- ## Opis problemu:

Kodowanie Huffmana (ang. Huffman Coding) – jedna z najprostszych metod kompresji bezstratnej opracowana w 1952r. przez Davida Huffman. Tworzy optymalny kod prefiksowy, którego długość jest odwrotnie proporcjonalna do prawdopodobieństwa wystąpienia znaku tz. im częściej dany symbol występuje w ciągu danych, tym mniej zajmie bitów. Jest to przykład algorytmu zachłannego.

Niech K będzie kodowaniem źródła informacji, tj. alfabetu $A = \{a_1, a_2, \dots, a_n\}$. Wraz z przyporządkowaną mu dystrybucją prawdopodobieństw. Oznaczamy przez d_i ($gdzie\ 1 \leq i \leq n$) długość słowa kodowego $K(a_i)$. Średnią długością słowa kodowego nazywamy wielkość:

$$L(K) = \sum_{i=1}^{n} d_i P(a_i)$$

Czyli wartość oczekiwaną zmiennej losowej $\{(d_i, P(a_i)) : i \in \{1,2, \dots, n\}\}$ lub średnia ważona prawdopodobieństw.

Najbardziej efektywnym jest taki kod natychmiastowy, dla którego wielkość $L(K)$ jest najmniejsza. Dla danego źródła informacji S oznaczmy przez $L_{min}(S)$ najmniejszą długość średnią słowa kodowego, czyli minimum po $L(K)$ dla wszystkich możliwych kodować alfabetu A. Minimum to istnieje, ponieważ mamy tylko skończoną liczbę wartości sumy mniejszych od n. **Kod K dla którego $L(K) = L_{min}(S)$ nazywamy kodem Huffmana.**

Rozważmy następujące źródło informacji:

x	A	B	D	K	R
$P(x)$	$\frac{5}{11}$	$\frac{2}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{2}{11}$

Przypiszmy słowa kodowe w trzech różnych kodach następująco:

x	A	B	D	K	R
$K_1(x)$	000	001	010	011	100
$K_2(x)$	0	10	110	1110	1111
$K_3(x)$	0	100	101	110	111

Otrzymujemy:

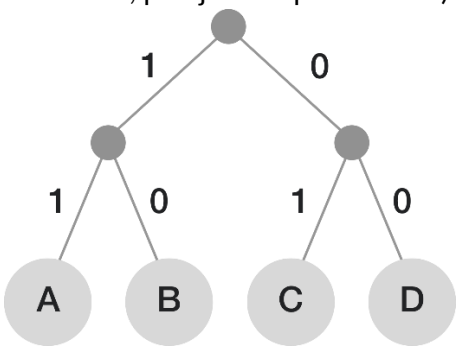
$$L(K_1) = 3 \qquad L(K_2) = \frac{24}{11} \qquad L(K_3) = \frac{23}{11}$$

Zatem $L_{min}(S) \leq \frac{23}{11}$. Analizując wartość sumy pod kątem znalezienia wartości minimalnej, szybko zauważamy że mamy tu równość.

[źródło: WSTĘP DO TEORII INFORMACJI I KODOWANIA - dr. Grzegorz Szkibiel]

- ## Opis algorytmu:

Algorytm statycznego kodowania Huffmana zlicza częstotliwość wystąpień poszczególnych symboli w analizowanym pliku, tworząc tak zwany las, w którym każde drzewo zawiera jeden symbol oraz wagę będącą liczbą jego wystąpień. Następnie wybierając dwa drzewa o najniższej liczbie wystąpień łączy je, tworząc w ten sposób nowe drzewo z sumą wag drzew składowych. Drzewa składowe z kolei stają się jego dziećmi oraz liśćmi drzewa binarnego. Operacja ta wykonuje się aż do uzyskania jednego drzewa, którego liście zamierają kodowane symbole. Dla każdego symbolu algorytm wyznacza słowo kodowe będące ścieżką od korzenia do liścia zawierającego dany symbol (przejście w lewo bit 0, przejście w prawo bit 1).



