Huffman

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Compare Struct Reference

Porownanie wezlow dla kolejki priorytetowej.

`#include <huffman.h>`

Public Member Functions

- bool operator() (Node *a, Node *b)
  Operator porownania.

3.1.1 Detailed Description

Porownanie wezlow dla kolejki priorytetowej.

Struktura sluzy do porownywania dwoch wezlow drzewa Huffmanna na podstawie ich czestotliwosci. Jest uzywana przy implementacji kolejki priorytetowej.

Definition at line 45 of file huffman.h.

3.1.2 Member Function Documentation

3.1.2.1 operator()()

```
bool Compare::operator() (
            Node * a,
            Node * b)  [inline]
```

Operator porownania.

Porownuje dwa wezly drzewa Huffmanna.

Parameters

a	Wskaznik na pierwszy wezel.
b	Wskaznik na drugi wezel.

Returns

true, jesli czestotliwosc wezla a jest wieksza niz czestotliwosc wezla b, w przeciwnym razie false.

Definition at line 55 of file huffman.h.

The documentation for this struct was generated from the following file:

- projekt/HuffmanV1/HuffmanV1/huffman.h

3.2 Node Struct Reference

Struktura wezla drzewa Huffmanna.

`#include <huffman.h>`

Public Member Functions

- Node (char sym, int freq)
  Konstruktor wezla.

Public Attributes

- char symbol
- int frequency
- Node * left
- Node * right

3.2.1 Detailed Description

Struktura wezla drzewa Huffmanna.

Ta struktura reprezentuje wezel uzywany w drzewie Huffmanna, gdzie przechowywany jest symbol oraz czestotliwosc wystepowania tego symbolu. Posiada rowniez wskazniki do lewej i prawej galezi drzewa.

Definition at line 21 of file huffman.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node()

```
Node::Node (
            char sym,
            int freq)  [inline]
```

Konstruktor wezla.

Inicjalizuje wezel drzewa Huffmanna dla podanego symbolu i czestotliwosci. Wskazniki do lewego oraz prawego dziecka sa ustawione na nullptr.

Parameters

sym	Symbol, ktory reprezentuje dany wezel.
freq	Czestotliwosc wystepowania symbolu.

Definition at line 36 of file huffman.h.

3.2.3 Member Data Documentation

3.2.3.1 frequency

```
int Node::frequency
```

Czestotliwosc wystepowania symbolu.

Definition at line 23 of file huffman.h.

3.2.3.2 left

```
Node* Node::left
```

Wskaznik na lewe dziecko w drzewie Huffmanna.

Definition at line 24 of file huffman.h.

3.2.3.3 right

```
Node* Node::right
```

Wskaznik na prawe dziecko w drzewie Huffmanna.

Definition at line 25 of file huffman.h.

3.2.3.4 symbol

```
char Node::symbol
```

Symbol reprezentowany przez wezel.

Definition at line 22 of file huffman.h.

The documentation for this struct was generated from the following file:

- projekt/HuffmanV1/HuffmanV1/huffman.h

Chapter 4

File Documentation

4.1 projekt/HuffmanV1/HuffmanV1/huffman.cpp File Reference

```
#include "huffman.h"
```

Functions

- void buildCodeTable (Node *root, const string &str, unordered_map< char, string > &huffmanCode)

  Buduje tablice kodow Huffmana przez rekurencyjne przeszukiwanie drzewa.
- void encodeFile (const string &inputFile, const string &outputFile)

  Koduje plik wejsciowy przy uzyciu algorytmu Huffmana.
- void decodeFile (const string &inputFile, const string &outputFile)

  Dekoduje plik zakodowany algorytmem Huffmana.

4.1.1 Function Documentation

4.1.1.1 buildCodeTable()

```
void buildCodeTable (
            Node * root,
            const string & str,
            unordered_map< char, string > & huffmanCode)
```

Buduje tablice kodow Huffmana przez rekurencyjne przeszukiwanie drzewa.

Buduje tabele kodow Huffmanna.

