

Huffman coding

Generated by Doxygen 1.9.3

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 WezelDrzewa Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 ch	6
3.1.2.2 left	6
3.1.2.3 parent	6
3.1.2.4 probability	6
3.1.2.5 right	6
4 File Documentation	7
4.1 Helpers.cpp File Reference	7
4.1.1 Function Documentation	8
4.1.1.1 binary_to_decimal()	8
4.1.1.2 calculate_probability()	8
4.1.1.3 coded_message()	9
4.1.1.4 compare_by_prob()	9
4.1.1.5 create_dictionary()	10
4.1.1.6 create_tree()	10
4.1.1.7 decimal_to_binary()	11
4.1.1.8 decode_from_binary_to_text()	11
4.1.1.9 HuffmanDecode()	11
4.1.1.10 HuffmanEncode()	13
4.1.1.11 load_dic_file()	13
4.1.1.12 output_dictionary()	14
4.1.1.13 output_to_binary()	14
4.1.1.14 setPathByFlag()	14
4.1.1.15 split_string()	15
4.1.1.16 validate_input_arguments()	15
4.1.2 Variable Documentation	15
4.1.2.1 dictionary	15
4.2 Helpers.cpp	16
4.3 Helpers.h File Reference	20
4.3.1 Function Documentation	20
4.3.1.1 binary_to_decimal()	20
4.3.1.2 calculate_probability()	21
4.3.1.3 coded_message()	21

4.3.1.4 compare_by_prob()	22
4.3.1.5 create_dictionary()	22
4.3.1.6 create_tree()	23
4.3.1.7 decimal_to_binary()	23
4.3.1.8 decode_from_binary_to_text()	24
4.3.1.9 HuffmanDecode()	24
4.3.1.10 HuffmanEncode()	24
4.3.1.11 load_dic_file()	26
4.3.1.12 output_dictionary()	26
4.3.1.13 output_to_binary()	27
4.3.1.14 setPathByFlag()	27
4.3.1.15 split_string()	27
4.3.1.16 validate_input_arguments()	28
4.4 Helpers.h	28
4.5 huffman-coding.cpp File Reference	28
4.5.1 Function Documentation	29
4.5.1.1 main()	29
4.6 huffman-coding.cpp	29

Chapter 1

Class Index

1.1 Class List

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

[WezelDrzewa](#)

Struktura używana przy tworzeniu drzewa, przechowuje znak oraz prawdopodobieństwo jego wystąpienia w tekście

5

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Helpers.cpp	7
Helpers.h	20
huffman-coding.cpp	28

Chapter 3

Class Documentation

3.1 WezelDrzewa Struct Reference

Struktura używana przy tworzeniu drzewa, przechowuje znak oraz prawdopodobieństwo jego wystąpienia w tekście.

```
#include <Helpers.h>
```

Public Attributes

- struct [WezelDrzewa](#) * [parent](#) = NULL
rodzic węzła
- struct [WezelDrzewa](#) * [left](#) = NULL
lewe dziecko
- struct [WezelDrzewa](#) * [right](#) = NULL
prawe dziecko
- string [ch](#)
znak
- double [probability](#) = NULL

3.1.1 Detailed Description

Struktura używana przy tworzeniu drzewa, przechowuje znak oraz prawdopodobieństwo jego wystąpienia w tekście.

Definition at line [23](#) of file [Helpers.h](#).

3.1.2 Member Data Documentation

3.1.2.1 ch

```
string WezelDrzewa::ch
```

znak

Definition at line 44 of file [Helpers.h](#).

3.1.2.2 left

```
struct WezelDrzewa* WezelDrzewa::left = NULL
```

lewe dziecko

Definition at line 34 of file [Helpers.h](#).

3.1.2.3 parent

```
struct WezelDrzewa* WezelDrzewa::parent = NULL
```

rodzic węzła

Definition at line 29 of file [Helpers.h](#).

3.1.2.4 probability

```
double WezelDrzewa::probability = NULL
```

Definition at line 45 of file [Helpers.h](#).

3.1.2.5 right

```
struct WezelDrzewa* WezelDrzewa::right = NULL
```

prawe dziecko

Definition at line 39 of file [Helpers.h](#).

