

Algorytmy zachłanne

Sprawozdanie z laboratorium 11 – Piotr Sarna LK1

Cel ćwiczenia

Podczas zajęć zapoznaliśmy się ze sposobem rozwiązywania problemów wykorzystując algorytmy zachłanne. Następnie wykorzystaliśmy je do kompresji ciągu znaków za pomocą Kodowania Huffmana.

Wstęp teoretyczny

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

Zalety algorytmów zachłannych:

- Szybkość i prostota
- Efektywność
- Brak potrzeby pamięci o wcześniejszych decyzjach

Wady algorytmów zachłannych:

- Brak gwarancji optymalności – ponieważ algorytm podejmuje decyzje na podstawie lokalnych informacji, może łatwo „utknąć” w suuboptymalnym rozwiązaniu.
- Zależność od struktury problemu – W problemach, gdzie lokalne decyzje mogą prowadzić do złych globalnych wyników, heurystyki zachłanne nie są skuteczne.

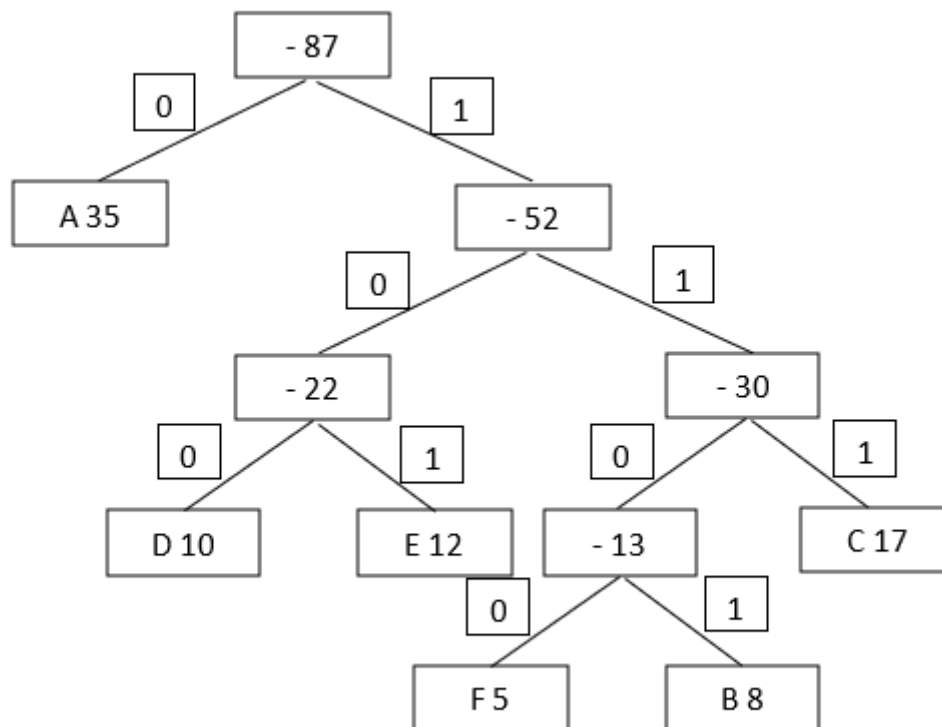
Opis algorytmu

Kodowanie Huffmana to metoda prostej kompresji bezstratnej. Wykorzystuje drzewo binarne do wyznaczenia kodów odpowiadających danemu znakowi na bazie częstotliwości jego występowania w oryginalnym 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 uprzednio 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 binarne prezentuje się następująco:



Źródło: Materiały z wykładów nt. algorytmów zachłannych

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 „1101”, a dla litery D „100”.

