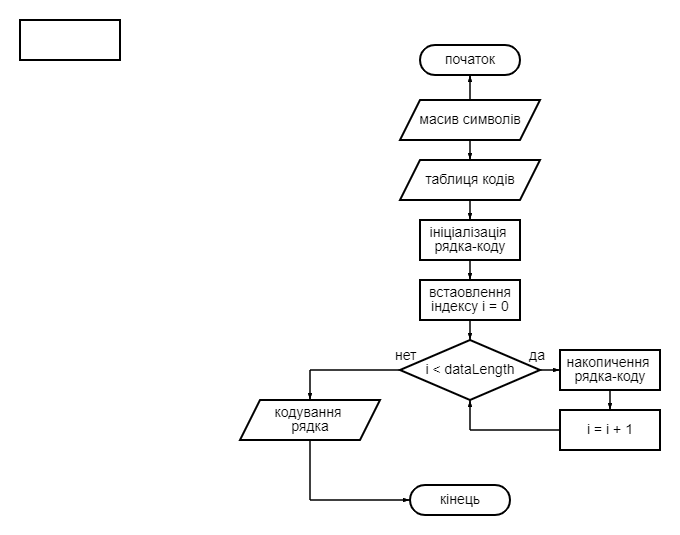
****

Рис. 2.1. Блок-схема кодування

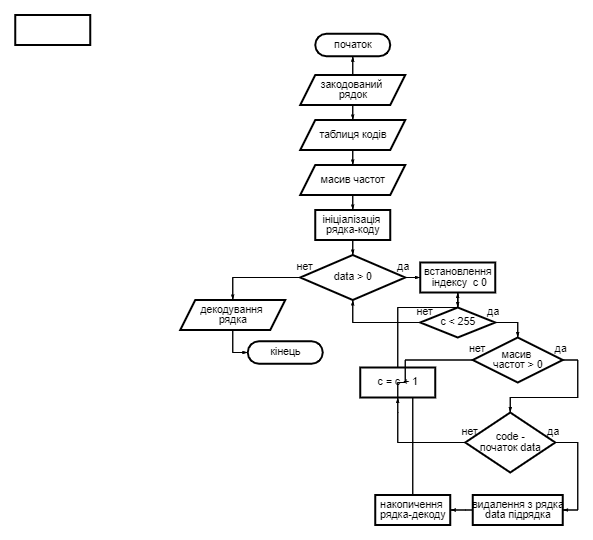
****

Рис. 2.2. Блок-схема декодування

**2.2 Огляд складових компонентів**

Стиснення Гаффмана – це статистичний метод стиснення, який зменшує середню довжину кодового слова символів алфавіту.

Класичний алгоритм Гаффмана на вході отримує частотну таблицю, потім на її основі будується дерево.

Код Гаффмана можна побудувати за таким алгоритмом:

1. Створюємо список символів, в якому один символ розглядається як дерево з вагою, яка рівна частоті його появи в рядку;
2. З отриманого списку обираємо два вузла, які мають найменшу вагу;
3. Формуємо новий вузол, вага якого буде дорівнювати сумі ваг обраних вузлів й додаємо до нього два обрані вузли, які будуть в ролі дітей;
4. До списку додаємо сформовані вузли замість двох об’єднаних;
5. Якщо у списку більше одного вузла, то повторюємо всі попередні дії доти, доки не залишиться тільки один вузол, який і буде вважатись коренем дерева.

Наприклад, у нас є слово Тест, й така частотна таблиця:

Таблиця 2.1

Частотна таблиця

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Т | Е | С |
| Частота | 2 | 1 | 1 |

Дерево для даної таблиці буде мати наступний вигляд:

0

|  |  |
| --- | --- |
| Т | 2 |
| Е | 1 |
| С | 1 |

|  |
| --- |
| ТЕС 4 |

0

|  |
| --- |
| ЕС 2 |

1

1

Запишемо код для кожної літери:

Т – 0, Е – 10, С – 11.