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

- [Helpers.h](#)

Chapter 4

File Documentation

4.1 Helpers.cpp File Reference

```
#include "Helpers.h"
#include <iostream>
#include <vector>
#include <map>
#include <utility>
#include <fstream>
#include <bitset>
#include <cstdlib>
#include <string>
#include <algorithm>
```

Functions

- int [validate_input_arguments](#) (int argc, vector< string > &arg)
- void [setPathByFlag](#) (string &flag, string &path, string &input_path, string &output_path, string &dic_path, string &mode)
- int [binary_to_decimal](#) (string &in)
- string [decimal_to_binary](#) (int in)
- pair< string, char > [split_string](#) (string napis)
- bool [compare_by_prob](#) (const [WezelDrzewa](#) &a, const [WezelDrzewa](#) &b)
- map< char, double > [calculate_probability](#) (string inputname)
- void [create_dictionary](#) ([WezelDrzewa](#) *tree, map< char, string > &[dictionary](#), string prevstring)
- template<typename K , typename V >
void [output_dictionary](#) (map< K, V > const &m, string dicfile, int zeros_added)
- [WezelDrzewa](#) [create_tree](#) (map< char, double > chars_with_prob)
- pair< string, int > [coded_message](#) (map< char, string > [dictionary](#), string inputfile)
- void [output_to_binary](#) (string huffman_coded_string, string outputfile)
- pair< map< string, char >, int > [load_dic_file](#) (string dicFile)
- void [decode_from_binary_to_text](#) (map< string, char > decoDic, int number_of_zeros_to_remove, string inputname, string outputname)
- void [HuffmanEncode](#) (string inputname, string outputname, string dicfile)
- void [HuffmanDecode](#) (string inputname, string outputname, string [dictionary](#))

Variables

- `map< char, string >` [dictionary](#)

4.1.1 Function Documentation

4.1.1.1 `binary_to_decimal()`

```
int binary_to_decimal (
    string & in )
```

Funkcja przyjmuje oktet, który zamienia z wartości binarnej na wartość dziesiętną.

Parameters

<i>in</i>	8 bitów zapisane jako string
-----------	------------------------------

Returns

int

Definition at line 62 of file [Helpers.cpp](#).

4.1.1.2 `calculate_probability()`

```
map< char, double > calculate_probability (
    string inputname )
```

Funkcja zapisuje wystąpienia znaków do mapy, aby na końcu podzielić ilość wystąpień znaku przez ilość wszystkich znaków w tekście. Otrzymujemy w ten sposób prawdopodobieństwo wystąpienia znaku.

Parameters

<i>filename</i>	nazwa pliku, z którego chcemy wczytać dane
-----------------	--------------------------------------------

Returns

`map<char, double>` mapa zawierająca znaki z prawdopodobieństwem ich wysąpienia

Definition at line 119 of file [Helpers.cpp](#).

4.1.1.3 coded_message()

```
pair< string, int > coded_message (
    map< char, string > dictionary,
    string inputfile )
```

Funkcja koduje dane z pliku wejściowego używając słownika stworzonego przez [create_dictionary\(\)](#). Po zakodowaniu wszystkich znaków sprawdza czy ilość zer i jedynek jest podzielna przez 8. Jeżeli nie to doklejamy odpowiednią ilość zer na koniec kodu.

Parameters

<i>dictionary</i>	słownik utworzony przez create_dictionary()
<i>filename</i>	nazwa pliku

Returns

pair<string, int> para zawierająca zakodowany tekst oraz ilość zer dodanych do ostatniego oktetu

Definition at line 207 of file [Helpers.cpp](#).

4.1.1.4 compare_by_prob()

```
bool compare_by_prob (
    const WezelDrzewa & a,
    const WezelDrzewa & b )
```

Funkcja pomagająca przy sortowaniu węzłów drzewa znajdujących się w wektorze porównując prawdopodobieństwo.

Parameters

<i>a</i>	pierwsz węzeł drzewa
<i>b</i>	drugi węzeł drzewa

Returns

true zwraca prawdę jeżeli prawdopodobieństwo zawarte w węźle a jest większe
false

See also

[WezelDrzewa](#)

Definition at line 114 of file [Helpers.cpp](#).

4.1.1.5 create_dictionary()

```
void create_dictionary (
    WezelDrzewa * tree,
    map< char, string > & dictionary,
    string prevstring )
```

Struktura [WezelDrzewa](#) jest używana do tworzenia drzewa binarnego. [WezelDrzewa](#) oprócz standardowych wartości, zawiera również wartość `ch` oraz `probability`, które kolejno oznaczają znak, oraz jego prawdopodobieństwo wystąpienia w tekście. Funkcja rekurencyjnie przechodzi po drzewie tworząc kody huffmana dla poszczególnych znaków. Gdy nie ma możliwości przejścia do kolejnych potomków, utworzony kod(w zapisie binarnym) zostaje dodany jako element mapy, wraz ze znakiem, dla którego kod utworzono.