Funkcja rekurencyjnie przechodzi przez drzewo Huffmana i przypisuje kazdemu lisciowi (symbolowi) odpowiedni kod binarny. Do lewego poddrzewa dodawane jest "0", a do prawego "1".

Parameters

root	Wskaznik do biezacego wezla drzewa Huffmana.
str	Aktualnie zbudowany kod binarny.
huffmanCode	Mapa, ktora przechowuje ostateczne kody Huffmana dla kazdego symbolu.

Definition at line 13 of file huffman.cpp.

4.1.1.2 decodeFile()

```
void decodeFile (
            const string & inputFile,
            const string & outputFile)
```

Dekoduje plik zakodowany algorytmem Huffmana.

Dekoduje zawartosc zakodowanego pliku przy uzyciu drzewa Huffmanna.

Funkcja odczytuje z pliku zakodowany tekst oraz tablice kodow (w postaci klucz: kod binarny, wartosc: symbol), odwraca mapowanie, a nastepnie dekoduje tekst przy uzyciu odwroconej tablicy kodow. Zdekodowany tekst jest zapisywany do pliku wyjsciowego.

Parameters

inputFile	Nazwa pliku wejsciowego z zakodowanym tekstem.
outputFile	Nazwa pliku wyjsciowego, do ktorego zapisany zostanie zdekodowany tekst.

Definition at line 113 of file huffman.cpp.

4.1.1.3 encodeFile()

```
void encodeFile (
            const string & inputFile,
            const string & outputFile)
```

Koduje plik wejsciowy przy uzyciu algorytmu Huffmana.

Koduje zawartosc pliku wejsciowego przy uzyciu drzewa Huffmanna.

Funkcja odczytuje zawartosc pliku wejsciowego, liczy czestotliwosc wystepowania kazdego znaku, buduje drzewo Huffmana, generuje kody dla znakow, a nastepnie koduje tekst. Zakodowany tekst oraz tablica kodow sa zapisywane do pliku wyjsciowego.

Parameters

inputFile	Nazwa pliku wejsciowego (sciezka do pliku).
outputFile	Nazwa pliku wyjsciowego (sciezka do pliku).

Definition at line 36 of file huffman.cpp.

