**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу**

**Кафедра системного проектування**

**Курсова робота**

**з дисципліни «Алгоритмізація та програмування»**

**на тему: «Задачі на графах»**

Виконав:

студент I курсу, групи ДА-13

Світличний Станіслав Сергійович \_\_\_\_\_\_\_\_\_\_

Керівник:

Старший викладач

Безносик Олександр Юрійович \_\_\_\_\_\_\_\_\_\_

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2021 рік

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ІПСА КПІ ім. І. Сікорського** | | | | | | |
| (назва вищого закладу освіти) | | | | | | |
| Кафедра | | ***Математичних методів системного аналізу*** | | | | |
| Дисципліна | | ***Алгоритмізація та програмування*** | | | | |
| Галузь знань | | ***122 Комп’ютерні науки*** | | | | |
| Курс | ***перший*** | | Група | ***ДА-13*** | Семестр | ***другий*** |

**ЗАВДАННЯ**

**на курсовий проект(роботу) студента**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Світличний Станіслав Сергійович | | | | | |
| (прізвище, ім’я, по батькові) | | | | | |
| 1. Тема проекту(роботи) | Задачі на графах | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| 2. Строк здачі студентом закінченого проекту(роботи) | | | ***25.05.2021 р.*** | | |
| 3. Вихідні дані до проекту(роботи) | |  | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| 4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці) | | | | |  |
| ***1. Постановка задачі*** | | | | | |
| ***2. Алгоритм розв’язку задачі*** | | | | | |
| ***3. Загальна блок-схема алгоритму та опис алгоритму*** | | | | | |
| ***4. Опис програмного продукту.*** | | | | | |
| ***5. Тестування програми*** | | | | | |
| ***6. Висновки.*** | | | | | |
| ***7. Список використаної літератури.*** | | | | | |
| ***Додаток А. Блок-Схема. Додаток Б. Лістинг.*** | | | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) | | | |  | |
| ***1. Загальна блок-схема алгоритму.*** | | | | | |
| ***2. Ілюстрації роботи програми.*** | | | | | |
| 6. Дата видачі завдання | 19.03.2021 | | | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№/п** | **Назва етапів курсового проекту (роботи)** | **Строк виконання**  **етапів роботи** | **Примітка** |
| 1. | Вибір теми курсової роботи. Опрацювання літератури. Оформлення листа Завдання. | 2,3-й тиждень лютого |  |
| 2. | Аналіз постановки задачі. Узгодження з керівником попереднього плану роботи | 3,4-й тиждень лютого |  |
| 3. | Вибір та дослідження методів та структур даних, Розробка загального алгоритму | 1-й тиждень березня |  |
| 4. | Перше узгодження з керівником. | 1-й тиждень березня |  |
| 5. | Проектування інтерфейсу. Розробка алгоритмів окремих блоків. | 2-й тиждень березня |  |
| 6. | Друге узгодження з керівником. | 2-й тиждень березня |  |
| 7. | Програмна реалізація. | 3,4-й тижні березня |  |
| 8. | Демонстрація першого варіанту.  Трете узгодження з керівником. | 1-й тиждень квітня |  |
| 9. | Доопрацювання програми. Заключне тестування програми. | 2,3-й тижні квітня |  |
| 10. | Аналіз результатів. Оформлення звіту. | до 2-го тижня травня |  |
| 11. | Захист та демонстрація курсової роботи. | до 25.05– «А»  до 30.05 –«В,С»  до 05.06 – «D,E» |  |

|  |  |  |
| --- | --- | --- |
| Студент |  | Світличний Станіслав Сергійович |
|  | (підпис) | (прізвище, ім’я, по батькові) |

|  |  |  |
| --- | --- | --- |
| Керівник |  | Безносик Олександр Юрійович |
|  | (підпис) | (прізвище, ім’я, по батькові) |

|  |  |
| --- | --- |
| 25.05.2021 |  |
| (дата) |  |

Зміст

[ВСТУП 5](#_Toc72878763)

[РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ 7](#_Toc72878764)

[1.1 Теоретичні відомості 7](#_Toc72878765)

[1.1.1 Стиснення без втрат 7](#_Toc72878766)

[1.1.2 Стиснення з втратами 9](#_Toc72878767)

[1.1.3 Ентропійне кодування 10](#_Toc72878768)

[1.1.4 Кодування словником 11](#_Toc72878769)

