Algorytmy zachłanne

Sprawozdanie z laboratorium 11 – Łukasz Konieczny, LK4

## 1. Wstęp teoretyczny

### 1.1 Algorytmy zachłanne

Algorytmy zachłanne, w przeciwieństwie do dynamicznych, w każdym kroku podejmują decyzję która w danym kroku, a nie całym przejściu algorytmu (lokalnie, a nie globalnie) wydaje się najkorzystniejsza.

Różnicę tą dobrze ilustruje problem plecakowy: algorytm dynamiczny będzie korzystał ze wszystkich możliwych rozwiązań dla wszystkich możliwych pojemności plecaka i ułożeń przedmiotów aby wyznaczyć rozwiązanie optymalne, natomiast algorytm zachłanny w każdym kroku będzie umieszczał w plecaku najbardziej wartościowy przedmiot, co może doprowadzić do wyznaczenia rozwiązania suboptymalnego

Przykłady zastosowania algorytmów dynamicznych:

* Problem plecakowy
* Problem wydawania reszty
* Algorytm Dijkstry
* Algorytm Kruskala

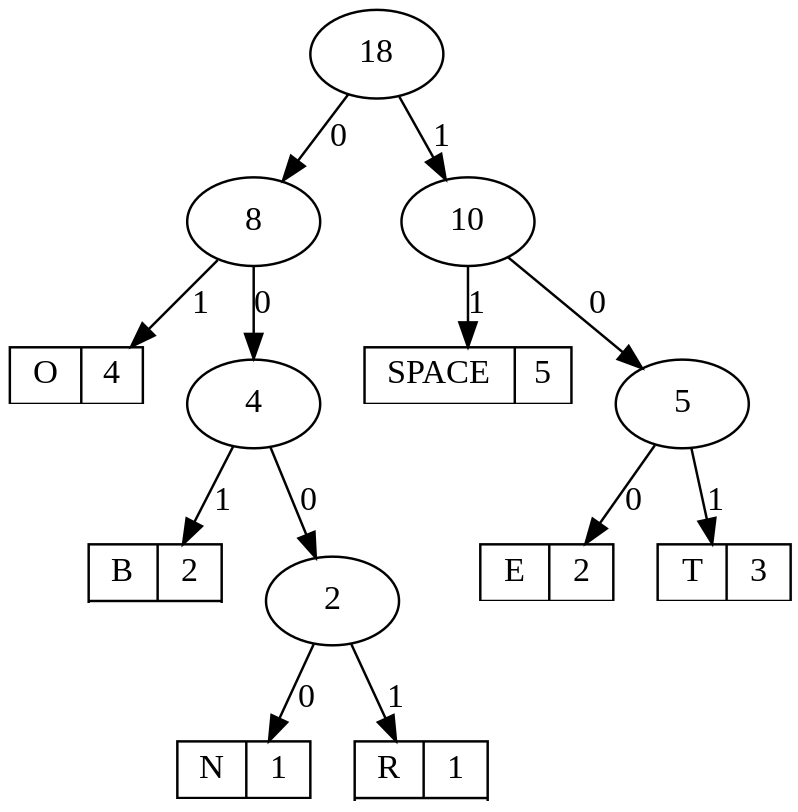
### 1.2 Kodowanie Huffmana

Kodowanie Huffmana to metoda prostej kompresji bezstratnej. Wykorzystuje drzewo binarne do wyznaczenia kodów odpowiadających danemu znakowi / liczbie / innemu typowi danych na bazie częstotliwości jego występowania w oryginalnym pliku / zbiorze. Jest to algorytm niedeterministyczny, ponieważ daje nam dowolność wyboru drzew o takim samym prawdopodobieństwie, ani nie określa które z usuwanych drzew staje się lewym, a które prawym poddrzewem.

Algorytm tworzenia drzewa Huffmana, na bazie którego wyznaczane są kody, wygląda następująco:

* Wybieramy ze zbioru dwa elementy o najmniejszej częstotliwości
* Tworzymy nowy węzeł, który nie posiada danych, a jego częstotliwość jest równa sumie częstotliwości wybranych elementów
* Utworzony węzeł staje się rodzicem wybranych elementów
* Wybieramy kolejne dwa elementy o najmniejszych częstotliwościach (przy wyborze uwzględniamy utworzony wcześniej węzeł!)
* Powtarzamy poprzednie kroki aż do wyczerpania elementów zbioru
* Lewym krawędziom drzewa przypisujemy „wagę” zero, a prawym jeden

Gotowe drzewo Huffmana, utworzone w oparciu o zdanie „TO BE OR NOT TO BE”, prezentuje się następująco:



Wyznaczenie kodu dla poszczególnej litery jest bardzo proste, jest nim ciąg „wag” krawędzi, przy pomocy których możemy dostać się do węzła zawierającego daną literę.

Przykładowo, powyższe drzewo wyznacza dla litery „B” kod „001”, a dla litery „T” „101”

## 2. Implementacja w C++

W programie warto skorzystać z dwóch struktur danych: kolejki priorytetowej oraz mapy.

