Документация: Алгоритъм на Хъфман за компресия на данни

Изготвил: Николета Младенова, КН, курс 3, група 6, фн 81760

1. Увод:

1.1 Описание и идея на проекта

Идеята на проекта е да се реализира програма, която извършва компресия на данни без загуба чрез алгоритъм на Хъфман. Алгоритъмът се базира на простата идея, че най-често срещаните символи в даден низ трябва да се записват с възможно най-малък брой битове. Кодирането е обратимо – кодираната последователност може да се декомпресира т.е. да се възстановят оригиналните данни.

1.2 Цел и задачи на разработката

Да се реализират три режима на програмата:

- Компресия – прочита информация от файл, кодира я чрез алгоритъм на Хъфман и я записва в нов файл. При компресия информацията, нужна за декомпресия, се записва в отделен файл dataFile.txt. Пресмята се степента на компресия.
- Декомпресия – прочита кодирана информация и я декодира с помощта на информацията в dataFile.txt.
- Декомпресия Дебъг – алтернативен режим, при който вместо декомпресиране под формата на текстов файл, на стандартния изход се извеждат поредица от числа от 0 до 255, получени от разбиване на декомпресираната информация на блокове то по 8 бита.
- Програмата приема командни параметри, които указват в кой режим ще работи тя.

2. Преглед на предемната област

2.1 Основни дефиниции, концепции и алгоритми, които ще бъдат използвани.

Структурите от данни са структури, програмирани да съхраняват информация по такъв начин, че да можем ефективно да извършваме операции върху данните. Съществуват по-прости и по-сложни структури от данни. Например, integer, float, char, Boolean и др. са примитивни структури от данни. Но когато стане въпрос за обработка на голям обем от данни и по-сложни алгоритми се използват стекове, опашки, свързани списъци, дървета, графи и други.

С помощта на структурата от данни дърво, в тази програма се реализира ефикасно компресия на данни без загуба. Построява се двоично дърво на Хъфман, от което ще се определя азбуката за компресиране. Алгоритъмът за построяване на дървото се състои от следните стъпки:

1.Създава се честотна таблица на низа – за всеки символ се записва броя на срещанията му.

2.Нека различните символи в низа са n на брой. Създават се n дървета от по един възел, който съдържа наредена двойка: символ и число, означаващо броя на срещанията на символа в низа

3.Намират се двете дървета с най-малки числа в корените. Създава се ново дърво с двете намерени дървета като поддървета, а сумата от съответните им числа се записват в корена на новото дърво

4.Повтаря се стъпка 3, докато не се получи само едно дърво – дървото на Хъфман за дадения низ.

В резултат, построеното дърво е двоично и има точно n на брой листа, като всяко листо отговаря на един символ от честотната таблица.

След като е построено дървото се строи азбуката. На всяко ребро от дървото се съпоставя двоична цифра 0 (за ляво ребро) или 1 (за дясно ребро). Така на всеки път от корена до някое листо отговаря двоичен низ. Най-често срещаните символи са най-близо до корена и съответно на тях отговарят най-къси последователности - това съкращава големината на кодирания бинарен низ.

Построява се таблица, в която се записват бинарните кодове за всеки символ.

За да се декомпресират данните са нужни дървото и таблицата за кодиране, затова при компресия се записват в отделен файл, който се използва при декомпресия.

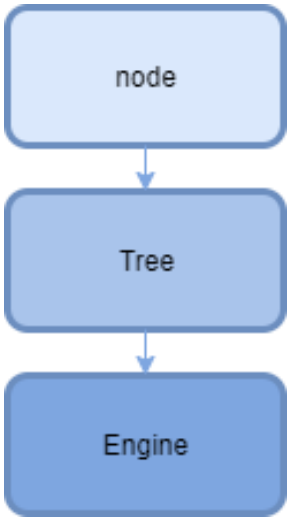
Декомпресията се реализира по следния начин: обхождат се едновременно двоичния низ и дървото, като всеки път, когато се срещне 0 в низа, се завива наляво, а при 1 – надясно. Когато се стигне до листо, намира се на кой символ от честотната таблица съответства пътят и се извежда, а обхождането на дървото се рестартира от корена. Така стъпка по стъпка, символ по символ, се получава оригиналния низ.

3. Проектиране

3.1 Обща архитектура

За проекта са реализирани структура node, клас Tree и клас Engine.

3.2 Диаграми (на структура и поведение)



Структура **node** – описва един връх на дърво. Има член данни data, left и right. Член данната data е от тип pair<char, int>. Ако върхът е листо, той съдържа двойка символ и число, което указва колко пъти се повтаря символа в низа, който ще компресираме. В противен случай съдържа интервал за pair.first и сбора на двете числа в корените на лявото и дясното поддърво.

Клас **Tree** – описва дървото. Има една член данна – node* root.