[1.1.5 Кодування довжин серій 11](#_Toc72878770)

[1.1.6 Код Хаффмана 12](#_Toc72878771)

[1.1.7 Алгоритм Лемпеля-Зіва 15](#_Toc72878772)

[1.1.8 Перетворення Берроуза-Вілера 17](#_Toc72878773)

[1.1.9 Перехід до початку 18](#_Toc72878774)

[1.2 Постановка задачі проектування 19](#_Toc72878775)

[1.3 Вибір середовища розробки 20](#_Toc72878776)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 22](#_Toc72878777)

[2.1 Математична постановка задачі 22](#_Toc72878778)

[2.2 Алгоритм розв’язку задачі 22](#_Toc72878779)

[РОДІЛ 3. ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ 23](#_Toc72878780)

[3.1 Логіка та структура 23](#_Toc72878781)

[3.2 Опис інтерфейсу 23](#_Toc72878782)

[3.3 Результати роботи програмного продукту 23](#_Toc72878783)

[ВИСНОВОК 24](#_Toc72878784)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 25](#_Toc72878785)

# ВСТУП

Стиснення даних — це процедура перекодування даних, яка проводиться з метою зменшення їхнього обсягу, розміру, об'єму.

Стиснення базується на усуненні надлишку інформації, яка міститься у вихідних даних. Наприклад, повторення в тексті фрагментів (наприклад, слів природної або машинної мови). Подібний надлишок зазвичай усувається заміною повторюваних послідовностей коротшим значенням (кодом). Інший вид надлишковості пов'язаний з тим, що деякі значення в даних, що стискаються, трапляються частіше інших, при цьому можна замінювати дані, що часто трапляються, коротшими кодами, а ті, що рідко, довшими (ймовірнісне стиснення). Стиснення даних, які не мають властивості надлишку (наприклад випадковий сигнал чи шум), неможливе. Також, зазвичай, неможливо стиснути зашифровану інформацію [1].

Тема є актуальною з моменту появи комп’ютерів і вивчається великою кількістю науковців вже протягом декількох десятиліть.

Метою курсової роботи є вивчення деяких алгоритмів стиснення так кодування даних. Порівняння їх ефективності та швидкодії.

Основним завданням є реалізація власної програми для ефективного стиснення текстових файлів. Це включає розробку алгоритмів та розробку графічного користувацького інтерфейсу.

Отриману програму можна буде використовувати для економії місця на комп’ютері завдяки стисканню текстових файлів

При виконанні роботи було використано таке програмне забезпечення:

* Операційна система — Microsoft Windows;
* Основне середовище розробки — Visual Studio 2019;
* Текстовий редактор для підготовки та оформлення курсової роботи — Microsoft Word 2010

Робота складається зі вступу, трьох розділів, висновків, додатків та списку використаних джерел.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

## 1.1 Теоретичні відомості

Алгоритми стиснення поділяють на такі види:

* Стиснення без втрат — можливо відновлення вихідних даних без спотворень.
* Стиснення з втратами — відновлення можливе з незначними спотвореннями [1].

### 1.1.1 Стиснення без втрат

Стиснення без втрат — метод стиснення даних, при використанні якого закодована інформація може бути повністю відновлена зі стиснутих даних. Навпаки, стиснення з втратами дозволяє лише відновлення даних, які є тільки наближенням до початкових даних. Для кожного з типів цифрової інформації, як правило, існують свої оптимальні алгоритми стиснення без втрат.

Стиснення даних без втрат використовується в багатьох програмах. Наприклад, воно використовується в усіх файлових архіваторах. Воно також використовується як компонент в стисненні з втратами.

Стиснення без втрат використовується, коли важливо, щоб відновленні дані були ідентичні оригіналу. Типовий приклад — виконуваний файл або джерельний код, текстовий файл. Деякі графічні файлові формати, наприклад, PNG та GIF використовують тільки стиснення без втрат, тоді як формати TIFF та MNG можуть використовувати стиснення як з втратами, так й без втрат. Формати стиснення звуку без втрат використовується для архівування або виробничих цілей, в той час, як менші формати стиснення аудіо з втратами використовуються в аудіопрогравачах та в ситуаціях коли простір для зберігання інформації обмежений або нема потреби в точному відтворенні інформації.