Przykładowe drzewo binarne wygenerowane przez algorytm.

Tworzenie lasu prawdopodobieństwa:

- Otwieram analizowany plik a jego zawartość kopiuję do tablicy char(1B) o odpowiedniej długości.
- Tworzę obiekt struktury reprezentujący drzewo/liść o następującej budowie:

```
struct HTNode
{
        HTNode * left; // wskaźnik do lewego dziecka
        HTNode * right; // wskaźnik do prawego dziecka
        bool type = false; // false - węzeł, true - liść;
        int count = 0; // ilość wystąpień
        char key; // symbol
};
```

- Tworze listę drzew reprezentującą las,
- Iteruję tablicę ze znakami:
  - Jeżeli znak nie występował wcześniej tworzę nowe drzewo z odpowiednimi własnościami i dodaję je do listy.
  - Jeżeli reprezentacja znaku istnieje już na liście, wartość count zwiększam o jeden.
- Algorytm kończy swoje działanie gdy przeanalizuje całą tablicę charów, tworząc w ten sposób las, w którym każde drzewo zawiera jeden symbol oraz jego wagę.

Tworzenie drzewa Huffmana:

- Dopóki długość listy nie jest równa jeden wykonuje:
  - Sortuję listę drzew rosnąco ze względu na wartości count,
  - Tworzę drzewo (struktura HTNode), gdzie lewe i prawe dziecko (liście) to dwa pierwsze obiekty z listy, a waga (count) to suma wag dzieci.

        - Usuwam dwa pierwsze elementy z listy i dodaje na początek utworzone wcześniej drzewo.
    - Usunięcie elementu z listy nie usuwa obiektu ponieważ lista przechowuje tylko wskaźniki.
    - Utworzyłem w ten sposób drzewo Huffmana które posłuży do kodowania i dekodowania danych.

Kodowanie:

- Dla każdego elementu tablicy z nie zakodowanymi znakami rekurencyjnie przeszukuję drzewo Huffmana metodą pre-order (przejście wzdłużne):
    - Każde przejście w prawo/lewo zapisywane jest w tablicy binarnej (droga algorytmu),
    - W przypadku gdy algorytm 'cofa się' do wyższego węzła usuwany jest ostatni element tablicy,
    - Gdy algorytm znajdzie odpowiedni liść tz. znak jaki przechowuje liść będzie równał się znakowi z tablicy algorytm znalazł poprawną ścieżkę, (kod prefiksowy znaku).
    - Kod ten zapisywany jest do tablicy binarnej przechowującej wszystkie znalezione ścieżki zakodowanych znaków.
- W celu zapisania tak utworzonej tablicy binarnej do pliku, muszę dopełnić ją do takiej długości, żeby wyrażenie *długość* % 8 = 0, operacja ta pozwoli mi łatwo przekształcić ją na tablicę charów. (Nie ma wpływu na wielkość pliku a pozwoli mi to na późniejsze poprawne dekodowanie).
- Dzielę tak przygotowaną tablicę co 8 bitów tworząc z każdej sekwencji znak char który zapisuję do pliku.