Parameters

<i>tree</i>	gotowe drzewo utworzone przez funkcję create_tree()
<i>dictionary</i>	mapa do której mają zostać zapisane znaki oraz ich kody
<i>prevstring</i>	zmienna pomocnicza, do której podczas rekurencji doklejane są zera i jedynki

See also

[create_tree\(\)](#)

Definition at line 142 of file [Helpers.cpp](#).

4.1.1.6 create_tree()

```
WezelDrzewa create_tree (
    map< char, double > chars_with_prob )
```

Funkcja tworzy drzewo binarne. W pierwszym kroku wartości z mapy utworzonej przez [calculate_probability\(\)](#) zamieniane są na [WezelDrzewa](#) i wkładane do wektora. Następnie wykorzystując węzły zawarte w wektorze zaczynamy budować drzewo. Bierzymy dwa elementy o najmniejszym prawdopodobieństwie i przypisujemy je do nowego [WezelDrzewa](#) jako jego dzieci. Następnie sortujemy wektor korzystając z [compare_by_prob\(\)](#). Te dwa korki powtarzamy, aż w wektorze zostanie nam jeden element, który będzie gotowym drzewem huffmana.

Parameters

<i>chars_with_prob</i>	mapa zawierająca znaki z prawdopodobieństwem wystąpienia w tekście
------------------------	--------------------------------------------------------------------

Returns

[WezelDrzewa](#) gotowe drzewo huffmana

Definition at line 168 of file [Helpers.cpp](#).

4.1.1.7 decimal_to_binary()

```
string decimal_to_binary (
    int in )
```

Funkcja zamienia liczbę z zapisu dziesiętnego na binarny.

Parameters

<i>in</i>	liczba do zamiany
-----------	-------------------

Returns

string zwraca liczbę zapisaną binarnie

Definition at line 70 of file [Helpers.cpp](#).

4.1.1.8 decode_from_binary_to_text()

```
void decode_from_binary_to_text (
    map< string, char > decoDic,
    int number_of_zeros_to_remove,
    string inputname,
    string outputname )
```

Funkcja wczytuje po osiem bitów z pliku binarnego jako char. Następnie przy użyciu [decimal_to_binary\(\)](#) chary zamieniane są na zera i jedynki i doklejone do stringa decodedmessage. W kolejnym kroku przy pomocy słownika decoDic dekodujemy zawartość decoded message i wypisujemy do pliku wyjściowego jako zdekompresowane dane.

Parameters

<i>decoDic</i>	słownik utworzony przez load_dic_file()
<i>number_of_zeros_to_remove</i>	ilość zer do usunięcia z ostatniego oktetu
<i>inputname</i>	nazwa zkompresowanego pliku
<i>outputname</i>	nazwa pliku po dekompresji

Definition at line 278 of file [Helpers.cpp](#).

4.1.1.9 HuffmanDecode()

```
void HuffmanDecode (
    string inputname,
    string outputname,
    string dictionary )
```

Funkcja dekompresująca dane z podanego pliku. Żeby skorzystać z tej funkcji musimy podać nazwę pliku do dekompresji, oraz nazwę pliku zawierającego słownik.

Parameters

<i>inputname</i>	nazwa pliku do dekompresji
<i>outputname</i>	nazwa pliku po dekompresji
<i>dictionary</i>	nazwa pliku zawierająca słownik

Definition at line 327 of file [Helpers.cpp](#).

4.1.1.10 HuffmanEncode()

```
void HuffmanEncode (
    string inputname,
    string outputname,
    string dicfile )
```

Funkcja kompresująca zawartość pliku. Tworzony jest zkompresowany plik, oraz słownik, który jest wymagany do dekompresji.

Parameters

<i>inputname</i>	nazwa pliku do kompresji
<i>outputname</i>	nazwa pliku po kompresji
<i>dicfile</i>	nazwa słownika

Definition at line 314 of file [Helpers.cpp](#).

4.1.1.11 load_dic_file()

```
pair< map< string, char >, int > load_dic_file (
    string dicFile )
```

Funkcja wczytuje pierwszą linię z pliku, w której zapisano ilość zer dodaną do ostatniego oktetu wiadomości. Następnie wczytujemy kolejne linijki zawierające znaki(zapisać jako int) oraz ich odpowiedniki w kodzie huffmana. Te wartości wstawiamy do mapy, która będzie użyta do odkodowania danych.