У загальних рисах значення стиснення без втрат полягає в пошуку закономірності в початкових даних і з її урахуванням генерації іншої послідовності, яка повністю описує початкову. Наприклад, для кодування бінарних послідовностей, в яких багато нулів та мало одиниць, ми можемо використати таку заміну:

00 → 0

01 → 10

10 → 110

11 → 111

В такому випадку 16 бітів

00 01 00 00 11 10 00 00

будуть перетворені у 13 бітів

0 10 0 0 111 110 0 0

Така заміна є видом префіксного коду, тобто має таку особливість: якщо ми запишемо стиснений рядок без пропусків, ми все одно зможемо розставити в ній пропуски — а тому, і відновити початкову послідовність. Найбільш відомим префіксним кодом є код Хаффмана.

Більшість алгоритмів стиснення без втрат працюють у дві стадії: на першій генерується статистична модель для вхідних даних, друга представляє вхідні дані в бітовому вигляді, використовуючи модель для отримання «ймовірнісних» даних (тобто таких, що зустрічаються часто).

Статистичні моделі алгоритмів для тексту (чи текстових бінарних даних, таких як виконувальні файли) містять:

* Перетворення Берроуза-Вілера
* LZ77 і LZ78
* LZW

Алгоритми стиснення без втрат не можуть гарантувати стиснення для усіх видів вхідних даних. Іншими словами, для будь-якого алгоритму стиснення без втрат, існує такий набір вхідних даних, які не зменшуються після обробки алгоритмом, а навпаки — збільшуються.

Будь-який алгоритм, що робить деякі файли меншими, повинен робити деякі файли більшими, але не обов'язково, що вони стануть дуже великими. Практично використовуються алгоритми, що забезпечують собі механізм «виходу», що зупиняє кодування файлів, які можуть стати більшими після дії стиснення. Теоретично, один лиш додатковий біт потрібен, щоб сказати декодеру, що кодування вимкнене для усіх вхідних даних; проте, більшість кодувальних алгоритмів використовують більше ніж один повний байт для цієї цілі. Наприклад, файли стисненні алгоритмом DEFLATE ніколи не збільшуються більше ніж на 5 байтів на 65 535 байтів вхідних даних.

Файли, що складаються з випадкових даних, не можуть бути успішно стиснені ні одним із розумних алгоритмів: дійсно, результат такої дії використовується для визначення концепції випадковості в теорії алгоритмічної складності [2].

### 1.1.2 Стиснення з втратами

Стиснення з втратами — метод стиснення даних, при якому розпакований файл відрізняється від оригіналу, проте може бути корисним для використання. Стиснення із втратами найчастіше використовується для мультимедіа-даних (аудіо, відео, зображення), особливо для потокової передачі даних та в телефонії. В цьому контексті такі методи часто називаються кодеками.

Існують дві основних схеми стиску із втратами:

* У трансформуючих кодеках стиснення беруться фрейми зображень або звуку, розрізуються на невеликі сегменти, трансформуються в новий базисний простір і здійснюється квантування. Результат потім стискується ентропійними методами.
* У предиктивних кодеках стиснення попередні і/або наступні дані використаються для того, щоб пророчити поточний фрейм зображення або звуку. Помилка між передбаченими даними і реальними разом з додатковою інформацією, необхідною для здійснення предикту, потім квантизується і кодується.

У деяких системах ці дві техніки комбінуються шляхом використання трансформуючих кодеків для стиску помилкових сигналів, згенерованих на стадії пророкування.

Перевага методів стиснення із втратами над методами стиску без втрат полягає в тому, що перші істотно перевершують по ступені стиску, продовжуючи задовольняти поставленим вимогам. Методи стиску із втратами часто використаються для стиску звуку або зображень. У таких випадках розпакований файл може дуже сильно відрізнятися від оригіналу на рівні порівняння «біт у біт», але практично не відрізняється для людського вуха або ока в більшості практичних застосувань.

Багато методів фокусуються на особливостях будови органів почуттів людини. Психоакустична модель визначає те, наскільки сильно звук може бути стиснений без погіршення сприйманої якості звуку. Помітні для людського вуха або ока недоліки, що виникли через стиснення із втратами, відомі як артефакти стиску [3].

### 1.1.3 Ентропійне кодування

Ентропійне кодування — кодування послідовності значень з можливістю однозначного відновлення з метою зменшення обсягу даних (довжини послідовності) за допомогою усереднення ймовірностей появи елементів у закодованій послідовності.