Клас **Engine** – описва компресията, декомпресията и дебъг декомпресията. Съдържа следните член данни:

string **info** – символният низ, който трябва да се компресира
vector <pair<char, int>> **frequency_table** – честотната таблица
Tree **huffman_tree** – дървото на Хъфман
vector<pair<char, string>> **coding_table** – таблицата на кодиране
string **binary_string** – кодирания низ
int **percentage** – колко процента е кодирана информация спрямо оригиналния обем.

Архитектурата на проекта се състои в това да се реализират структурата и двата класа. Структурата node е нужна за класа Tree. Класът Tree реализира дървото. В него са написани фунцкиите, свързани с обхождане на дървото. При компресия дървото се създава чрез честотната таблица и се кодира в отделен файл под формата на низ. При декомпресия се възстановява дървото чрез декодиране на низа. Поради тази причина класът Tree има няколко конструктора.

Класът Engine съдържа всички данни, които се използват за компресия и декомпресия, за да може да се проследи правилното им последователно построяване. За тази цел са и създадени print функции. В public секцията са методите compress, decompress и decompressdebug, които се извикват спрямо режима на програмата. Compress е дълъг алгоритъм, който построява дървото на Хъфман, декодира нужната информация за декомпресия в отделен файл и записва кодирания низ в output файла. Затова са създадени помощни фунцкии в private секцията, които инициализират нужните данни.

4. Реализация и тестове

4.1 Реализация на класовете

Клас Tree

- void copy(node*& self, node* other) – помощна функция, която се вика в конструктора и копи конструктура.
- void erase(node* self) – помощна функция, която се вика в деструктора.

- void makeCodingTableHelper(node* root, string code, vector<pair<char, string>>& coding_table) – помощан фунцкия.
- int findIndex(string str, int start, int end) – помощна фунцкия.
- node* fromStringToTreeHelper(string tree_string, int start, int end) – помощна фунцкия
- Голяма четворка и оператор =.
- Tree(vector<pair<char,int>> frequency_table) – конструктор, който построява дърво при дадена честотна таблица по време на компресия.
- Tree(string tree_string, int start, in end) – конструктор по време на декомпресия, който възстановява дървото от кодирания низ, записан в отделен файл при компресия.
- getters.
- vector<pair<char,string>> makeCodingTable() – функция, която обхожда дървото и построява таблица на кодиране.
- void writeTree(node* t, ofstream &dataFile) – фунцкия, която кодира дървото в низ и го записва в отделен файл по време на компресия.
- string getSymbol(vector<pair<string, string>> v, string s) – помощна функция за функция result.
- string result(string binary_string, vector<pair<string, string>> v) – финална функция, която обхождайки дървото възстановява кодиранта информация.
- void print(node* root) – принтира дървото.

Клас Engine

Помощни функции за метода compress:

- void readfromfile(string filename) – прочита символния низ от зададения input файл.
- void makeFrequencyTable() – създава честотната таблица на низа.
- void makeCodingTable(Tree t) – създава кодиращата таблица.
- string getCharCode(char c) – помощна фунцкия, която по зададен символ връща неговият бинарен код.
- void makeBinaryString() – построява кодирания бинарен низ.

- void writetofile(string filename) – записва кодирания бинарен низ в зададения output файл.
- void writeTable(ofstream &dataFile) – кодира честотната таблица в отделен файл за последваща декомпресия.
- void calculatePercentage() – пресмята колко процента е обемът на кодирания низ спрямо оригиналният низ.

Конструктора по подразбиране – докато при компресията даваме стойности на всички член-данни, при декомпресията са ни нужни само дървото и таблицата на кодиране. Поради тази причина, когато се пуска програмата се създава един обект, вика му се конструктора по подразбиране и след това се вика метода compress/decompress/decompressDebug в зависимост от това дали работим в режим на компресия/декомпресия/декомпресия дебъг. Така в режимите компресия и декомпресия можем да видим данните, нужни за реализацията на съответния алгоритъм чрез фунцкиите print():

- void printInfo()
- void printFrequencyTable()
- void printTree()
- void printCodingTable()
- void printBinaryString()
- void printPercentage() - в режим компресия на стандартния изход се извежда степента на компресия.

4.2 Планиране, описание и създаване на тестови сценарии

Програмата е реализирана да се указва чрез командни параметри в кой режим ще работи тя. В main се създават три флага: bool compress, bool decompress, bool decompressDebug, инициализирани с false. Ако се въведе параметър –c[ompress] флагът compress се вдига и се изпълнява компресия. Аналогично за –d[ecompress] и –dd/-ddecompressdebug. Декларирани са string-овете inputFile и outputFile. Параметър –i указва, че следва името на input файла, което се записва в стринга inputFile. Параметър –o указва, че следва името на output файла, което се записва в стринга outputFile.

Тестваме програмата със символния низ ABRACADABRA:

В input.txt записваме ABRACADABRA.

