Документация

Тодор Георгиев Шишманов 45603

1. Създаване на честотна таблица
2. Създаване на Хъфманово дърво
3. Кодиране и декодиране
4. Примери

1. Създаване на честотна таблица (FreqTable)

class FreqTable

{

private:

std::vector<int> table\_freq;

std::vector<int> table\_letters;

int size;

Честотната таблица има два вектора, в които се складират честотите и самите символи, в ASCII вариант, както и размерност. Класът притежава член-функции с функционалност отговаряща на имената им :

void print();

int get\_size();

int get\_letter(int place\_in\_table);

int get\_freq(int place\_in\_table);

Базовият конструктур на класа се използва само за наследяването по-късно в кода. Класът притежава и друг конструктур:

FreqTable::FreqTable(std::vector<short> input) {

const int n = input.size();

bool alreadyIn = 0;

for (int i = 0; i < n; i++) {

for (size\_t j = 0; j < table\_letters.size(); j++) {

if (input[i] == table\_letters[j]) {

table\_freq[j]++;

alreadyIn = 1;

}

}

if (!alreadyIn) {

table\_letters.push\_back(input[i]);

table\_freq.push\_back(1);

}

alreadyIn = 0;

}

size = table\_freq.size();

}

Той приема вектор от числа, от които с ASCII таблица могат да бъдат получени символи, намира използваните числа и изчислява честотата на срещането им, след което ги вкарва във векторите table\_letters(числата отговарящи на символи) и table\_freq(честотата на символите). Алгоритъмът работи по следния начин: Минава по всички символи, ако даден символ не е срещан го добавя в table\_letters и поставя честота 1 в table\_freq, ако дадени символ е срещам, то само увеличава честотата му.

Край на стъпка 1.

2. Създаване на Хъфманово дърво

Стъпки за създаване на дървото:

1. Създаваме клас Tree, притежаващ честота и сумвол и за всеки елемент в честотната таблица създаваме дърво
2. Намиране двете най-малки по честота дървета
3. Създаваме ново дърво, което има честота равна на сбора на честотите на двете дървета от стъпка 2 и записваме несъществуваща цифра в ASCII таблицата – 256
4. Повтаряме стъпки 2 и 3 докато не остане само едно дърво, тогава то е дървото на Хъфман.

1-ва стъпка: Клас Tree

class Tree

{

private:

int freq;

int letter;

Tree\* left;

Tree\* right;

Всяко дърво има честота и символ, също така и pointer-и, които сочат към двете дървета, които то ще убедини в стъпка 3. Ако няма такива, то то е листо със легитимен символ (между 0 и 255). Всички останали функции са себеописващи се и прости, заради това класа Tree няма .cpp файл, а само header.

Създаваме и клас Algorithm, където ще се извършват всички действия от следващите стъпки. Клас Algorithm:

class Algorithm :

public Tree,

public FreqTable

{

private:

std::priority\_queue<Tree\*, std::vector<Tree\*>, compare\_trees> all\_trees;

std::vector<short> letters;

std::vector<std::string> coded\_letters;

Този клас наследява честотните таблици и дърветата, защото ще трябва да създаде дървета за всеки елемент от честотната таблица. В coded\_letters ще вкарваме кодировката на всеки символ, а в letters самите символи в ASCII код. Създаваме и опашка, с идеята, че в този вектор ще вкараме всички дървета, но ще ни трябват само тези две от тях с най-малка честота. Създаваме и структурата compate\_trees като начин за подредба. Тя е дефинирата така:

//same as std::greater<T>, but for Tree. Needed in minHeap

struct compare\_trees {

bool operator()(Tree\* first, Tree\* second) {

return(first->get\_freq() > second->get\_freq());

}

};

В конструктора се създават всички първоначални дървета:

Algorithm::Algorithm(FreqTable table) {

int table\_size = table.get\_size();

Tree\*left;

Tree\* right;

Tree\* new\_tree;

for (int i = 0; i < table\_size; i++) {

all\_trees.push(new Tree(table.get\_freq(i), table.get\_letter(i)));