Передбачається, що до кодування окремі елементи послідовності мають різну ймовірність появи. Після кодування в результуючій послідовності ймовірності появи окремих символів практично однакові (ентропія на символ максимальна).

Розрізняють декілька варіантів кодів:

* Зіставлення кожному елементу вхідної послідовності різного числа елементів результуючої послідовності. Чим більше вірогідність появи вхідного елемента, тим коротше відповідна результуюча послідовність. Прикладом можуть служити код Шеннона-Фано, код Хаффмана.
* Зіставлення кількох елементів вхідної послідовності фіксованого числа елементів кінцевої послідовності. Прикладом є код Танстола.
* Інші структурні коди, засновані на операціях з послідовністю символів [4].

### 1.1.4 Кодування словником

Метод стиснення з використанням словника — розбиття даних на слова і заміна їх на індекси в словнику. В даний час це найбільш поширений підхід для стиснення даних, він є природним узагальненням RLE.

У найбільш поширеному варіанті реалізації словник поступово поповнюється словами з вихідного блоку даних в процесі стиснення.

Основний параметр будь-якого словникового методу — це розмір словника. Чим більше словник, тим вище ефективність. Однак для неоднорідних даних надмірно великий розмір може бути шкідливий, так як при різкій зміні типу даних словник буде заповнений неактуальними словами. Для ефективної роботи цих методів при стисненні потрібна додаткова пам'ять — приблизно на порядок більше, ніж потрібно для вихідних даних словника. Суттєва перевага словникових методів — проста і швидка процедура розпакування. Додаткова пам'ять при цьому не потрібно. Така особливість вкрай важлива, якщо необхідний оперативний доступ до даних [5].

### 1.1.5 Кодування довжин серій

Кодування довжин серій або Кодування повторів (RLE) — простий алгоритм стиснення даних, який оперує серіями даних, тобто послідовностями, в яких один і той же символ зустрічається кілька разів поспіль. При кодуванні рядок однакових символів, що становлять серію, замінюється рядком, який містить сам повторюваний символ і кількість його повторів.

Розглянемо зображення, що містить простий чорний текст на суцільному білому фоні. Тут буде багато серій білих пікселів в порожніх місцях, і багато коротких серій чорних пікселів в тексті. Нехай, наприклад, дано довільний рядок чорно-білого зображення. Тут B позначає чорний піксель, а W — білий:

WWWWBBBBWWWWW

Якщо ми застосуємо просте кодування довжин серій до цього рядка, то отримаємо таке:

4W4B5W

Таким чином, код подає вихідні 13 символів у вигляді всього лише 6

Однак, у випадку, якщо рядок містить велику кількість неповторюваних символів, його обсяг може зрости.

ABCABCABCDDDFFFFFF

1A1B1C1A1B1C1A1B1C3D6F

Проблема вирішується досить просто. Алфавіт, в якому записані довжини серій, розділяється на дві (зазвичай рівні) частини. Алфавіт цілих чисел можна, наприклад, розділити на дві частини: додатні і від'ємні. Додатні використовують для запису кількості повторюваних однакових символів, а від'ємні — для запису кількості неоднакових.

-9ABCABCABC3D6F

Оскільки чисельні типи даних на комп'ютері завжди мають певну межу, виникає ще одна проблема. Припустимо, ми використовуємо тип signed char для запису довжин серій. Тоді ми не можемо записати серію, довшу ніж 127 символів, однією парою «довжина-символ». Якщо поспіль записано 256 символів A, їх розділяють на мінімальну кількість груп:

127A127A2A

Вочевидь, що таке кодування ефективно для даних, що містять велику кількість серій, наприклад, для простих графічних зображень, таких як іконки та графічні малюнки. Однак це кодування погано підходить для зображень з плавним переходом тонів, таких як фотографії [6].

### 1.1.6 Код Хаффмана

Алгоритм Хаффмана — адаптивний жадібний алгоритм оптимального префіксного кодування алфавіту з мінімальною надмірністю. Був розроблений аспірантом Массачусетського технологічного інституту Девідом Хаффманом при написанні ним курсової роботи та надрукований в статті 1952 року «A Method for the Construction of Minimum-Redundancy Codes». В даний час використовується в багатьох програмах стиснення даних без втрат.