Таким чином, наше закодоване слово буде мати такий вигляд: 010110. Його довжина буде 26 бітів.

**2.3 Розгляд методів програми**

Спочатку нам потрібно підключити усі потрібні бібліотеки, які знадобляться нам для написання нашого архіватора й подальшої роботи в програмі Visual Studio (Рис. 2.3).

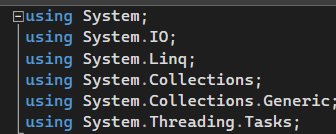


Рис. 2.3. Бібліотеки

Безпосередньо, перед тим, як створювати основу алгоритму Гаффмана – дерево, потрібно написати клас, який буде зберігати в собі всі символи, частоту та посилання на нульовий і одиничний біти нашого дерева (Рис. 2.4).

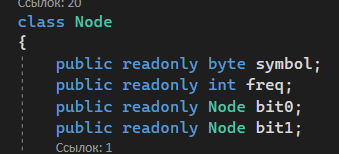


Рис. 2.4. Клас Node

Після того, як ми створили клас Node й записали в нього поля, які будемо використовувати при створенні черги й звісно ж під час написання функцій кодування та декодування, ми повинні написати два конструктори. Перший конструктор буде ініціалізувати частоту та символ. Другий буде ініціалізувати біти й також частоту. В нашому випадку, нам потрібно два конструктори для того, щоб перший ми використовували для створення листа, а другий – для прийому бітів й частот (Рис. 2.5 і 2.6).

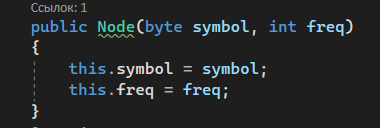


Рис. 2.5. Перший конструктор

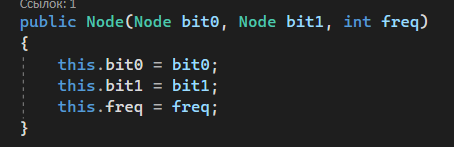


Рис. 2.6. Другий конструктор

Наступним кроком ми маємо створити клас, який буде визначати пріоритетну чергу. У ньому також потрібно буде створити змінну типу int й для неї написати властивість з методом доступу get. Вона буде зберігати розмір черги. Також ми повинні написати відсортований список та бібліотеку. В цьому класі нам також знадобиться користувальницький конструктор, в якому ми вказуємо початковий розмір черги (Рис. 2.7 і 2.8).

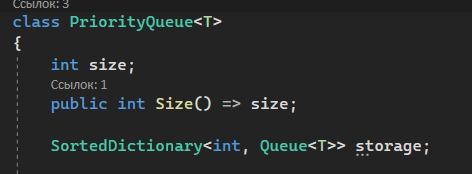


Рис. 2.7. Клас PriorityQueue

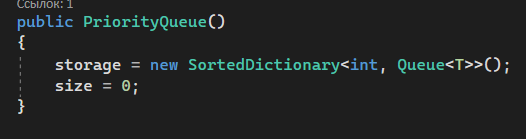


Рис. 2.8 Користувальницький конструктор

Після цього, нам потрібно написати метод Enqueue, який дозволяє додавати нові елементи в чергу. Ми звертаємось до черги storage, додаємо новий елемент й збільшуємо чергу. Але, якщо у нас немає ключа, то створюємо нову чергу (Рис. 2.9).

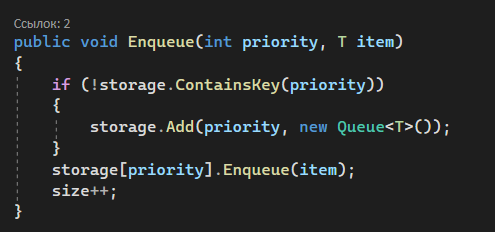


Рис. 2.9. Метод додавання елемента в чергу

Для того, щоб мати змогу побудувати дерево ми маємо створити ще один метод в цьому класі, який дістає з черги елемент за його пріоритетом. В даному методі T Dequeue ми вказуємо, що якщо розмір нашої черги дорівнює нулю, то це означає, що черга пуста. Потім ми створюємо foreach, який буде перебирати всі елементи, дивлячись на їх пріоритет (Рис. 2.10).

