Zadanie 14 – Teksty

Algorytm Huffmana to prosty algorytm kompresji danych, stosowany głównie w odniesieniu do tekstów.

Istota algorytmu polega na zbudowaniu drzewa binarnego, którego kształt zależy od częstotliwości występowania znaków w tekście. Znaki przechowywane są jedynie w liściach drzewa. Węzły wewnętrzne nie przechowują znaków. Węzeł albo jest liściem albo ma dwa następniki (nie może mieć tylko jednego). Kod Huffmama danego znaku determinowany jest przez ścieżkę od korzenia do liścia zawierającego ten znak - wybór lewego poddrzewa oznacza bit 0, a prawego bit 1. Idea algorytmu polega na tym, aby częściej występujące znaki miały krótsze kody. Jeśli drzewo jest jednoelementowe (sam korzeń) to odpowiada mu kod 0 (przypadek szczególny - przecież nie idziemy ani w lewo, ani w prawo).

Kompresja danych składa się zatem z następujących etapów:
- zbudowania drzewa Huffmana,
- utworzenia słownika kodów
- użycia słownika do zakodowania znaków w postaci binarnej.

Zadanie składa się z trzech etapów.

ETAP 1 (1.5 pkt.)

W **konstruktorze klasy Huffman** należy zbudować drzewo Huffmana. Składowa `root` powinna wskazywać na korzeń drzewa.

Algorytm budowania drzewa Huffmana dla napisu S jest następujący:
1. Określ częstość występowania każdego symbolu w napisie S.
2. Utwórz kolejkę priorytetową drzew binarnych, które w węzłach (klasa Node) przechowują informacje: symbol (pole `Node.character`) i liczba wystąpień symbolu (pole `Node.freq`). Priorytet - liczba wystąpień (mniejsza lepiej).
3. Na początku drzewa składają się wyłącznie z korzeni.
4. Dopóki w kolejce jest więcej niż jedno drzewo, powtarzaj:
    1. Pobierz z kolejki dwa pierwsze drzewa (o najmniejszych liczbach wystąpień reprezentowanych przez nie znaków).
    2. Wstaw nowe drzewo, w którego korzeniu jest suma liczby wystąpień odpowiadających pobranym drzewom, natomiast one same stają się jego lewym i prawym poddrzewem. Korzeń drzewa nie przechowuje symbolu.

Pomocniczo można zobaczyć animowany gif Huffman_demo.gif znajdujący się w archiwum z zadaniem. W tym etapie warto wykorzystać, dostarczoną w pliku PriorityQueue.cs, implementację kolejki priorytetowej.

ETAP 2 (1.5 pkt.)

Należy zaimplementować metodę `BitList Compress(string content)`.
Na podstawie otrzymanego w etapie 1 drzewa należy wypełnić słownik `codesMap` parami <znak, kod Huffmana>, a następnie należy zapisać tekst `content` jako skompresowany ciąg bitów w postaci obiektu klasy `BitList`. Szczególnie pomocna okaże się tu metoda `BitList.Append` (przypominamy - przy generowaniu kodów należy przyjąć, że lewa gałąź to bit 0, a prawa to 1).

ETAP 3 (1.0 pkt.)

Należy zaimplementować metodę `string Decompress(BitList compressed)`. Metoda ta powinna dekodować ciąg bitów podany w zmiennej `compressed`. (oczywiście korzystając z wygenerowanego w konstruktorze drzewa). Jako wynik operacji, powinien zostać zwrócony oryginalny napis.