Ідея алгоритму така: знаючи ймовірності появи символів у повідомленні, можна описати процедуру побудови кодів змінної довжини, що складаються з цілої кількості бітів. Символам з більшою ймовірністю ставляться у відповідність коротші коди. Коди Хаффмана володіють властивістю префіксності (тобто жодне кодове слово не є префіксом іншого), що дозволяє однозначно їх декодувати.

Класичний алгоритм Хаффмана на вході отримує таблицю частот з якими зустрічаються символи у повідомленні. Далі на підставі цієї таблиці будується дерево кодування Хаффмана.

1. Символи вхідного алфавіту утворюють список вільних вузлів. Кожен лист має вагу, яка може бути рівною або ймовірності, або кількості входжень символу у стиснене повідомлення.
2. Вибираються два вільних вузли дерева з найменшими вагами.
3. Створюється їхній батьківський вузол з вагою, рівною їх сумарній вазі.
4. Вузол-батько додається в список вільних вузлів, а два його нащадки видаляються з цього списку.
5. Одній дузі, котра виходить з вузла батька, ставиться у відповідність біт 1, інший — біт 0.
6. Кроки, починаючи з другого, повторюються доти, поки в списку вільних вузлів не залишиться тільки один вільний вузол. Він і буде вважатися коренем дерева.

Розглянемо приклад описаний у таблиці 1.

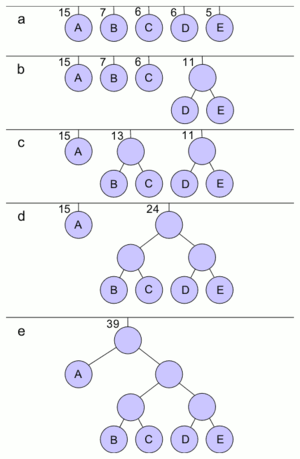
Таблиця 1 – Частоти символів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 | 7 | 6 | 6 | 5 |
| A | B | C | D | E |

Цей процес можна подати як побудову дерева (описано на рисунку 1), корінь якого — символ з сумою ймовірностей об'єднаних символів, отриманий при об'єднанні символів з останнього кроку, його n0 нащадків — символи з попереднього кроку і т. д.

Щоб визначити код для кожного із символів, що входять у повідомлення, потрібно пройти шлях від листка дерева, який відповідає поточному символу, до його кореня, накопичуючи біти при переміщенні по гілках дерева (перша гілка в шляху відповідає молодшому біту). Отримана таким чином послідовність бітів є кодом даного символу, записаним у зворотному порядку.

Для даної таблиці символів коди Хаффмана будуть виглядати так, як описано в таблиці 2.



*Рис. 1 – Приклад побудови дерева Хаффмана*

Таблиця 2 – Коди символів утворених алгоритмом Хаффмана

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 100 | 101 | 110 | 111 |

Оскільки жоден з отриманих кодів не є префіксом іншого, вони можуть бути однозначно декодовані при читанні їх з потоку. Крім того, найбільш частий символ повідомлення А закодований найменшою кількістю біт, а найбільш рідкісний символ E — найбільшою.

При цьому загальна довжина повідомлення, що складається з наведених у таблиці символів, складе 87 біт (в середньому 2,2308 біта на символ). При використанні рівномірного кодування загальна довжина повідомлення склала б 117 біт (рівно 3 біти на символ). Зауважимо, що ентропія джерела, яке незалежним чином породжує символи із зазначеними частотами, складає ~ 2,1858 біта на символ, тобто надмірність побудованого для такого джерела коду Хаффмана, що розуміється, як відмінність середнього числа біт на символ від ентропії, становить менше 0,05 біта на символ.

Класичний алгоритм Хаффмана має ряд істотних недоліків. По-перше, для відновлення вмісту стиснутого повідомлення декодер повинен знати таблицю частот, якою користувався кодер. Отже, довжина стиснутого повідомлення збільшується на довжину таблиці частот, яка повинна надсилатися попереду даних, що може звести нанівець всі зусилля щодо стиснення повідомлення. Крім того, необхідність наявності повної частотної статистики перед початком власне кодування вимагає двох проходів по повідомленню: одного для побудови моделі повідомлення (таблиці частот і дерева Хаффмана), іншого для власне кодування. По-друге, надмірність кодування обертається на нуль лише в тих випадках, коли ймовірності кодованих символів є оберненими степеням числа 2. По-третє, для джерела з ентропією, що не перевищує 1 (наприклад, для двійкового джерела), безпосереднє застосування коду Хаффмана позбавлене сенсу.