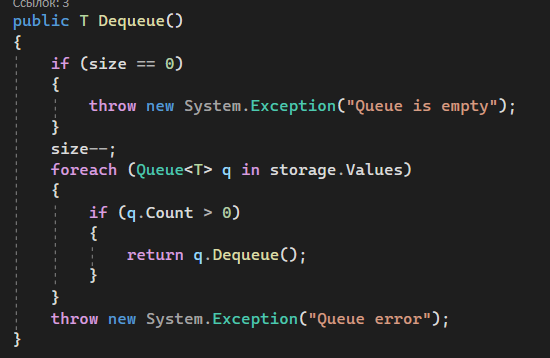


Рис. 2.10. Метод T Dequeue

Тепер нам потрібно створити клас Huffman, в якому ми будемо записувати стискання та розтискання файлу. Спочатку ми записуємо метод CompressFile, який зчитує наш файл, викликає метод, що кодує даний файл й потім записує результат (Рис. 2.11).

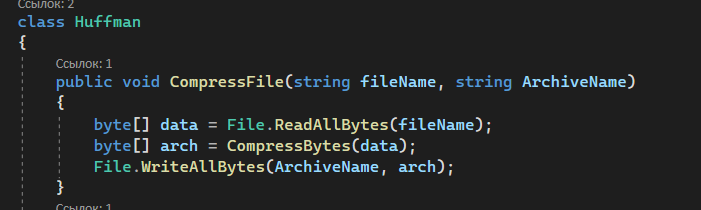


Рис. 2.11. Метод CompressFile

Потім ми також створюємо метод DecompressFile для декодування файлу, але в ньому ми будемо зчитувати не файл, а архів, який будемо розтискати й записувати отриманий результат (Рис. 2.12).

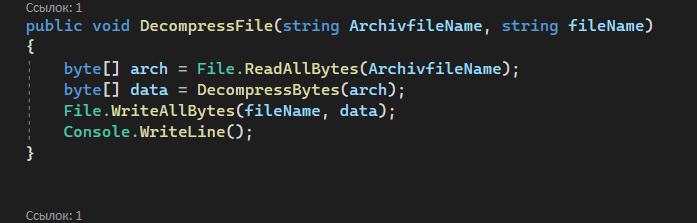


Рис. 2.12 Метод DecompressFile

**2.4 Висновок до другого розділу**

У даному розділі були побудовані блок-схеми, відповідно до коду програми. Було прописано, які саме складові програми потрібно створити в першу чергу.

Також, які методи потрібні для нормального кодування файлу й показано з чого складаються основні компоненти коду (клас вузлів, клас з чергою).

Також було показано бібліотеки, які використовуються.

**Розділ 3. Реалізація архіватора файлів**

**3.1 Переваги алгоритму Гаффмана**

Трохи повернемось до історії створення методу Гаффмана. Як нам вже відомо, алгоритм вигадав Девід А. Гаффман у 1951 році. Йому та його однокурсникам дали вибір – або курсова робота, або випускний екзамен. Професор Роберт М. Фано задав студентам курсову роботу з проблемою пошуку найбільш ефективного двійкового коду. Гаффман спочатку хотів відмовитись, адже ніяк не міг довести найефективніший з якихось кодів, але майже в останню мить йому прийшла ідея. Вона заклечалась у наступному - використання двійкового дерева з сортуванням по частоті. Й після цього було доведено, що даний метод є найбільш ефективним.

При цьому, Девід Гаффман перевершив свого професора Фано, який працював над розробкою кодування з Клодом Шенноном. Його побудова дерева знизу вгору гарантувала оптимальність, на відміну від кодування Шеннона-Фано.