Implementacja rozwiązania problemu Kodowania Huffmana za w C++.

```
struct Node {
    char value;
    int frequency;
    Node* leftChild;
    Node* rightChild;

    Node(char val, int freq) : value(val), frequency(freq), leftChild(nullptr), rightChild(nullptr) {}
};
```

Struktura Node, reprezentuje pojedynczy znak. Zawiera ona jego wartość, częstotliwość jego występowania oraz adresy w pamięci dla lewej i prawej gałęzi.

```
void sortNodeVector(std::vector<Node*>& nodeVector) {
    for (size_t i = 0; i < nodeVector.size(); i++) {
        for (size_t j = i + 1; j < nodeVector.size(); j++) {
            if (nodeVector[j]->frequency < nodeVector[i]->frequency) {
                std::swap(nodeVector[i], nodeVector[j]);
            }
        }
    }
}
```

Funkcja sortNodeVector sortuje wektor zawierający struktury Node rosnąco, na podstawie częstotliwości ich występowania.

```
void huffmanCoding(std::vector<Node*>& nodeVector) {
    while (nodeVector.size() > 1) {
        sortNodeVector(nodeVector);

        Node* left = nodeVector[0];
        Node* right = nodeVector[1];

        Node* parent = new Node('\0', left->frequency + right->frequency);
        parent->leftChild = left;
        parent->rightChild = right;

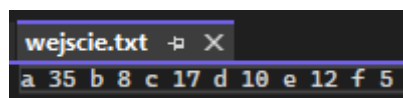
        nodeVector.erase(nodeVector.begin(), nodeVector.begin() + 2);
        nodeVector.push_back(parent);
    }
}
```

Funkcja huffmanCoding wybiera dwa znaki, które występują najrzadziej, dzięki przedstawionej wyżej funkcji sortującej.

Następnie tworzy nowy węzeł bez wartości, o częstotliwości równej sumie częstotliwości obu wybranych uprzednio znaków. Są one następnie „podpinane” do nowo utworzonego węzła oraz usuwane z wektora z danymi. Nowy węzeł jest dodawany do wektora z danymi. Pętla wykonuje się tak długo, aż w tablicy danych pozostanie tylko jeden element – będzie to korzeń drzewa binarnego, o największej częstotliwości występowania.

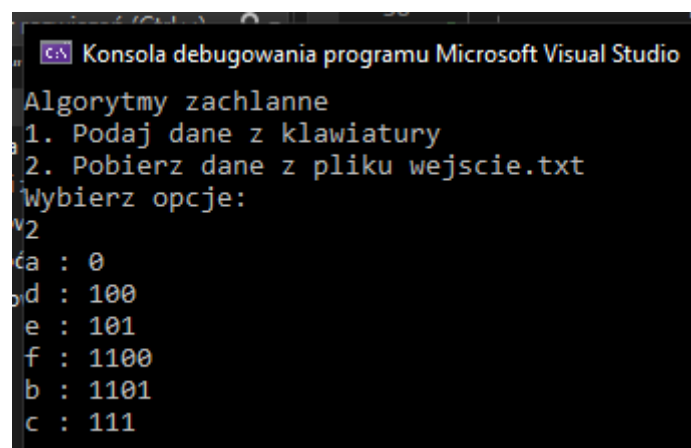
Prezentacja działania mojej implementacji w C++

Dla następujących danych:



```
wejście.txt
a 35 b 8 c 17 d 10 e 12 f 5
```

Otrzymujemy następujące wyniki:



```
Konsola debugowania programu Microsoft Visual Studio
Algorytmy zachłanne
1. Podaj dane z klawiatury
2. Pobierz dane z pliku wejście.txt
Wybierz opcje:
v2
ca : 0
d : 100
e : 101
f : 1100
b : 1101
c : 111
```

Wnioski

Algorytm cechuje złożoność czasowa $O(n^3)$. Dzieje się tak, ponieważ do posortowania tablicy wykorzystuję sortowanie bąbelkowe o złożoności $O(n^2)$. Główna pętla przechodzi jeden raz po tablicy – złożoność $O(n)$. Iloczyn $O(n) \times O(n^2) = O(n^3)$.

Bibliografia

<https://esezam.okno.pw.edu.pl/mod/book/view.php?id=2243&chapterid=4410>

https://home.agh.edu.pl/~turcza/ts/1_Huffman.pdf

<http://www.algorytm.org/algorytmy-kompresji/kody-huffmana/huffman-c.html>