4.2 huffman.cpp

Go to the documentation of this file.

```
00001 #include "huffman.h"
00002
00013 void buildCodeTable(Node* root, const string& str, unordered_map<char, string>& huffmanCode) {
00014     if (!root)
00015         return;
00016     // Jezeli wezel jest lisciem (brak lewego i prawego potomka), przypisz kod do symbolu.
00017     if (!root->left && !root->right) {
00018         huffmanCode[root->symbol] = str;
00019     }
00020     // Rekurencyjne przejscie do lewego poddrzewa z dodaniem "0" do kodu.
00021     buildCodeTable(root->left, str + "0", huffmanCode);
00022     // Rekurencyjne przejscie do prawego poddrzewa z dodaniem "1" do kodu.
00023     buildCodeTable(root->right, str + "1", huffmanCode);
00024 }
00025
00036 void encodeFile(const string& inputFile, const string& outputFile) {
00037     // Otwieranie pliku wejsciowego w trybie binarnym.
00038     ifstream input(inputFile, ios::binary);
00039     if (!input.is_open()) {
00040         cerr « "Nie mozna otworzyc pliku wejsciowego!" « endl;
00041         return;
00042     }
00043
00044     // Zliczanie czestotliwosci wystepowania znakow.
00045     unordered_map<char, int> frequency;
00046     stringstream buffer;
00047     buffer « input.rdbuf();
00048     string text = buffer.str();
00049     input.close();
00050
00051     // Przeliczanie wystapienia kazdego znaku w tekscie.
00052     for (char ch : text) {
00053         frequency[ch]++;
00054     }
00055
00056     // Tworzenie kolejki priorytetowej do budowy drzewa Huffmana.
00057     priority_queue<Node*, vector<Node*>, Compare> pq;
00058     for (auto pair : frequency) {
00059         pq.push(new Node(pair.first, pair.second));
00060     }
00061
00062     // Budowa drzewa Huffmana.
00063     while (pq.size() > 1) {
00064         Node* left = pq.top();
00065         pq.pop();
00066         Node* right = pq.top();
00067         pq.pop();
00068         // Tworzenie nowego wezla sumujacego czestotliwosci.
00069         Node* sum = new Node('\0', left->frequency + right->frequency);
00070         sum->left = left;
00071         sum->right = right;
00072         pq.push(sum);
00073     }
00074     Node* root = pq.top();
00075
00076     // Generowanie tablicy kodow Huffmana.
00077     unordered_map<char, string> huffmanCode;
00078     buildCodeTable(root, "", huffmanCode);
00079
00080     // Kodowanie tekstu przy pomocy wygenerowanych kodow.
00081     string encodedText;
00082     for (char ch : text) {
00083         encodedText += huffmanCode[ch];
00084     }
00085
00086     // Zapisanie tablicy kodow i zakodowanego tekstu do pliku wyjsciowego.
00087     ofstream output(outputFile, ios::binary);
00088     if (!output.is_open()) {
00089         cerr « "Nie mozna otworzyc pliku wyjsciowego!" « endl;
00090         return;
00091     }
00092     // Zapis rozmiaru tablicy kodow.
00093     output « huffmanCode.size() « endl;
00094     // Zapis par: symbol:kod.
00095     for (auto pair : huffmanCode) {
00096         output « pair.first « ":" « pair.second « endl;
00097     }
00098     // Zapis zakodowanego tekstu.
00099     output « encodedText;
00100     output.close();
00101 }
00102
```

```
00113 void decodeFile(const string& inputFile, const string& outputFile) {
00114     // Otwieranie pliku wejsciowego w trybie binarnym.
00115     ifstream input(inputFile, ios::binary);
00116     if (!input.is_open()) {
00117         cerr « "Nie mozna otworzyc pliku wejsciowego!" « endl;
00118         return;
00119     }
00120 
00121     // Odczyt tablicy kodow zapisanej w pliku.
00122     unordered_map<string, char> reverseHuffmanCode; // Klucz: kod binarny, Wartosc: symbol.
00123     int tableSize; // Liczba roznych elementow (symboli) zapisanych w tablicy.
00124     input » tableSize;
00125     input.ignore(); // Ignorowanie znaku nowej linii.
00126 
00127     for (int i = 0; i < tableSize; i++) {
00128         char symbol;
00129         string code;
00130         input.get(symbol);
00131         input.ignore(1); // Pomijanie dwukropka ':'.
00132         getline(input, code);
00133         reverseHuffmanCode[code] = symbol;
00134     }
00135 
00136     // Odczyt zakodowanego tekstu z pliku.
00137     string encodedText;
00138     getline(input, encodedText, '\0');
00139     input.close();
00140 
00141     // Dekodowanie tekstu przy uzyciu odwroconej tablicy kodow.
00142     string decodedText;
00143     string currentCode;
00144     for (char bit : encodedText) {
00145         currentCode += bit;
00146         if (reverseHuffmanCode.count(currentCode)) {
00147             decodedText += reverseHuffmanCode[currentCode];
00148             currentCode = "";
00149         }
00150     }
00151 
00152     // Zapis zdekodowanego tekstu do pliku wyjsciowego.
00153     ofstream output(outputFile, ios::binary);
00154     if (!output.is_open()) {
00155         cerr « "Nie mozna otworzyc pliku wyjsciowego!" « endl;
00156         return;
00157     }
00158     output « decodedText;
00159     output.close();
00160 }
```

4.3 projekt/HuffmanV1/HuffmanV1/huffman.h File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <unordered_map>
#include <queue>
#include <vector>
#include <string>
```

Classes

- struct Node

  Struktura wezla drzewa Huffmanna.

- struct Compare

  Porownanie wezlow dla kolejki priorytetowej.

Functions

- void encodeFile (const string &inputFile, const string &outputFile)

  Koduje zawartosc pliku wejsciowego przy uzyciu drzewa Huffmanna.
- void decodeFile (const string &inputFile, const string &outputFile)

  Dekoduje zawartosc zakodowanego pliku przy uzyciu drzewa Huffmanna.
- void buildCodeTable (Node *root, const string &str, unordered_map< char, string > &huffmanCode)

  Buduje tabele kodow Huffmanna.

4.3.1 Function Documentation

4.3.1.1 buildCodeTable()