**3.2 Реалізація методів кодування та декодування**

Після того, як ми створили в класі Huffman метод CompressFile й у ньому викликали метод CompressBytes, нам тепер потрібно його створити. Він буде містити в собі наступне: частотний словник, дерево, масив з кодами (якому ми передаємо посилання на елемент – корінь), заголовок та безпосередньо саме стиснення (Рис. 3.1).

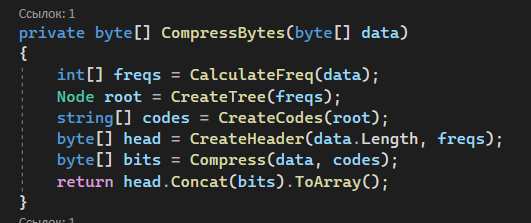


Рис. 3.1. Метод CompressBytes

Тепер ми створюємо метод CalculateFreq, який буде виконувати функцію словника й перебирати байти в масиві та рахувати їх (Рис. 3.2). Також у ньому буде ще один метод, який нормалізує частотну таблицю (щоб максимальне значення було 255).

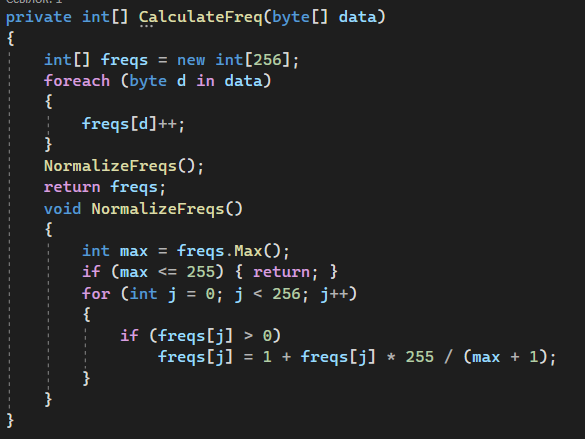


Рис. 3.2. Метод CalculateFreq

Наступний крок – це створення дерева. Для цього ми створюємо метод CreateTree, у якому записуємо екземпляр класу PriorityQueue й потім у циклі for перебираємо всі елементи масиву. Потім у циклі while зображуємо процес створення нових вузлів (Рис. 3.3).

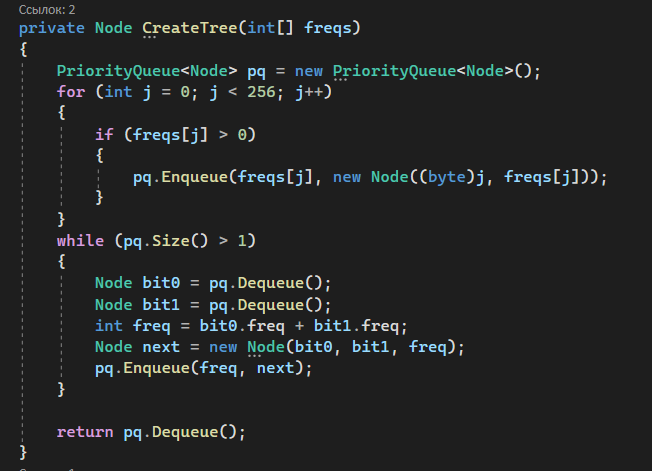


Рис. 3.3. Метод CreateTree (дерево)

Далі ми створюємо метод CreateCodes, у якому буде вкладена функція Next. Вона приймає вузол й накопичує код, але якщо нульовий біт дорівнює null, то записується код, якщо ні – рекурсивно йдемо по нульовому і одиничному бітах (Рис. 3.4).

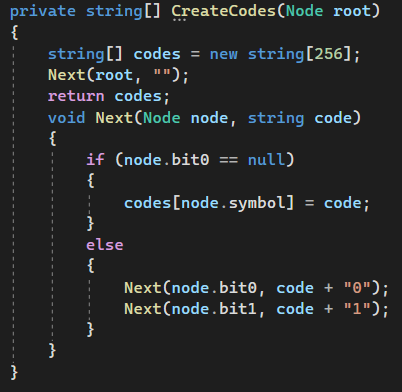


Рис. 3.4 Метод CreateCodes

Також нам потрібно створити метод CreateHeader, який буде виконувати функції таблиці (Рис. 3.5).

