Kodowanie Huffmana

Kompresja:

1) Wczytaj dane wejściowe.
2) Utwórz tablicę wystąpień każdego znaku (np. typu char). Wykorzystaj w tym celu dane wejściowe.
3) Utworzyć na potrzeby kodowania drzewo Huffmana (drzewo Huffmana ma odpowiadać liczbie wystąpień danego znaku w danych wejściowych).
4) Utworzyć odpowiednią tablicę słów kodowych, aby powiązać łańcuch bitów (bezprefiksowe słowo kodowe) z odpowiednim znakiem (char) z danych wejściowych.
5) Zapisać drzewo Huffmana zakodowane jako łańcuch bitów.
6) Zapisać liczbę znaków w danych wyjściowych zakodowaną jako łańcuch bitów.
7) Wykorzystać tablicę słów kodowych do "przetłumaczenia" znaków wejściowych na (bezprefiksowe) słowa kodowe oraz zapisania ich (tj. bezprefiksowych słów kodowych).

Rozpakowanie:

1) Wczytać drzewo Huffmana (zakodowane na początku pliku).
2) Wczytać liczbę znaków do odkodowania.
3) Wykorzystać drzewo Huffmana do odkodowania strumienia bitów.

Przykład 1 (C++). Implementacja kompresji metodą Huffmana (zrealizowane są tylko dwa pierwsze punkty (wiersze na niebiesko) oraz częściowo punkt trzeci (wiersze na zielono). Pola oznaczone szarym tłem w poniższym kodzie można pominąć. Plik z przykładową treścią do zakodowania do pobrania ze strony przedmiotu (shesells.txt).

[huff_v01.cpp]

```
// huff_v01.cpp
// demonstrates Huffman code (only a few first points from list above are
// completed)
// Andrzej Pisarski
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using std::vector;
using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::ifstream;
using std::noskipws;

class Character
{
    public:
        char c;
        int nTimes;

        Character(char ct=-1,int nt=0):c(ct),nTimes(nt)
        {}
```

```
        void insChar(char ic){
            c=ic;
        }

        void insnTimes(int it){
            nTimes=it;
        }

        void plusnTimes(){
            nTimes++;
        }

        char getChar(){
            return c;
        }

        int getnTimes(){
            return nTimes;
        }
};

class Link{
    public:
        char LChar;
        int LnTimes;
        Link *pNext;
        Link *pLeftChild;
        Link *pRightChild;

        Link(char lc, int lnt): LChar(lc), LnTimes(lnt), pNext(NULL),
            pLeftChild(NULL), pRightChild(NULL)
        {}

        void display(){
            cout<<LChar<<" "<<LnTimes<<endl;
        }
};

class ReadFile{
    public:
        vector<char> text;
        string fullpath;
        vector<Character> freq;
        Link* pFirst;
        int nElems;
        ReadFile(const string &name, const string &path=""):
            fullpath("  "), pFirst(NULL)
        {
            fullpath=path+name+".txt";
            ifstream f(fullpath.c_str());

            char temp;
            while(f>>noskipws>>temp)
                text.push_back(temp);

            f.close();

            freq.resize(256);
            for(unsigned int i=0;i<255;i++){
                freq[i].c=i;
```

```
        }
    }


    void display(){
        for(unsigned int i=0;i<text.size();i++)
            cout<<text[i];
    }

    void findFreq(){
        for(unsigned int i=0;i<text.size();i++){
            freq[text[i]].nTimes++;
        }

    }

    void displayF(){
        for(unsigned int i=0;i<256;i++){
            int ntemp=freq[i].nTimes;
            if(ntemp>0){
                cout<<freq[i].c<<" "<<ntemp<<endl;
            }
        }
    }

    void bubbleSort(){
        for(unsigned int i=0;i<freq.size();i++){
            for(unsigned int j=1;j<freq.size()-i;j++){
                if(freq[j].nTimes<freq[j-1].nTimes){
                    Character temp=freq[j-1];
                    freq[j-1]=freq[j];
                    freq[j]=temp;
                }
            }
        }
    }

    void makeList(){
        if(pFirst==NULL){
            pFirst=new Link(freq[0].c,freq[0].nTimes);

            Link* pPrevious=pFirst;
            for(int i=1;i<256;i++){
                if(freq[i].nTimes>0){
                    Link* temp=new Link(freq[i].c,freq[i].nTimes);
                    pPrevious->pNext=temp;
                    pPrevious=temp;
                    nElems++;
                }
            }
        }
    }

    void displayList(){
            Link* pPrevious=pFirst;
            while(pPrevious!=NULL){
                pPrevious->display();
                pPrevious=pPrevious->pNext;
            }
    }
};
```

```
int main(){

    string filename="shesells";
    ReadFile poem(filename);

    poem.display();

    poem.findFreq();
    poem.displayF();

    cout<<"=============="<<endl;

    poem.bubbleSort();
    poem.displayF();

    cout<<"##############"<<endl;

    poem.makeList();
    poem.displayList();

}
```

Opracowano na podstawie:
Robert Lafore – "Teach Yourself Data Structures And Algorithms In 24 Hours"
Robert Sedgewick – "Algorytmy"