Parameters

<i>dicFile</i>	nazwa pliku słownika
----------------	----------------------

Returns

pair<map<string, char>, int> para zawierająca gotowy słownik, oraz liczbę zer dodaną do ostatnie oktetu

Definition at line 258 of file [Helpers.cpp](#).

4.1.1.12 output_dictionary()

```
template<typename K , typename V >
void output_dictionary (
    map< K, V > const & m,
    string dicfile,
    int zeros_added )
```

Funkcja wypisująca słownik(mapę) utworzony przez [create_dictionary\(\)](#) do pliku.

Parameters

<i>m</i>	mapa z której pobieramy wartości
<i>filename</i>	nazwa pliku
<i>zeros_added</i>	ilość zer dodana do ostatniego oktetu

Definition at line [156](#) of file [Helpers.cpp](#).

4.1.1.13 output_to_binary()

```
void output_to_binary (
    string huffman_coded_string,
    string outputfile )
```

Funkcja przyjmuje zakodowaną wiadomość jako string. Następnie w pętli będziemy w pętli dzielić tekst na oktety, które zostaną zapisane jako znaki wpliku wyjściowym(.bin). Do zamiany oktetu na char używamy [binary_to_decimal\(\)](#)

Parameters

<i>huffman_coded_string</i>	zakodowane dane wejściowe w postaci zer i jedynek
<i>filename</i>	nazwa pliku, do którego będziemy zapisywać dane w postaci binarnej

Definition at line [234](#) of file [Helpers.cpp](#).

4.1.1.14 setPathByFlag()

```
void setPathByFlag (
    string & flag,
    string & path,
    string & input_path,
    string & output_path,
    string & dic_path,
    string & mode )
```

Definition at line [43](#) of file [Helpers.cpp](#).

4.1.1.15 split_string()

```
pair< string, char > split_string (
    string napis )
```

Funkcja dzieli podany napis na dwie części po napotkaniu znaku spacji.

Parameters

<i>napis</i>	wejściowy string zawierający dwie wartości oddzielone spacją
--------------	--------------------------------------------------------------

Returns

pair<string, char> zwraca dwie wartości: kod huffmana, znak ASCII zapisany jako int

Definition at line 87 of file [Helpers.cpp](#).

4.1.1.16 validate_input_arguments()

```
int validate_input_arguments (
    int argc,
    vector< string > & arg )
```

Definition at line 16 of file [Helpers.cpp](#).

4.1.2 Variable Documentation

4.1.2.1 dictionary

```
map<char, string> dictionary
```

Definition at line 13 of file [Helpers.cpp](#).

4.2 Helpers.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Helpers.h"
00002 #include <iostream>
00003 #include <vector>
00004 #include <map>
00005 #include <utility>
00006 #include <fstream>
00007 #include <bitset>
00008 #include <cstdlib>
00009 #include <string>
00010 #include <algorithm>
00011 using namespace std;
00012
00013 map<char, string> dictionary;
00014
00015
00016 int validate_input_arguments(int argc, vector<string> &arg) {
00017     if (argc == 1 || arg[1] == "-h" || arg[1] == "--help") {
00018         cout << "User manual\n"
00019              << "Usage:\n"
00020              << "  -i(--input) input_filename\n"
00021              << "  -o(--output) output_filename\n"
00022              << "  -s(--slownik) slownik_filename\n"
00023              << "  -t(--tryb) k - kompresja, d - dekompresja\n"
00024              << "All four arguments are necessary to run program. For compression output file type
00025              << "should be .bin and for decompression input should be .bin\n";
00026         return -1;
00027     }
00028     if (argc > 1 && argc != 9) {
00029         cout << argc;
00030         cout << "The number of arguments is not equal 4. Type '-h' and check manual\n";
00031         return -1;
00032     }
00033     if (arg[1] == arg[3]) {
00034         cout << "Can't use the same flag two times. Type '-h' and check manual\n";
00035         return -1;
00036     }
00037     if (arg[2] == arg[4]) {
00038         cout << "Can't use the same file for input and output. Type '-h' and check manual\n";
00039         return -1;
00040     }
00041     return 0;
00042 }
00043 void setPathByFlag(string& flag, string& path, string& input_path, string& output_path, string&
00044 dic_path, string& mode) {
00045     if (flag == "-i" || flag == "--input") {
00046         input_path = path;
00047     }
00048     else if (flag == "-o" || flag == "--output") {
00049         output_path = path;
00050     }
00051     else if (flag == "-s" || flag == "--slownik") {
00052         dic_path = path;
00053     }
00054     else if (flag == "-t" || flag == "--tryb") {
00055         mode = path;
00056     }
00057     else {
00058         cout << "Unknown flag. Type '-h' and check manual\n";
00059     }
00060 }
00061
00062 int binary_to_decimal(string& in)
00063 {
00064     int result = 0;
00065     for (int i = 0; i < (int)in.size(); i++)
00066         result = result * 2 + in[i] - '0';
00067     return result;
00068 }
00069
00070 string decimal_to_binary(int in)
00071 {
00072     string temp = "";
00073     string result = "";
00074     while (in)
00075     {
00076         temp += ('0' + in % 2);
00077         in /= 2;
00078     }
00079     result.append(8 - temp.size(), '0');
00080     for (int i = temp.size() - 1; i >= 0; i--)

```