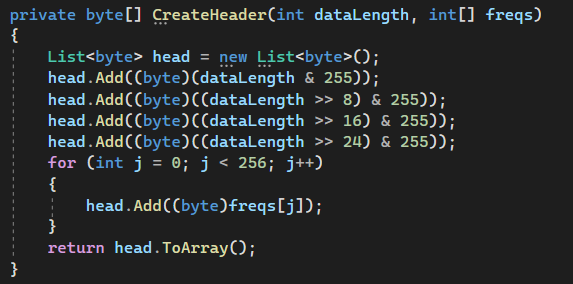


Рис. 3.5. Метод CreateHeader

Для завершення кодування ми створюємо метод Compress, який стискає файл з вхідними даними і кодами. Він містить список байтів, змінні, які накопичують суми чергового байта. Також він має два foreach, перший спочатку перебирає всі символи, другий потім по символу дивиться код. Якщо після цього ще щось залишилось – його ми додаємо у список, потім все це конвертуємо у масив і повертаємо (Рис. 3.6).

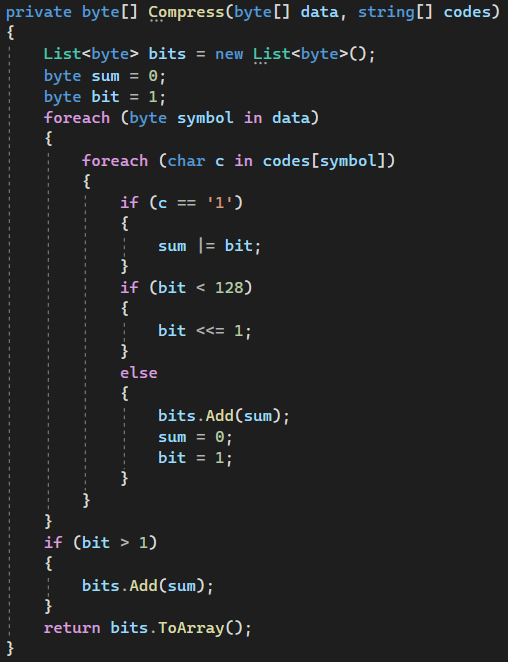


Рис. 3.6. Метод Compress

Тепер розглянемо декодування. Його принцип дуже схожий на метод CompressFile, але замість файлу, ми вже будемо зчитувати архів й його розтискати. Спершу створюємо метод DecompressFile (Рис. 3.7).

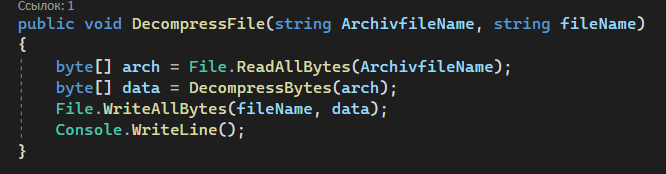


Рис. 3.7. Метод DecompressFile

Створюємо метод DecompressBytes, який буде виконувати функцію декомпресії (Рис. 3.8).

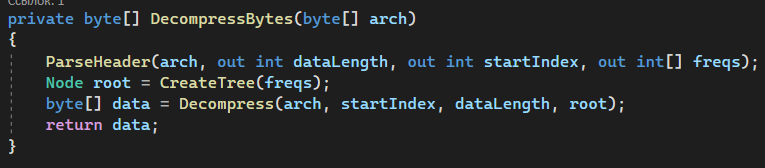


Рис.3.8. Метод DecompressBytes

Тепер нам потрібно створити метод ParseHeader, який зчитує наш заголовок (Рис. 3.9).

