Akademia Nauk Stosowanych w Nowym Sączu Teoretyczne i technologiczne podstawy multimediów - laboratorium			
Temat: Kodowanie Huffmana			**Symbol: TiTPM_L2**
Nazwisko i imię:		**Ocena sprawozdania**	**Zaliczenie:**
1. Szczepanek Piotr			
Data wykonania ćwiczenia: 11.10.2022	**Grupa: L2**		

1. Kod programu

kodowanie_Huffmana.cpp

kodowanie_Huffmana (Globalny zasięg)

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

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

//alokacja nowego drzewa
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 obiektów
struct comp
{
    bool operator()(wezel* l, wezel* r)
    {
        // element o najwyższym priorytecie ma najniższą częstotliwość
        return l->czestotliwosc > r->czestotliwosc;
    }
};
// przemierzamy drzewo
void koduj(wezel* root, string str,
    unordered_map<char, string>& kodHuffmana)
{
    if (root == nullptr)
        return;

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

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

kodowanie_Huffmana.cpp

kodowanie_Huffmana (Globalny zasięg)

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

    // węzeł liścia
    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);
}

// Buduje drzewo Huffmana i dekoduje podany tekst wejściowy
void budujDrzewo(string text)
{
    // liczymy częstotliwość występowania znaków
    unordered_map<char, int> czestotliwosc;
    for (char ch : text) {
        czestotliwosc[ch]++;
    }

    // tworzymy kolejkę
    priority_queue<wezel*, vector<wezel*>, comp> pq;

    // tworzymy węzeł liścia dla każdej kolejki priorytetowej
    for (auto pair : czestotliwosc) {
        pq.push(pobierzWezel(pair.first, pair.second, nullptr, nullptr));
    }

    // wykonuje się dopóki w kolejce będzie więcej niż jeden węzeł
    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. Dodajemy nowy węzeł do kolejki priorytetowej.
        int sum = lewy->czestotliwosc + prawy->czestotliwosc;
        pq.push(pobierzWezel('\0', sum, lewy, prawy));
    }
```

```
    // root przechowuje wskaźnik do korzenia drzewa
    wezel* root = pq.top();

    //przechowujemy kody Huffmana na mapie
    unordered_map<char, string> kodHuffmana;
    koduj(root, "", kodHuffmana);

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



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



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

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

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

```
int main()
{
    string tekst;
    cout << "Podaj tekst do zakodowania: " << 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 << "\nPoszczegolne znaki wystepuja w nastepujacej ilosci:" << endl;
    for (i = 0; i < 26; i++)
        cout << char(i + 'a') << " : " << alphabet[i] << endl;

    return 0;
}
```

Rysunek 1 Kod programu

2. Działanie programu

```
Konsola debugowania programu Microsoft Visual Studio
Podaj tekst do zakodowania:
aaaaaaaabbbbbbbbccccccccddddddddbbbbbbbb
Kody Huffmana maja postac:

a 00
c 01
d 10
b 11

Oryginalny tekst mial postac:
aaaaaaaabbbbbbbbccccccccddddddddbbbbbbbb


Zakodowany tekst ma postac:
0000000000000000111111111111111101010101010101101010101010101011111111111111

Odkodowany tekst ma postac:
aaaaaaaabbbbbbbbccccccccddddddddbbbbbbbb
Poszczegolne znaki wystepuja w nastepujacej ilosci:
a : 8
b : 16
c : 8
d : 8
```

Rysunek 2 Działanie programu

Działanie Naszego kodu można opisać w kilku prostych krokach.

1) Alokujemy nowe drzewo

2) Następnie porównujemy poszczególne obiekty ze sobą. Element o najwyższym priorytecie ma najniższą częstotliwość występowania.

3) Kolejno – tworzymy z każdego wpisu węzeł drzewa. Przemierzamy drzewo i sortujemy węzły.

4) Po uprzednim posortowaniu wybieramy dwa najmniejsze węzły i łączymy je w jeden poprzez zsumowanie gałęzi.

5) Liczba w węźle rodzicu jest sumą ilości wystąpień dzieci. Tak utworzony węzeł dodajemy do listy.

6) Jeżeli ilość węzłów jest większa niż 1 – wracamy do sortowania ich.

7) Usuwamy dwa węzły z najwyższym priorytetem – najniższą częstotliwością.

8) Tworzymy nowy węzeł wewnętrzny.

9) Przydzielamy krawędziom bity 0 oraz 1. Wszystkim krawędziom wychodzącym w stronę lewego dziecka nadajemy bit 1 a krawędziom wychodzącym w stronę prawego dziecka 0. Nie jest istotne, z której strony znajduje się który bit. Ważne by zawsze z lewej strony był ten sam i analogicznie z prawej.

10) Dla każdej literki tworzymy kod czytając wartości przy krawędziach od góry.

11) Przystępujemy w Naszym programie do kodowania / dekodowania podanego wcześniej przez Nas kodu.