Стиснення даних Хаффмана застосовується під час стиснення фото- і відеозображень (JPEG, стандарти стиснення MPEG), в архіваторах (PKZIP, LZH та інших), в протоколах передачі даних MNP5 і MNP7 [7].

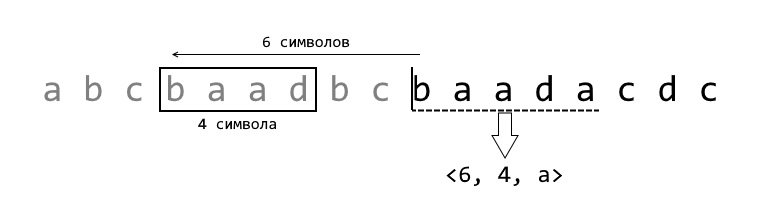
### 1.1.7 Алгоритм Лемпеля-Зіва

LZ77 і LZ78, алгоритм Лемпеля-Зіва — універсальний алгоритм стискування даних без втрат, створений у 1977—1978 роках Авраамом Лемпелем і Яковом Зівом. Алгоритм розроблений так, щоб його можна було швидко реалізувати, але він не обов'язково є оптимальним, оскільки не проводить жодного аналізу вхідних даних. 1984 року Террі Велчем опублікована покращена реалізація алгоритму LZ78 — алгоритм Лемпеля-Зіва-Велча (LZW) [8].

У кодованих рядках часто містяться збігаються довгі підрядки. Ідея, що лежить в основі LZ77, полягає в заміні повторень на посилання на позиції в тексті, де такі підрядки вже зустрічалися.

Інформацію про повторення можна закодувати парою чисел — зміщенням назад від поточної позиції (offset) і довжиною збігається підрядка (length). В такому випадку, наприклад, рядок pabcdeqabcde може бути представлена як pabcdeq⟨6,5⟩. Вираз ⟨6,5⟩ означає «повернися на 6 символів назад і виведи 5 символів».

Алгоритм LZ77 кодує посилання блоками з трьох елементів — ⟨offset, length, next⟩. На додаток до двох вже описаним елементам, новий параметр next означає перший символ після знайденого збігається фрагмента. Якщо LZ77 не вдалося знайти збіг, то вважається, що offset = length = 0. Приклад роботи алгоритму наведено на рисунку 2.



*Рис. 2 – Алгоритм LZ77*

Для ефективного пошуку повторів в LZ77 застосовується метод «ковзного вікна» – збіги шукаються не на всьому обробленому префікс, а в невеликому буфері, що складається з останніх оброблених символів. Зазвичай довжина буфера дорівнює 2, 4 або 32KB. Таким чином, при великих обсягах введення алгоритм витрачає менше часу за рахунок того, що проглядається не вся вихідна рядок. З іншого боку, занадто маленька довжина словника може привести до того, що розташовані далеко один від одного збігу (на більшій відстані, ніж довжина словника) не будуть враховані, і кодування стане менш оптимальним.

Алгоритм LZ78 має трохи іншу ідею: цей алгоритм в явному вигляді використовує словниковий підхід, генеруючи тимчасовий словник під час кодування і декодування.

Спочатку словник порожній, а алгоритм намагається закодувати перший символ. На кожній ітерації ми намагаємося збільшити кодується префікс, поки такий префікс є в словнику. Кодові слова такого алгоритму будуть складатися з двох частин — номера в словнику найдовшого знайденого префікса (pos) і символу, який йде за цим префіксом (next). При цьому після кодування такої пари префікс з приписаним символом додається в словник, а алгоритм продовжує кодування з наступного символу [9].

### 1.1.8 Перетворення Берроуза-Вілера

Перетворення Берроуза-Вілера (BWT) — метод перестановки символів у стрічці T в іншу стрічку BWT(T) таким чином, що із BWT(T) можна отримати початкову послідовність, але в той же час вона краще надається до стиснення.

Щоб знайти BWT(T) від стрічки T довжиною n слід згенерувати n циклічних ротацій цієї стрічки, відсортувати їх у лексикографічному порядку і отримати нову стрічку з останніх символів цих ротацій. Якщо вихідна стрічка містить багато повторів, то трансформована міститиме багато серій (послідовностей, в яких один і той же символ зустрічається кілька разів поспіль).