```

00081     {
00082         result += temp[i];
00083     }
00084     return result;
00085 }
00086
00087 pair<string, char> split_string(string napis)
00088 {
00089     pair<string, char> wynik;
00090     string znak = "";
00091     string kod = "";
00092     bool space = false;
00093     for (int i = 0; i < (int)napis.length(); i++)
00094     {
00095         if (napis[i] == ' ')
00096         {
00097             space = true;
00098             continue;
00099         }
00100         if (!space)
00101         {
00102             znak += napis[i];
00103         }
00104         else
00105         {
00106             kod += napis[i];
00107         }
00108     }
00109     wynik.second = char(stoi(znak));
00110     wynik.first = kod;
00111     return wynik;
00112 }
00113
00114 bool compare_by_prob(const WezelDrzewa& a, const WezelDrzewa& b)
00115 {
00116     return a.probability > b.probability;
00117 }
00118
00119 map<char, double> calculate_probability(string inputname)
00120 {
00121     int number_of_characters = 0;
00122     string line;
00123     map<char, double> dane;
00124     ifstream infile(inputname);
00125     while (getline(infile, line))
00126     {
00127         number_of_characters += line.length();
00128         for (int i = 0; i < (int)line.length(); i++)
00129         {
00130             dane[line[i]]++;
00131         }
00132         dane['\n']++;
00133     }
00134     infile.close();
00135     for (auto& it : dane)
00136     {
00137         dane[it.first] = it.second / number_of_characters;
00138     }
00139     return dane;
00140 }
00141
00142 void create_dictionary(WezelDrzewa* tree, map<char, string>& dictionary, string prevstring)
00143 {
00144     if (tree)
00145     {
00146         if (int(tree->ch[0]) != 0)
00147         {
00148             dictionary.insert(std::pair<char, string>(char(tree->ch[0]), prevstring));
00149         }
00150         create_dictionary(tree->left, dictionary, prevstring + "0");
00151         create_dictionary(tree->right, dictionary, prevstring + "1");
00152     }
00153 }
00154
00155 template <typename K, typename V>
00156 void output_dictionary(map<K, V> const& m, string dicfile, int zeros_added)
00157 {
00158     ofstream plik;
00159     plik.open(dicfile);
00160     plik << zeros_added << '\n';
00161     for (auto const& pair : m)
00162     {
00163         plik << int(pair.first) << " " << pair.second << '\n';
00164     }
00165     plik.close();
00166 }
00167

```