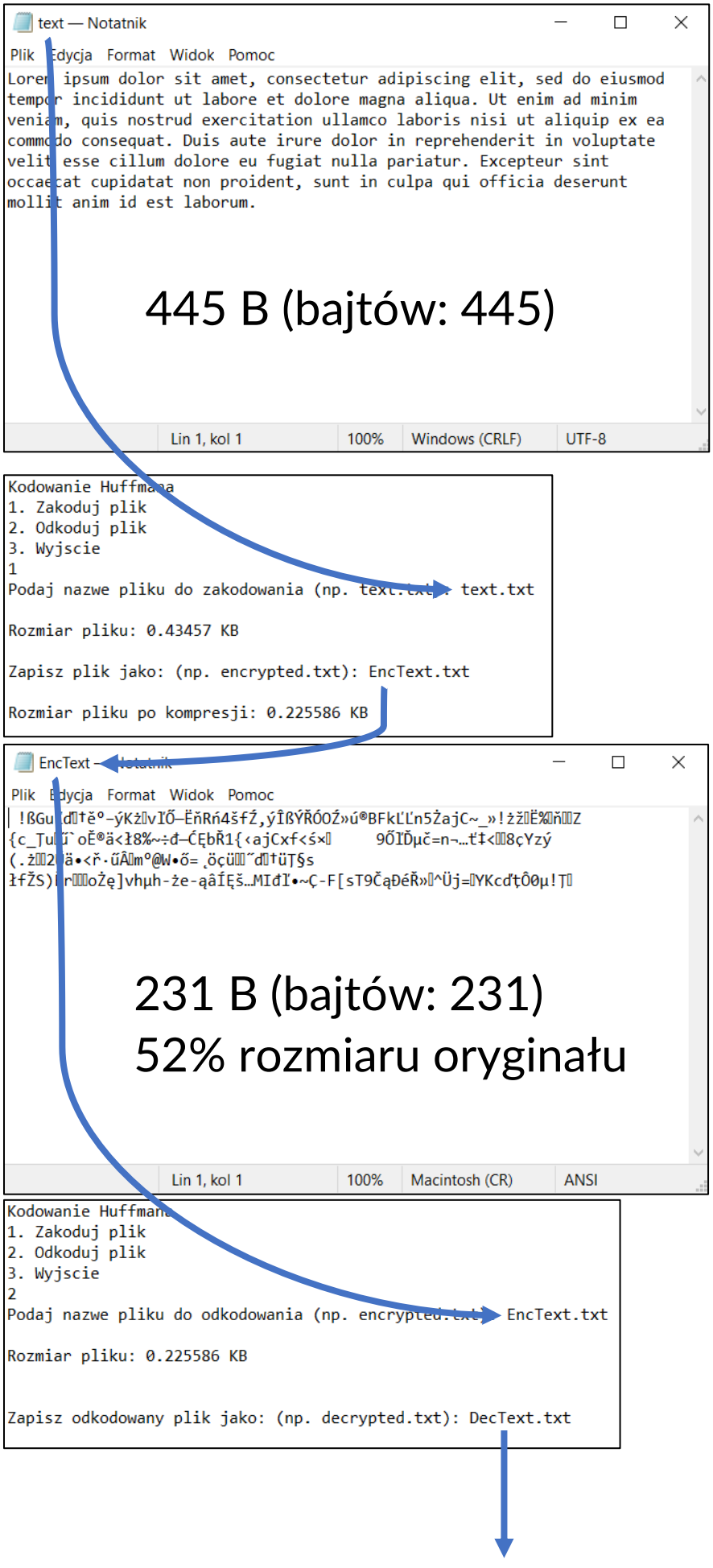
Dekodowanie:

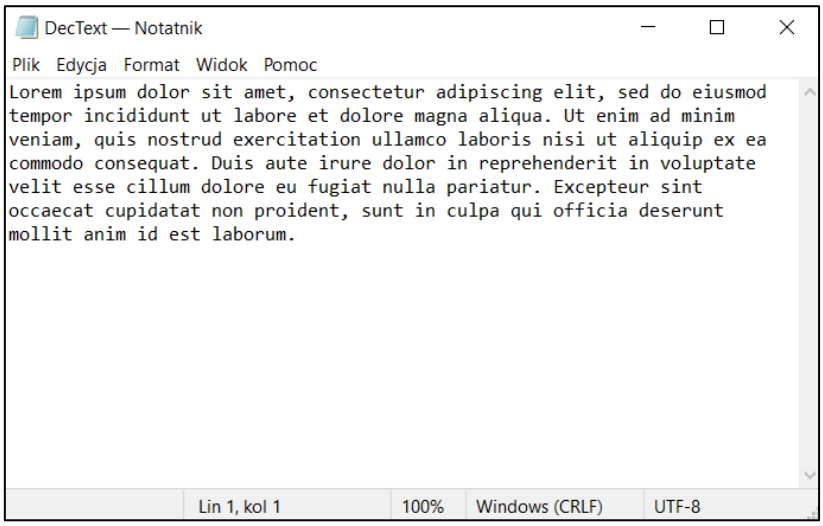
- Otwieram plik do dekodowania i kopiuję jego zawartość do tablicy char o odpowiedniej długości.
- Zamieniam znaki char z tablicy na wartości binarne i całość przechowuję w tablicy binarnej.
- Odczytuję zakodowane znaki iterując tablice binarną zawierającą cała sekwencję zero jedynkową. Zaczynając od korzenia, każde zero to przejście w lewo a każda jedynka to przejście w prawo. Gdy znajdziemy się w liściu oznacza to że odkodowaliśmy jeden znak i wracamy do korzenia.