Примітно в BWT не те, що він генерує більш легко кодований вихід — звичайне сортування зробило би це, — а в тому, що він робить це оборотно, дозволяючи оригінальному документу відтворити дані з останнього стовпця [10].

Існує декілька реалізацій алгоритму оберненого перетворення. Найпростішою є побудова на основі вхідного останнього стовпця матриці усіх циклічних ротацій шляхом додавання початкового стовпця і подальшого сортування як продемонстровано у таблиці 3.

Таблиця 3 — Обернене перетворення рядка BCABAA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Додавання 1 | Сортування 1 | Додавання 2 | Сортування 2 | Додавання 3 | Сортування 3 |
| B  C  A  B  A  A | A  A  A  B  B  C | BA  CA  AA  BB  AB  AC | AA  AB  AC  BA  BB  CA | BAA  CAB  AAC  BBA  ABB  ACA | AAC  ABB  ACA  BAA  BBA  CAB |
| Додавання 4 | Сортування 4 | Додавання 5 | Сортування 5 | Додавання 6 | Сортування 6 |
| BAAC  CABB  AACA  BBAA  ABBA  ACAB | AACA  ABBA  ACAB  BAAC  BBAA  CABB | BAACA  CABBA  AACAB  BBAAC  ABBAA  ACABB | AACAB  ABBAA  ACABB  BAACA  BBAAC  CABBA | BAACAB  CABBAA  AACABB  BBAACA  ABBAAC  ACABBA | AACABB  ABBAAC  ACABBA  BAACAB  BBAACA  CABBAA |

Оскільки відповідну таблицю можна отримати з будь-якого рядка, що залишилися в останньому стовпці, то при кодуванні слід зберігати також додаткову інформацію як наприклад індекс початкового рядку.

На жаль подібна реалізація буде працювати вкрай повільно, через те, що асимптотична складність алгоритму O(N3logN), де N — кількість символів. На декодування досить великих файлів може піти надзвичайно велика кількість часу. Є декілька оптимізацій алгоритму, що були використані у курсовій роботі, які дозволяють алгоритму виконуватися за O(NlogN) чи навіть за лінійний час, але для розуміння алгоритму «наївної» версії достатньо.

### 1.1.9 Перехід до початку

Перехід до початок (MTF) — перетворення для кодування даних, розроблене для поліпшення продуктивності ентропійного кодування. При гарній реалізації воно досить швидко для включення як додатковий крок в алгоритмах стиснення даних. Також може використовуватися спільно з BWT, перетворенням Берроуза-Вілера.

Основною ідеєю перетворення є заміна кожного вхідного символу його номером в спеціальному стеці недавно використаних символів. Послідовності ідентичних символів, наприклад, будуть замінені (починаючи з другого символу) на послідовність однакових чисел (у моїй реалізації одиницею). Якщо ж символ довго не з'являвся у вхідній послідовності, він буде замінений великим числом. Перетворення замінює послідовність вхідних символів на послідовність цілих чисел, якщо у вхідних даних було багато локальних кореляцій, то серед цих чисел будуть переважати невеликі, краще стискувані ентропійним кодуванням, ніж вихідні дані [11].

## 1.2 Постановка задачі проектування

Для виконання курсової роботи слід програмно реалізувати декілька основних алгоритмів кодування даних та їх подальшого декодування. Це включає в себе розробку допоміжних функцій для роботи з файлами. Потрібно розробити користувацький інтерфейс для вибору файлу та вибору алгоритмів, якими він буде закодований. Важливою складовою роботи є порівняння різних алгоритмів, тому також буде корисним виведення програмою інформації щодо часу витраченого на виконання різних алгоритмів.

Підсумовуючи, програма повинна:

1. Давати можливість користувачу вибирати файли, які він бажає кодувати чи декодувати;
2. Давати можливість користувачу вибирати алгоритм який програма повинна використовувати для кодування;
3. У випадку декодування розпізнавати яким алгоритмом було закодовано файл і відновлювати його початкову версію;
4. Повідомляти користувача про час, витрачений на виконання алгоритму;
5. Давати можливість зберегти результуючий файл, для порівняння з початковим, щоб наочно впевнитися у ефективності алгоритмів.

## 1.3 Вибір середовища розробки

Важливою частиною розробки будь-якої програми є вибір середовища розробки, що може як допомагати при розробці так і перешкоджати їй.