```

00168 WezelDrzewa create_tree(map<char, double> chars_with_prob)
00169 {
00170     vector<WezelDrzewa> elementyDoUtworzeniaDrzewa;
00171     for (auto const& pair : chars_with_prob)
00172     {
00173         WezelDrzewa x;
00174         x.left = NULL;
00175         x.right = NULL;
00176         x.ch = pair.first;
00177         x.probability = pair.second;
00178         elementyDoUtworzeniaDrzewa.push_back(x);
00179     }
00180     //Create tree from nodes
00181     while (elementyDoUtworzeniaDrzewa.size() > 1)
00182     {
00183         WezelDrzewa* node1 = new WezelDrzewa(elementyDoUtworzeniaDrzewa.back());
00184         elementyDoUtworzeniaDrzewa.pop_back();
00185         WezelDrzewa* node2 = new WezelDrzewa(elementyDoUtworzeniaDrzewa.back());
00186         elementyDoUtworzeniaDrzewa.pop_back();
00187
00188         WezelDrzewa x;
00189         x.ch = "";
00190         x.probability = node1->probability + node2->probability;
00191         if (node1->probability < node2->probability)
00192         {
00193             x.left = node1;
00194             x.right = node2;
00195         }
00196         else
00197         {
00198             x.left = node2;
00199             x.right = node1;
00200         }
00201         elementyDoUtworzeniaDrzewa.push_back(x);
00202         sort(elementyDoUtworzeniaDrzewa.begin(), elementyDoUtworzeniaDrzewa.end(), compare_by_prob);
00203     }
00204     return elementyDoUtworzeniaDrzewa.back();
00205 }
00206
00207 pair<string, int> coded_message(map<char, string> dictionary, string inputfile)
00208 {
00209     string huffman_coded_string = "";
00210     ifstream file(inputfile);
00211     string line;
00212     while (getline(file, line))
00213     {
00214         for (int i = 0; i < (int)line.length(); i++)
00215         {
00216             huffman_coded_string += dictionary[line[i]];
00217         }
00218         huffman_coded_string += dictionary['\n'];
00219     }
00220     file.close();
00221
00222     int number_added_zeros = 8 - huffman_coded_string.length() % 8;
00223     if (number_added_zeros != 0)
00224     {
00225         for (int i = 0; i < number_added_zeros; i++)
00226         {
00227             huffman_coded_string += "0";
00228         }
00229     }
00230     pair<string, int> result(huffman_coded_string, number_added_zeros);
00231     return result;
00232 }
00233
00234 void output_to_binary(string huffman_coded_string, string outputfile)
00235 {
00236     string in = "";
00237     string part_of_string = "";
00238     ofstream compressed(outputfile, ios::binary);
00239     for (int i = 1; i <= (int)huffman_coded_string.length(); i++)
00240     {
00241         if (part_of_string.length() == 7)
00242         {
00243             part_of_string += huffman_coded_string[i - 1];
00244             in += (char)binary_to_decimal(part_of_string);
00245             part_of_string = "";
00246         }
00247         else
00248         {
00249             part_of_string += huffman_coded_string[i - 1];
00250         }
00251     }
00252     compressed.write(in.c_str(), in.size());
00253     compressed.close();
00254 }

```

```

00255
00256
00257 // Decoding
00258 pair<map<string, char>, int> load_dic_file(string dicFile)
00259 {
00260     map<string, char> decoDic;
00261     string line, number = "";
00262
00263     ifstream dict(dicFile);
00264     getline(dict, line);
00265     number.push_back(line[0]);
00266
00267     int number_of_zeros_to_remove = stoi(number);
00268
00269     while (getline(dict, line))
00270     {
00271         decoDic.insert(pair<string, char>(split_string(line).first, split_string(line).second));
00272     }
00273     dict.close();
00274     pair<map<string, char>, int> res(decoDic, number_of_zeros_to_remove);
00275     return res;
00276 }
00277
00278 void decode_from_binary_to_text(map<string, char> decoDic, int number_of_zeros_to_remove, string
    inputname, string outputname)
00279 {
00280
00281     ifstream codedmessage(inputname, ifstream::in | ios::binary);
00282     string decodedmessage = "";
00283     vector<unsigned char> text;
00284     unsigned char textseg;
00285     codedmessage.read(reinterpret_cast<char*>(&textseg), 1);
00286     while (!codedmessage.eof())
00287     {
00288         text.push_back(textseg);
00289         codedmessage.read(reinterpret_cast<char*>(&textseg), 1);
00290     }
00291     codedmessage.close();
00292     for (int i = 0; i < (int)text.size(); i++)
00293     {
00294         decodedmessage += decimal_to_binary(text[i]);
00295     }
00296
00297     ofstream decoded_message_file(outputname);
00298     string ciag = "";
00299     map<string, char>::iterator it;
00300     for (int i = 0; i < (int)decodedmessage.length() - number_of_zeros_to_remove; i++)
00301     {
00302         ciag += decodedmessage[i];
00303         it = decoDic.find(ciag);
00304         if (it != decoDic.end())
00305         {
00306             decoded_message_file << it->second;
00307             ciag = "";
00308         }
00309     }
00310     decoded_message_file.close();
00311
00312 }
00313
00314 void HuffmanEncode(string inputname, string outputname, string dicfile)
00315 {
00316     map<char, double> chars_with_prob = calculate_probability(inputname);
00317     WezelDrzewa* prepared_tree = new WezelDrzewa(create_tree(chars_with_prob));
00318     map<char, string> dictionary;
00319
00320     create_dictionary(prepared_tree, dictionary, "");
00321     pair<string, int> data_for_output = coded_message(dictionary, inputname);
00322
00323     output_to_binary(data_for_output.first, outputname);
00324     output_dictionary(dictionary, dicfile, data_for_output.second);
00325 }
00326
00327 void HuffmanDecode(string inputname, string outputname, string dictionary)
00328 {
00329     pair<map<string, char>, int> dictionary_data = load_dic_file(dictionary);
00330     decode_from_binary_to_text(dictionary_data.first, dictionary_data.second, inputname, outputname);
00331 }

```

4.3 Helpers.h File Reference