}

while (all\_trees.size() != 1) {

left = all\_trees.top();

all\_trees.pop();

right = all\_trees.top();

all\_trees.pop();

new\_tree = new Tree(left->get\_freq() + right->get\_freq(), 256);

new\_tree->set\_left\_to(left);

new\_tree->set\_right\_to(right);

all\_trees.push(new\_tree); //because i defined it as min heap it will be sorted when pushed

}

}

Алгоритъмтъ е съвсем прост, имаме по един пойнтер за двата минимални елемента и един за новото дърво. Първоначално създаваме всички стандартни дървета, после докато не остане само едно дърво задаваме пойнтерите left и right и изкарваме двете най-малки дървета, после създаваме новото дърво и го вкарваме във вектора all\_trees. Защото е priority\_queue се самосортира след вкарването на новото дърво. Щом на всяка итерация вкарваме едно дърво и махаме две, все някога бройката ще стане 1-ца. Така в края на конструктура на Algorithm имаме Хъфмановото дърво, което може да ползваме за кодиране или декодиране.

3. Кодиране и декодиране

Кодирането се случва с рекурсивна функция, в която се приема пойнтер към началото на Хъфмановото дърво и string, в който ще запишем кодировката.

if (!root)

return;

if (root->get\_letter() != 256) {

std::cout << (char)root->get\_letter() << ": " << code << std::endl;

letters.push\_back(root->get\_letter());

coded\_letters.push\_back(code);

}

Algorithm\_code(root->left\_child(), code + '0');

Algorithm\_code(root->right\_child(), code + '1');

С всеки ляв клон вписваме 0 и с всеки десен - 1. Целият код е записан в coded\_letters, а срещу всеки код – символа. Принтирането се извършва с друга функция. Принтира се първо оригиналното съобщение, после кодираното.

void Algorithm::print\_the\_code(std::vector<short> original) {

int org\_size = original.size();

int letters\_size = letters.size();

std::cout << "Original Message:" << std::endl;

for (int i = 0; i < org\_size; i++) {

std::cout << (char)original[i];

}

std::cout << std::endl << "Coded Message:" << std::endl;

for (int i = 0; i < org\_size; i++) {

for (int j = 0; j < letters\_size; j++) {

if (original[i] == letters[j]) std::cout << coded\_letters[j]; //all the couts are for documentation purposes

}

}

}

Декодирането става като се ходи по кодираното съобщение и с всяка 0 или 1 се отива вляво или вдясно от сегашното дърво, докато не се стигне такова със символ в интервала между 0 и 255.

void Algorithm::Algorithm\_decode(Tree \* root, std::string coded\_message) {

int size = coded\_message.size();

Tree\* iterator = root;

for (int i = 0; i < size; i++) {

if (coded\_message[i] == '0')

iterator = iterator->left\_child();

else

iterator = iterator->right\_child();

if (iterator->get\_letter() != 256)

{

std::cout << (char)iterator->get\_letter();

iterator = root;

}

}

}

4. Примери

1. Честотните таблици:

//testing Tabels

vector<short> test = { 1,2,3,3,3,3,3,3,3,3,3,4,5,5,5,6,7,2,3,1,3 };

FreqTable test\_table(test);

test\_table.print();

Изход на конзолата:

Letter: Frequency:

1 2

2 2

3 11

4 1

5 3

6 1

7 1

Press any key to continue . . .

1. Създаване на кодировка:

//testing Algorithm

Algorithm testing(test\_table);

//see the table of coding

string code="";

testing.Algorithm\_code(testing.top\_element(), code);

Изход на конзолата:

1: 000

2: 001

5: 010

7: 0110

4: 01110

6: 01111

3: 1

Press any key to continue . . .

1. Кодировка:

//testing coding

testing.print\_the\_code(test);

cout << endl;

Изход на конзолата:

Original Message:

123333333334555672313

Coded Message:

000 001 1 1 1 1 1 1 1 1 1 01110 010 010 010 01111 0110 001 1 000 1

Поставени са шпации, за да е по-лесно за проверка

1. Декодировка:

//testing decode

string decode\_testing;

cin >> decode\_testing;

testing.Algorithm\_decode(testing.top\_element(),decode\_testing);

В cin вкарваме 0000101

Изход на конзолата:

153Press any key to continue . . .