```
void buildCodeTable (
            Node * root,
            const string & str,
            unordered_map< char, string > & huffmanCode)
```

Buduje tabele kodow Huffmanna.

Funkcja rekurencyjnie przechodzi drzewo Huffmanna i przypisuje kazdemu symbolowi odpowiedni kod (ciag znakow) na podstawie jego pozycji w drzewie.

Parameters

root	Wskaznik na korzen drzewa Huffmanna.
str	Aktualny ciag kodu (sciezka do danego wezla) w drzewie.
huffmanCode	Referencja do mapy, w ktorej zapisywane sa kody Huffmanna.

Buduje tabele kodow Huffmanna.

Funkcja rekurencyjnie przechodzi przez drzewo Huffmana i przypisuje kazdemu lisciowi (symbolowi) odpowiedni kod binarny. Do lewego poddrzewa dodawane jest "0", a do prawego "1".

Parameters

root	Wskaznik do biezacego wezla drzewa Huffmana.
str	Aktualnie zbudowany kod binarny.
huffmanCode	Mapa, ktora przechowuje ostateczne kody Huffmana dla kazdego symbolu.

Definition at line 13 of file huffman.cpp.

4.3.1.2 decodeFile()

```
void decodeFile (
            const string & inputFile,
            const string & outputFile)
```

Dekoduje zawartosc zakodowanego pliku przy uzyciu drzewa Huffmanna.

Funkcja odtwarza oryginalna zawartosc pliku, ktory zostal zakodowany metoda Huffmanna.

Parameters

inputFile	Sciezka do pliku wejsciowego z zakodowanymi danymi.
outputFile	Sciezka do pliku, do ktorego zostanie zapisana zdekompresowana zawartosc.

Dekoduje zawartosc zakodowanego pliku przy uzyciu drzewa Huffmanna.

Funkcja odczytuje z pliku zakodowany tekst oraz tablice kodow (w postaci klucz: kod binarny, wartosc: symbol), odwraca mapowanie, a nastepnie dekoduje tekst przy uzyciu odwroconej tablicy kodow. Zdekodowany tekst jest zapisywany do pliku wyjsciowego.

Parameters

inputFile	Nazwa pliku wejsciowego z zakodowanym tekstem.
outputFile	Nazwa pliku wyjsciowego, do ktorego zapisany zostanie zdekodowany tekst.

Definition at line 113 of file huffman.cpp.

4.3.1.3 encodeFile()

```
void encodeFile (
            const string & inputFile,
            const string & outputFile)
```

Koduje zawartosc pliku wejsciowego przy uzyciu drzewa Huffmanna.

Funkcja przetwarza plik wejsciowy, koduje jego zawartosc metoda Huffmanna i zapisuje wynik do pliku wyjsciowego.

Parameters

inputFile	Sciezka do pliku wejsciowego.
outputFile	Sciezka do pliku wyjsciowego.

Koduje zawartosc pliku wejsciowego przy uzyciu drzewa Huffmanna.

Funkcja odczytuje zawartosc pliku wejsciowego, liczy czestotliwosc wystepowania kazdego znaku, buduje drzewo Huffmana, generuje kody dla znakow, a nastepnie koduje tekst. Zakodowany tekst oraz tablica kodow sa zapisywane do pliku wyjsciowego.

Parameters

inputFile	Nazwa pliku wejsciowego (sciezka do pliku).
outputFile	Nazwa pliku wyjsciowego (sciezka do pliku).

Definition at line 36 of file huffman.cpp.

4.4 huffman.h

Go to the documentation of this file.