```
#include <iostream>
#include <vector>
#include <map>
#include <utility>
#include <fstream>
#include <bitset>
#include <cstdlib>
#include <string>
```

Classes

- struct [WezelDrzewa](#)

Struktura używana przy tworzeniu drzewa, przechowuje znak oraz prawdopodobieństwo jego wystąpienia w tekście.

Functions

- int [validate_input_arguments](#) (int argc, vector< string > &arg)
- void [setPathByFlag](#) (string &flag, string &path, string &input_path, string &output_path, string &dic_path, string &mode)
- int [binary_to_decimal](#) (string &in)
- string [decimal_to_binary](#) (int in)
- pair< string, char > [split_string](#) (string napis)
- bool [compare_by_prob](#) (const [WezelDrzewa](#) &a, const [WezelDrzewa](#) &b)
- map< char, double > [calculate_probability](#) (string inputname)
- void [create_dictionary](#) ([WezelDrzewa](#) *tree, map< char, string > &[dictionary](#), string prevstring)
- template<typename K , typename V >
void [output_dictionary](#) (map< K, V > const &m, string dicfile, int zeros_added)
- [WezelDrzewa](#) [create_tree](#) (map< char, double > chars_with_prob)
- pair< string, int > [coded_message](#) (map< char, string > [dictionary](#), string inputfile)
- void [output_to_binary](#) (string huffman_coded_string, string outputfile)
- pair< map< string, char >, int > [load_dic_file](#) (string dicFile)
- void [decode_from_binary_to_text](#) (map< string, char > decoDic, int number_of_zeros_to_remove, string inputname, string outputname)
- void [HuffmanEncode](#) (string inputname, string outputname, string dicfile)
- void [HuffmanDecode](#) (string inputname, string outputname, string [dictionary](#))

4.3.1 Function Documentation

4.3.1.1 [binary_to_decimal\(\)](#)

```
int binary_to_decimal (
    string & in )
```

Funkcja przyjmuje oktet, który zamienia z wartości binarnej na wartość dziesiętną.

Parameters

<i>in</i>	8 bitów zapisane jako string
-----------	------------------------------

Returns

int

Definition at line 62 of file [Helpers.cpp](#).

4.3.1.2 calculate_probability()

```
map< char, double > calculate_probability (
    string inputname )
```

Funkcja zapisuje wystąpienia znaków do mapy, aby na końcu podzielić ilość wystąpień znaku przez ilość wszystkich znaków w tekście. Otrzymujemy w ten sposób prawdopodobieństwo wystąpienia znaku.

Parameters

<i>filename</i>	nazwa pliku, z którego chcemy wczytać dane
-----------------	--------------------------------------------

Returns

map<char, double> mapa zawierająca znaki z prawdopodobieństwem ich wystąpienia

Definition at line 119 of file [Helpers.cpp](#).

4.3.1.3 coded_message()

```
pair< string, int > coded_message (
    map< char, string > dictionary,
    string inputfile )
```

Funkcja koduje dane z pliku wejściowego używając słownika stworzonego przez [create_dictionary\(\)](#). Po zakodowaniu wszystkich znaków sprawdza czy ilość zer i jedynek jest podzielna przez 8. Jeżeli nie to doklejamy odpowiednią ilość zer na koniec kodu.

Parameters

<i>dictionary</i>	słownik utworzony przez create_dictionary()
<i>filename</i>	nazwa pliku

Returns

`pair<string, int>` para zawierająca zakodowany tekst oraz ilość zer dodanych do ostatniego oktetu

Definition at line 207 of file [Helpers.cpp](#).

4.3.1.4 compare_by_prob()