Зазвичай цей вибір опирається на декілька основних факторів:

* Операційна система, на якій проходить розробка — в залежності від ОС деякі середовища розробки можуть бути менш зручними або просто не підтримуватися;
* Складність поставленої задачі — для менш складних задач використовувати повноцінну IDE може бути надлишковим, а для складніших багато файлових програм щось просте може виявитись просто недостатнім;
* Мова програмування — деякі середовища розробки краще підтримують певні мови програмування значно краще ніж інші як наприклад CLion, PyCharm чи IntelliJ.

Оскільки розробка проводилась на ОС Windows, на мові програмування C++, тому вибір був наступний:

* VSCode — крос-платформний редактор коду, який можливо налагодити для розробки на будь-якій мові програмування. Втім досить рідко використовується для розробки на C++, бо щоб вести розробку великих проектів на цій мові програмування доведеться докласти значних зусиль по налаштуванню середовища розробки;
* Code::Blocks — ще одне крос-платформне середовище для розробки. Зручне для написання програм на мові C++, але на жаль через свою простоту не завжди є кращим вибором для розробки складних програм;
* Dev-C++ — досить зручне середовище розробки спеціально під Windows, яке надає достатній функціонал для написання програм різного рівня;
* CLion — повноцінна багатофункціональна крос-платформна інтегрована середа для розробки на мові програмування C/C++. На жаль не є безкоштовною;
* Visual Studio — також є багатофункціональною інтегрованою середою для розробки, працюючою під Windows та macOS. Перевірена часом і широкою спільнотою розробників по всьому світу. Не є найзручнішою для малих програм, але для задачі курсової роботи підходить найкраще;
* Qt Creator — інтегрована середа розробки, для створення крос-платформних застосунків з використанням фреймворку Qt. Не є зручним для розробки класичних консольних програм, але значно полегшує задачу розробки графічного інтерфейсу.

Отже, у якості середовища для розробки основного коду було вибрано Visual Studio, а для розробки графічного інтерфейсу Qt Creator.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

## 2.1 Математична постановка задачі

Для кращого розуміння теорії стискання даних слід розуміти обмеження, які накладаються на будь-який алгоритм. Одним з таких обмежень є обмеження на оптимальне кодування символу, яке описане в теоремі про кодування джерела Шеннона.

## 2.2 Алгоритм розв’язку задачі

# РОДІЛ 3. ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Логіка та структура

## 3.2 Опис інтерфейсу

## 3.3 Результати роботи програмного продукту

# ВИСНОВОК

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Інтернет-енциклопедія «Вікіпедія» Стиснення даних

<https://uk.wikipedia.org/wiki/Стиснення_даних>

1. Інтернет-енциклопедія «Вікіпедія» Стиснення без втрат

<https://uk.wikipedia.org/wiki/Стиснення_без_втрат>

1. Інтернет-енциклопедія «Вікіпедія» Стиснення з втратами

<https://uk.wikipedia.org/wiki/Стиснення_з_втратами>

1. Інтернет-енциклопедія «Вікіпедія» Ентропійне кодування

<https://uk.wikipedia.org/wiki/Ентропійне_кодування>

1. Інтернет-енциклопедія «Вікіпедія» Метод стиснення з використанням словника

<https://ru.wikipedia.org/wiki/Метод_сжатия_с_использованием_словаря>

1. Інтернет-енциклопедія «Вікіпедія» RLE

<https://uk.wikipedia.org/wiki/RLE>

1. Інтернет-енциклопедія «Вікіпедія» Код Хаффмана

<https://uk.wikipedia.org/wiki/Код_Гаффмана>

1. Інтернет-енциклопедія «Вікіпедія» LZ77 і LZ78

<https://uk.wikipedia.org/wiki/LZ77_і_LZ78>

1. Інтернет-енциклопедія «Вікіпедія» Алгоритми LZ77 и LZ78

<https://neerc.ifmo.ru/wiki/index.php?title=Алгоритмы_LZ77_и_LZ78>

1. Інтернет-енциклопедія «Вікіпедія» Перетворення Берроуза-Вілера

<https://uk.wikipedia.org/wiki/Перетворення_Берроуза-Вілера>

1. Інтернет-енциклопедія «Вікіпедія» Move-To-Front

https://ru.wikipedia.org/wiki/Move-To-Front