* Kolejka priorytetowa automatycznie ustawi węzły o najmniejszej częstotliwości na swoim początku, dzięki czemu nie będziemy musieli martwić się ciągłym sortowaniem listy węzłów
* Mapa posłuży do zliczania wystąpień danej litery w tekście, na bazie którego utworzymy kod Huffmana, a także do zapisania zależności między literą, a odpowiadającym jej kodem

Aby ułatwić sobie zarządzanie danymi, korzystać będziemy z następującej struktury, definiującej węzeł drzewa Huffmana:

struct Node {

    char value;

    int freq;

    Node \*left, \*right;

    Node(char value, int freq, Node\* left = nullptr, Node\* right = nullptr){

        this->value = value;

        this->freq = freq;

        this->left = left;

        this->right = right;

    }

};

Aby móc wykorzystać kolejkę priorytetową, musimy napisać strukturę przeładowującą operator, przy pomocy którego kolejka dokonuje wyznaczenia priorytetu:

struct Comparator {

    bool operator()(Node\* l, Node\* r) {

        return l->freq > r->freq;

    }

};

Główna funkcja programu prezentuje się następująco:

void huffman(string text){

    // Mapa zliczajaca czestotliwosc wystapienia znaku w tekscie

    unordered\_map<char, int> freq;

    for (char c: text) {

        freq[c]++;

    }

    // Kolejka przechowujaca wezly drzewa Huffmana

    priority\_queue<Node\*, vector<Node\*>, Comparator> pq;

    // Dodajemy do kolejki kazda litere wystepujaca w tekscie wraz z odpowiadajaca czestotliwoscia

    // Kolejka priorytetyzuje litery o jak najnizszej czestotliwosci

    for (auto pair: freq) {

        pq.push(new Node(pair.first, pair.second));

    }

    int sum;

    // Dopoki w kolejce nie pozostal tylko ostatni wezel bez wartosci:

    while (pq.size() != 1) {

        // Wyciagamy z kolejki dwa wezly o najnizszej czestotliwosci i usuwamy je z niej

        Node \*left = pq.top(); pq.pop();

        Node \*right = pq.top(); pq.pop();

        // Dodaje do kolejki wezel bez wartosci, o czestotliwosci rownej sumie wybranych wezlow

        sum = left->freq + right->freq;

        pq.push(new Node('\0', sum, left, right));

    }

    // Na poczatku kolejki znajduje sie wezel bez wartosci o sumie wszystkich czestotliwosci liter

    Node\* root = pq.top();

    // Tworzymy mape przechowujaca kody, oraz wypelniamy ja funkcja "generateCodes"

    unordered\_map<char, string> huffmanCode;

    generateCodes(root, "", huffmanCode);

    menu(huffmanCode, root);

}

Funkcja wyznaczająca kody dla poszczególnych liter:

void generateCodes(Node\* root, string str, unordered\_map<char, string>& huffmanCode) {

    if (root == nullptr)

        return;

    // Wezel ktory nie ma dzieci jest wezlem z danymi. Zapisujemy w mapie wartosc wyznaczonego dla niego kodu

    if (!root->left && !root->right) {

        huffmanCode[root->value] = str;

    }

    // Rekurencyjnie sprawdzamy dzieci wezla. Jesli ich nie ma, funkcja przerwie sie dla nich na jej poczatku

    generateCodes(root->left, str + "0", huffmanCode); // Zakladamy, ze lewe krawedzie maja wage "0" a prawe "1"

    generateCodes(root->right, str + "1", huffmanCode);

}

Funkcja dekodująca kod Huffmana na literę, oraz wypisująca ją:

void decode(Node\* root, int& index, string str) {

    if (root == nullptr) {

        return;

    }

    // Jesli dotarlismy do wezla ktory nie ma dzieci, wypisujemy przypisana mu liczbe

    if (!root->left && !root->right) {

        cout << root->value;

        return;

    }

    // Jesli nie, przechodzimy o jeden znak dalej w napisie

    index++;

    // A nastepnie powtarzamy rekurencyjnie wyszukiwanie wezla "bezdzietnego"

    if (str[index] == '0')

        decode(root->left, index, str);

    else

        decode(root->right, index, str);

}

### 3. Wnioski

Podczas zajęć zapoznaliśmy się z definicją algorytmów zachłannych, porównaliśmy je z podejściem dynamicznym, a także nauczyliśmy się wykorzystywać kodowanie Huffmana do bezstratnej kompresji danych.

## Bibliografia

<https://pl.wikipedia.org/wiki/Kodowanie_Huffmana>

<https://gist.github.com/pwxcoo/72d7d3c5c3698371c21e486722f9b34b>

<https://www.cs.put.poznan.pl/mszachniuk/mszachniuk_files/lab_aisd/Szachniuk-ASD-t5.pdf>

<https://www.youtube.com/watch?v=YcvBS0epTGo&pp=ygUSa29kb3dhbmllIGh1ZmZtYW5h>

<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>

<https://www.geeksforgeeks.org/huffman-coding-using-priority-queue/>

Materiały z wykładów – Zbigniew Kokosiński