Z racji, że nie zapisuję nigdzie drzewa binarnego albo informacji które pozwoliły by na odtworzenie go, dekodowanie pliku możliwe jest tylko w sytuacji gdy informacje te znajdują się w pamięci programu (nie zamkniemy go po zakodowaniu). Można było by się pokusić o stworzenie nagłówka zawierającego dane pozwalające na odtworzenie drzewa i tym samym stworzenie bardzo prymitywnego programu kompresującego.

Algorytm radzi sobie z **każdym rodzajem pliku nie tylko z tekstem**. Przykładowo 24bitowe zdjęcie BMP o rozmiarze 1,19 MB udało się skompresować do 653 KB co stanowi **53% pliku wejściowego.** Jedynym ograniczeniem jest rozmiar pliku (max ok. 2MB), ponieważ sposób przeszukiwania drzewa binarnego jest mało efektywny i czas kompresji bardzo się wydłuża.

- **Przykładowy proces kompresji na tekście łacińskim:**





- **Fragment kodu:**

```
void frequency()
{
    for(int i=0; i<size; i++)
    {
        bool exist = false;
        for( list<HTNode*>::iterator iter=Hlist.begin(); iter != Hlist.end(); iter++ )
            {
                if((*iter)->key == memblock[i])
                {
                    (*iter)->count++;
                    exist = true;
                }
            }
        if(!exist)
        {
            HTNode *node;
            node = new HTNode;
            node->key = memblock[i];
            node->count = 1;
            node->type = true;
            Hlist.push_back(node);
        }
```

Algorytm tworzący las oraz liczący częstotliwość znaku.

```
void createTree()
{
    while(Hlist.size() != 1)
    {
        HTNode * node;
        node = new HTNode;
        list<HTNode*>::iterator iter=Hlist.begin();
        node->left = (*iter);
        node->count = (*iter)->count;
        iter = Hlist.erase(iter);
        node->right = (*iter);
        node->count += (*iter)->count;
        iter = Hlist.erase(iter);

        Hlist.push_front(node);
        sort();
    }
}
```

Algorytm tworzący drzewo Huffmana.

Wnioski:

Program poprawnie koduje oraz dekoduje pliki korzystając z algorytmu Huffmana. Został przetestowany na wielu plikach (txt, bmp, png, pdf, wav) o różnych rozmiarach i w każdym przypadku odkodowany plik w 100% odpowiadał oryginałowi i nie został uszkodzony. Pomimo, że algorytm ten nie należy do najefektywniejszych obliczeniowo systemów kompresji, często wykorzystuje się go jako ostatni etap tego procesu (np. JPEG, MP3). Doczekał się on również swoich ulepszonych, nieco bardziej zaawansowanych wariacji np. adaptacyjne kodowanie Huffmana, kodowanie Huffmana z nierównych kosztów literowych.

Źródła:

[1]. Wstęp do teorii informacji i kodowania - dr. Grzegorz Szkibiel

[2]. Algorytmy i struktury danych - mgr Jerzy Wałaszek