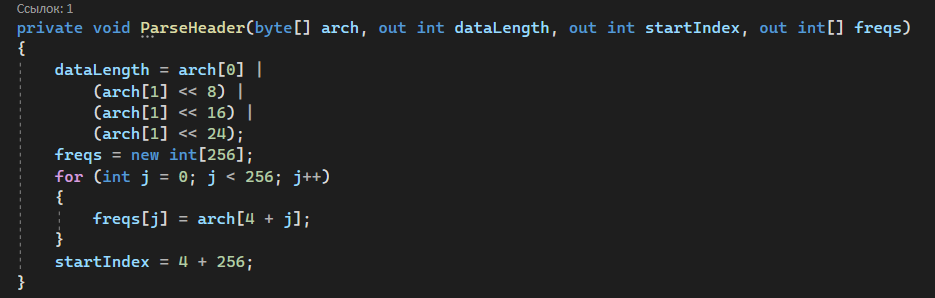


Рис. 3.9. Метод ParseHeader

І останнє, що нам залишається – це написати метод Decompress, який і повертає наш закодований файл у первинний вигляд (Рис. 3.10).

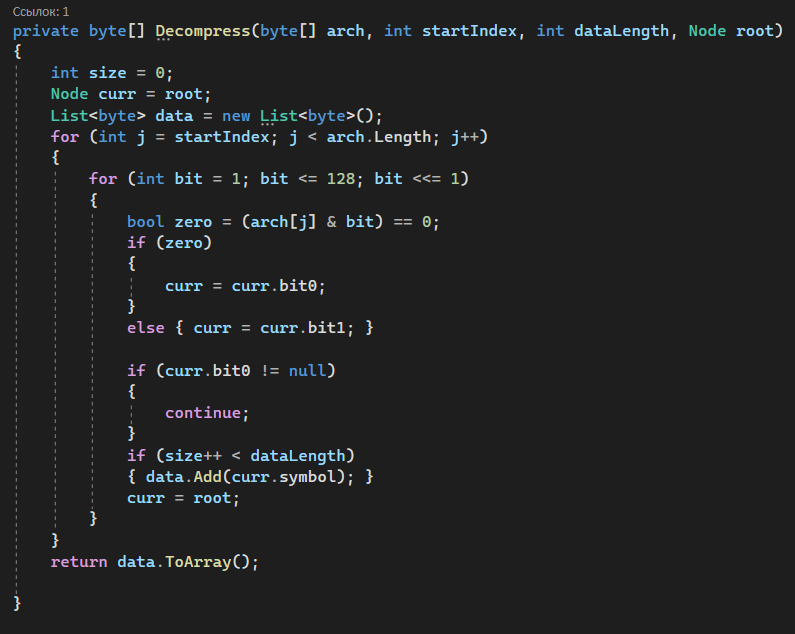


Рис. 3.10. Метод Decompress

У методі Main створюємо екземпляр класу Huffman й за допомогою нього викликаємо методи стиснення та розтискання (Рис. 3.11).

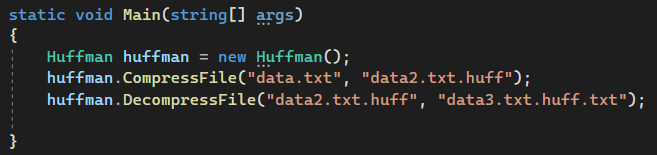


Рис. 3.11. Метод Main

Архіватор на основі коду Гаффмана готовий. Тепер можна відобразити його роботу вже з текстовим файлом. У нас є текстовий файл з розміром 1696 байт (Рис. 3.12).

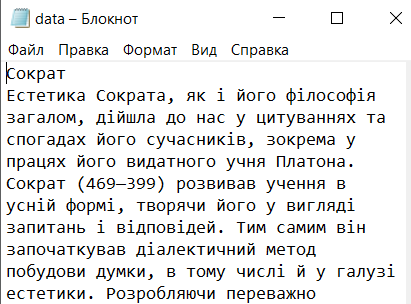


Рис. 3.12. Вхідний файл data

Після запуску програми створилось ще два файли. Один – закодований файл з розміром 1132 байти (Рис. 3.13), другий – декодований файл з розміром 1696 байт (має вигляд вхідного файлу) (Рис. 3.14).