```
00001 #ifndef HUFFMAN_H
00002 #define HUFFMAN_H
00003
00004 #include <iostream>
00005 #include <fstream>
00006 #include <sstream>
00007 #include <unordered_map>
00008 #include <queue>
00009 #include <vector>
00010 #include <string>
00011
00012 using namespace std;
00013
00021 struct Node {
00022     char symbol;
00023     int frequency;
00024     Node* left;
00025     Node* right;
00026
00036     Node(char sym, int freq) : symbol(sym), frequency(freq), left(nullptr), right(nullptr) {}
00037 };
00038
00045 struct Compare {
00055     bool operator()(Node* a, Node* b) {
00056         return a->frequency > b->frequency;
00057     }
00058 };
00059
00069 void encodeFile(const string& inputFile, const string& outputFile);
00070
00079 void decodeFile(const string& inputFile, const string& outputFile);
00080
00091 void buildCodeTable(Node* root, const string& str, unordered_map<char, string>& huffmanCode);
00092
00093 #endif // HUFFMAN_H
```

4.5 projekt/HuffmanV1/HuffmanV1/main.cpp File Reference

```
#include <iostream>
#include <cstring>
#include "huffman.h"
```

Functions

- int main (int argc, char ∗argv[ ])

  Glowna funkcja aplikacji do kodowania Huffmana.

4.5.1 Function Documentation

4.5.1.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

Glowna funkcja aplikacji do kodowania Huffmana.

Funkcja ta przetwarza argumenty wiersza polecen, aby zakodowac lub zdekodowac plik przy uzyciu kodowania Huffmana. Sprawdza, czy podano wszystkie wymagane argumenty oraz upewnia sie, ze operacje kodowania i dekodowania nie sa wykonywane jednoczesnie.

Sposob uzycia: program -i <plik_wejsciowy> -o <plik_wyjsciowy> -c/-d

Parameters

argc	Liczba argumentow wiersza polecen.
argv	Tablica ciagow znakow z argumentami wiersza polecen.

Returns

int Zwraca 0 przy powodzeniu lub 1 w przypadku bledu.

Definition at line 21 of file main.cpp.

4.6 main.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <cstring>
00003 #include "huffman.h"
00004
00005 using namespace std;
00006
00021 int main(int argc, char* argv[]) {
00022     // Sprawdzenie, czy podano minimalna liczbe argumentow
00023     if (argc < 5) {
00024         cerr « "Uzycie: program -i <plik_wejsciowy> -o <plik_wyjsciowy> -c/-d" « endl;
00025         return 1;
00026     }
00027
00028     string inputFile, outputFile;
00029     bool encode = false, decode = false;
00030
00031     // Przetwarzanie opcji z wiersza polecen
00032     for (int i = 1; i < argc; i++) {
00033         // Sprawdzenie opcji pliku wejsciowego
00034         if (strcmp(argv[i], "-i") == 0) {
00035             inputFile = argv[++i];
00036         }
00037         // Sprawdzenie opcji pliku wyjsciowego
00038         else if (strcmp(argv[i], "-o") == 0) {
00039             outputFile = argv[++i];
00040         }
00041         // Ustawienie flagi kodowania, gdy znaleziono opcje "-c"
00042         else if (strcmp(argv[i], "-c") == 0) {
00043             encode = true;
00044         }
00045         // Ustawienie flagi dekodowania, gdy znaleziono opcje "-d"
00046         else if (strcmp(argv[i], "-d") == 0) {
00047             decode = true;
00048         }
00049     }
00050
00051     // Upewnienie sie, ze nie wybrano jednoczesnie opcji kodowania i dekodowania
00052     if (encode && decode) {
00053         cerr « "Opcje -c i -d nie moga byc uzywane jednoczesnie!" « endl;
00054         return 1;
00055     }
00056
00057     // Wykonanie odpowiedniej operacji w zaleznosci od podanej opcji
00058     if (encode) {
00059         encodeFile(inputFile, outputFile);
00060     }
00061     else if (decode) {
00062         decodeFile(inputFile, outputFile);
00063     }
00064     else {
00065         cerr « "Brak podanej operacji kodowania lub dekodowania!" « endl;
00066         return 1;
00067     }
00068
00069     return 0;
00070 }
```

Index