Компилираме програмата и я изпълняваме с командни параметри:

-c –i input.txt –o compression.txt

Флагът компресия се вдига, създава се обект tocompress и се изпълнява методът compress. Той прочита низът в стринга info. С print функциите можем да проследим алгоритъма. **Построява се честотната таблица** на низа:

A, 5

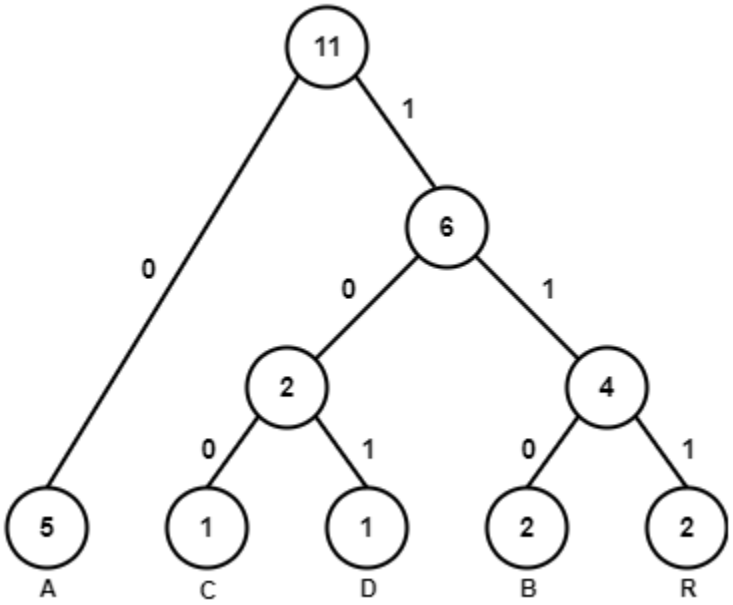
B, 2

R, 2

C, 1

D, 1

Построява се дървото:



и честотната таблица:

A, 0

C, 100

D, 101

B, 110

R, 111

В dataFile.txt се записват дървото и таблицата на кодиране, нужни за декомпресия:

(*(A)(*(*(C)(D))(*(B)(R))))

A0C100D101B110R111

Построява се бинарния низ:

ABRACADABRA -> A B R A C A D A B R A ->

0 110 111 0 100 0 101 0 110 111 0 -> 01101110100010101101110

И се записва във файл compression.txt

Пресмята се **степента на компресията** и се извежда на стандарния изход:

ABRACADABRA – 88 бита

01101110100010101101110 – 23 бита

Процент компресирана информация: 26%

Изпълняваме програмата с командни параметри:

-d –i compression.txt –o decompression.txt

Флагът за декомпресия се вдига, създава се обект todecompress и се вика методът decompress.

От файл compression.txt се прочита кодираната информация.

От файл dataFile.txt се прочитат кодираните дърво и честотна таблица и се разкодират.

Чрез обхождане едновременно на дървото и на бинарния низ се възвръща кодираната информация. Обхождаме низа и ако сме на 0, отиваме в лявото поддърво, ако сме на 1, завиваме в дясното поддърво. Когато стигнем листо, намираме на кой символ в таблицата на кодиране отговаря пътят.

01101110100010101101110 -> 0 110 111 0 100 0 101 0 110 111 0 -> ABRACADABRA

Декодираната информация се записва във файл decompression.txt

Изпълняваме програмата с командни параметри:

-dd –i compression.txt

Вдига се флагът за режим декомпресия дебъг. Бинарният низ се прочита от файл compression.txt. Разделя се на блокове по 8 бита и всеки блок се преобразува от бинарно в десетично число. Резултат се изписва на стандартния изход.

01101110100010101101110 -> 01101110 10001010 1101110 -> 110 138 110

Забележка: В условието на проекта, показанато дърво на Хъфман за символния низ ABRACADABRA е малко по-различно, защото при построяването на дървото се намират двете дървета с най-малка стойност в корена и от тях се построява ново дърво, а има момент, в който има 3 дървета със стойности в корена 2 – тривиално дърво B, тривиално дърво R и дърво с листа C и D. Реализираният алгоритъм първо взима дърветата B и R и прави от тях ново дърво, а при показаното дърво в условието първо се взимат дърво B и дърво с листа C и D. В и двата случая кодираната информация е 23 бита.

5. Заключение

5.1 Обобщение на изпълнението на началните цели

Целта програмата да работи в три режима: режим на компресия, режим на декомпресия и режим на декомпресия дебъг, бе изпълнена. Програмата успешно кодирата и разкодира символен низ, както и пресмята степента на компресия – колко процента е обемът на кодираната информация от оригиналната.

5.2 Насоки за бъдещо развитие и усъвършенстване

С навлизането на компютрите в живота на хората се появява възможността за бързо и ефективно превеждане на поток от данни в по-икономичо представяне. Създават се алгоритми за компресиране на информация, които с времето се усъвършенстват. Едни от най-познатите за нас програми за компресиране са ZIP и RAR. Средната степен на компресия

(колко процента е обемът на кодиранта информация от оригиналната) на ZIP е 38%. Средната степен на компресия на RAR е 30%. Разбира се това са програми, които работят с най-различни файлове – мултимедия (звук), картина (GIF, JPEG, PNG), филмов клип (MPEG, WEBM) и други. Реализираната програма средно компресира символни низове с 20-40% обем от оригинала.