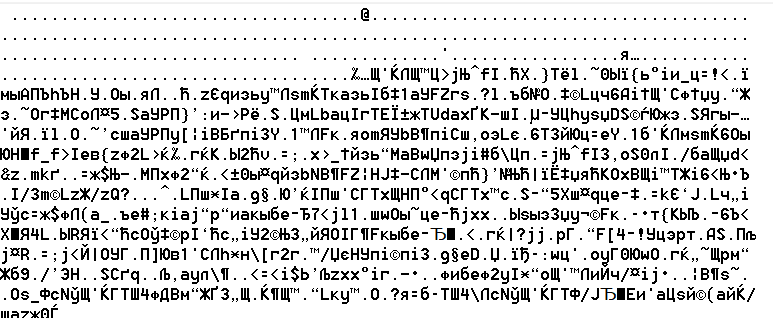


Рис. 3.13. Закодований файл data.txt.huff (двійковий вигляд)

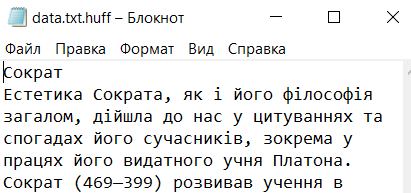


Рис. 3.14. Декодований файл data.txt.huff

Загальний вигляд розмірності файлів (Рис. 3.15).

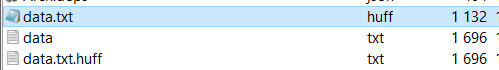


Рис. 3.15. Розміри файлів

**3.3 Висновок до третього розділу**

У даному розділі було більш детально розглянуто перевагу даного методу кодування файлів.

Потім було розписано про методи, які потрібні для стиснення файлу, у якій послідовності вони працюють. Також описано методи, які використовуються для декодування файлів.

Також, було показано як саме відбувається стиснення та розтискання файлу, що дійсно закодований файл має менший розмір, й зазначено те, що він має знаходитись безпосередньо у папці, що зберігає файл програмного продукту, тобто файл з кодом програми.

**Загальний висновок**

Під час виконання даної курсової роботи я дізналась велику кількість нової інформації про принципи роботи архіваторів, як саме відбувається процес стиснення інформації.

В документації було проаналізовано та сформульовано основні задачі даної роботи, її мету. Розроблено план виконання роботи.

Також було розглянуто ще декілька алгоритмів кодування інформації для того, щоб мати уявлення чим саме код Гаффмана відрізняється від інших. У нашому випадку від тих, які були наведені у документації.

Також було розглянуто поетапне створення коду програмного продукту. Описано з чого він складається, з яких методів, для чого ці методи, як вони працюють. Даний архіватор складається з трьох основних класів – Node, PriorityQueue і Huffman.

У другому розділі було створено дві блок-схеми, які показують процес кодування та декодування, відповідно до коду програми. Також розписано про складові коду. А саме те, що спочатку ми створюємо клас Node, який зберігає в собі інформацію про вузли, символи, частоту та біти. Потім клас PriorityQueue, який безпосередньо взаємодіє з чергою (списком). І основний клас Huffman у якому й відбуваються наші дві основні функції - кодування та декодування файлу.

У третьому розділі ми ще раз згадали, про перевагу саме методу Гаффмана й більш детально описали методи, які використовувалися для стиснення та розстикання інформації в файлі.

Отже, документація до курсової роботи містить у собі алгоритм дій поетапного створення архіватора на основі коду Гаффмана, його актуальність й реалізація даної програми.

**Використані джерела**

1. Ю.Є. Гребенович, Б.О. Онищенко, О.О. Супруненко – «МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ТА ОФОРМЛЕННЯ КУРСОВОЇ РОБОТИ», 2015 - С.32.

2. <https://habr.com/ru/post/144200/>.

3. <https://www.techiedelight.com/ru/huffman-coding/>.

4. <http://algolist.ru/compress/standard/huffman.php>.