

Akademia Nauk Stosowanych - Teoretyczne i technologiczne podstawy multimediiów - laboratorium			
Temat: Algorytm Huffmana.			Symbol: TiTPM
Nazwisko i imię: Fyda Kamil		Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 18.10.2022r.	Grupa: L1		

1. Opis teoretyczny:

Kodowanie Huffmana (ang. Huffman coding) – jedna z najprostszych i łatwych w implementacji metod kompresji bezstratnej. Została opracowana w 1952 roku przez Amerykanina Davida Huffmana. Algorytm Huffmana nie należy do najefektywniejszych obliczeniowo systemów bezstratnej kompresji danych, dlatego też praktycznie nie używa się go samodzielnie. Często wykorzystuje się go jako ostatni etap w różnych systemach kompresji, zarówno bezstratnej, jak i stratnej, np. MP3 lub JPEG. Pomimo że nie jest doskonały, stosuje się go ze względu na prostotę oraz brak ograniczeń patentowych. Jest to przykład wykorzystania algorytmu zachłannego.

2. Praktyczne zastosowanie algorytmu Huffmana:

Jednym z głównych problemów stosowania statycznego algorytmu Huffmana jest konieczność transmisji całego drzewa lub całej tablicy prawdopodobieństw. W przypadku transmisji drzewa węzły są odwiedzane w porządku *preorder*, węzeł wewnętrzny może zostać zapisany na jednym bicie (ma zawsze dwóch synów), liście natomiast wymagają jednego bitu plus takiej liczby bitów, jaka jest potrzebna do zapamiętania symbolu (np. 8 bitów). Np. drzewo z przykładu powyżej może zostać zapisane jako: (1, 0, 'D', 1, 0, 'C', 1, 0, 'B', 0, 'A'), czyli $7 + 4 \cdot 8 = 39$ bitów.

Lepszą kompresję, kosztem jednak bardzo szybkiego wzrostu wymagań pamięciowych, uzyskuje się, kodując kilka kolejnych znaków naraz, nawet jeżeli nie są one skorelowane.

3. Kod programu:

```
#include <iostream>
#include <string>
#include <queue>
#include <unordered_map>
using namespace std;

// węzeł drzewa
struct wezel
{
    char ch;
    int czestotliwosc;
    wezel* lewy, * prawy;
};

// nowe drzewo
wezel* pobierzWezel(char ch, int czestotliwosc, wezel* lewy, wezel* prawy)
{
    wezel* node = new wezel();

    node->ch = ch;
    node->czestotliwosc = czestotliwosc;
    node->lewy = lewy;
    node->prawy = prawy;

    return node;
}

// porównanie
struct comp
{
    bool operator()(wezel* l, wezel* r)
    {
        return l->czestotliwosc > r->czestotliwosc;
    }
};

void koduj(wezel* root, string str,
           unordered_map<char, string>& kodHuffmana)
{
    if (root == nullptr)
        return;

    // liść
    if (!root->lewy && !root->prawy) {
        kodHuffmana[root->ch] = str;
    }

    koduj(root->lewy, str + "0", kodHuffmana);
    koduj(root->prawy, str + "1", kodHuffmana);
}

// odkodowanie
void decode(wezel* root, int& index, string str)
{
    if (root == nullptr) {
        return;
    }

    // liść
    if (!root->lewy && !root->prawy)
    {
        cout << root->ch;
        return;
    }

    index++;

    if (str[index] == '0')
        decode(root->lewy, index, str);
    else
        decode(root->prawy, index, str);
}

// Budowanie drzewa i dekodowanie wejścia
void budujDrzewo(string text)
{
    unordered_map<char, int> czestotliwosc;
    for (char ch : text) {
        czestotliwosc[ch]++;
    }

    priority_queue<wezel*, vector<wezel*>, comp> pq;

    for (auto pair : czestotliwosc) {
        pq.push(pobierzWezel(pair.first, pair.second, nullptr, nullptr));
    }

    while (pq.size() != 1)
    {
        // Usuwamy dwa węzły z najwyższym priorytetem - najniższą częstotliwością z kolejki
        wezel* lewy = pq.top(); pq.pop();
        wezel* prawy = pq.top(); pq.pop();

        // Tworzymy nowy węzeł wewnętrzny
        int sum = lewy->czestotliwosc + prawy->czestotliwosc;
        pq.push(pobierzWezel('\0', sum, lewy, prawy));
    }
}
```

```

wezel* root = pq.top();

unordered_map<char, string> kodHuffmana;
koduuj(root, "", kodHuffmana);

cout << "\nKod Huffmana ma postac: \n" << '\n';
for (auto pair : kodHuffmana) {
    cout << pair.first << " " << pair.second << '\n';
}

cout << "\nOryginalny tekst: \n" << text << '\n' << endl;

// zakodowany tekst
string str = "";
for (char ch : text) {
    str += kodHuffmana[ch];
}

cout << "\nZakodowany tekst: \n" << str << '\n';

// dekodujemy wcześniej zakodowany kod
int index = -1;
cout << "\nOdkodowany tekst: \n";
while (index < (int)str.size() - 2) {
    decode(root, index, str);
}

main()

string tekst;
cout << "Wpisz tekst, który chcesz zakodować: " << endl;
cin >> tekst;


string text = tekst;

budujDrzewo(text);
int i = 0, alphabet[26] = { 0 }, j;
while (text[i] != '\0') {
    if (text[i] >= 'a' && text[i] <= 'z') {
        j = text[i] - 'a';
        ++alphabet[j];
    }
    ++i;
}
cout << "\nLiczba znaków występujących w kodzie:" << endl;
for (i = 0; i < 26; i++)
    cout << char(i + 'a') << " : " << alphabet[i] << endl;

return 0;
}

```

4. Wynik działania programu:

 Konsola debugowania programu Microsoft Visual Studio

```
Wpisz tekst, który chcesz zakodować:  
aaabbbcccd
```

```
Kod Huffmana ma postać:
```

```
a 00  
c 01  
b 10  
d 11
```

```
Oryginalny tekst:  
aaabbbcccd
```

```
Zakodowany tekst:  
000000101010010101111111
```

```
Odkodowany tekst:  
aaabbbcccd
```

```
Liczba znaków występujących w kodzie:  
a : 3  
b : 3  
c : 3  
d : 3
```