```
bool compare_by_prob (
    const WezelDrzewa & a,
    const WezelDrzewa & b )
```

Funkcja pomagająca przy sortowaniu węzłów drzewa znajdujących się w wektorze porównując prawdopodobieństwo.

Parameters

<i>a</i>	pierwsz węzeł drzewa
<i>b</i>	drugi węzeł drzewa

Returns

`true` zwraca prawdę jeżeli prawdopodobieństwo zawarte w węźle *a* jest większe
`false`

See also

[WezelDrzewa](#)

Definition at line 114 of file [Helpers.cpp](#).

4.3.1.5 create_dictionary()

```
void create_dictionary (
    WezelDrzewa * tree,
    map< char, string > & dictionary,
    string prevstring )
```

Struktura [WezelDrzewa](#) jest używana do tworzenia drzewa binarnego. [WezelDrzewa](#) oprócz standardowych wartości, zawiera również wartość `ch` oraz `probability`, które kolejno oznaczają znak, oraz jego prawdopodobieństwo wystąpienia w tekście. Funkcja rekurencyjnie przechodzi po drzewie tworząc kody huffmana dla poszczególnych znaków. Gdy nie ma możliwości przejścia do kolejnych potomków, utworzony kod (w zapisie binarnym) zostaje dodany jako element mapy, wraz ze znakiem, dla którego kod utworzono.

Parameters

<i>tree</i>	gotowe drzewo utworzone przez funkcję create_tree()
<i>dictionary</i>	mapa do której mają zostać zapisane znaki oraz ich kody
<i>prevstring</i>	zmienna pomocnicza, do której podczas rekurencji doklejane są zera i jedynki

See also

[create_tree\(\)](#)

Definition at line 142 of file [Helpers.cpp](#).

4.3.1.6 create_tree()

```
WezelDrzewa create_tree (
    map< char, double > chars_with_prob )
```

Funkcja tworzy drzewo binarne. W pierwszym kroku wartości z mapy utworzonej przez [calculate_probability\(\)](#) zamieniane są na [WezelDrzewa](#) i wkładane do wektora. Następnie wykorzystując węzły zawarte w wektorze zaczynamy budować drzewo. Bierzymy dwa elementy o najmniejszym prawdopodobieństwie i przypisujemy je do nowego [WezelDrzewa](#) jako jego dzieci. Następnie sortujemy wektor korzystając z [compare_by_prob\(\)](#). Te dwa korki powtarzamy, aż w wektorze zostanie nam jeden element, który będzie gotowym drzewem huffmana.

Parameters

<i>chars_with_prob</i>	mapa zawierająca znaki z prawdopodobieństwem wystąpienia w tekście
------------------------	--------------------------------------------------------------------

Returns

[WezelDrzewa](#) gotowe drzewo huffmana

Definition at line 168 of file [Helpers.cpp](#).

4.3.1.7 decimal_to_binary()

```
string decimal_to_binary (
    int in )
```

Funkcja zamienia liczbę z zapisu dziesiętnego na binarny.

Parameters

<i>in</i>	liczba do zamiany
-----------	-------------------

Returns

string zwraca liczbę zapisaną binarnie

Definition at line 70 of file [Helpers.cpp](#).

4.3.1.8 decode_from_binary_to_text()

```
void decode_from_binary_to_text (
    map< string, char > decoDic,
    int number_of_zeros_to_remove,
    string inputname,
    string outputname )
```

Funkcja wczytuje po osiem bitów z pliku binarnego jako char. Następnie przy użyciu [decimal_to_binary\(\)](#) chary zamieniane są na zera i jedynki i doklejone do stringa decodedmessage. W kolejnym kroku przy pomocy słownika decoDic dekodujemy zawartość decoded message i wypisujemy do pliku wyjściowego jako zdekompresowane dane.

Parameters

<i>decoDic</i>	słownik utworzony przez load_dic_file()
<i>number_of_zeros_to_remove</i>	ilość zer do usunięcia z ostatniego oktetu
<i>inputname</i>	nazwa zkompresowanego pliku
<i>outputname</i>	nazwa pliku po dekompresji

Definition at line 278 of file [Helpers.cpp](#).

4.3.1.9 HuffmanDecode()

```
void HuffmanDecode (
    string inputname,
    string outputname,
    string dictionary )
```

Funkcja dekompresująca dane z podanego pliku. Żeby skorzystać z tej funkcji musimy podać nazwę pliku do dekompresji, oraz nazwę pliku zawierającego słownik.

Parameters

<i>inputname</i>	nazwa pliku do dekompresji
<i>outputname</i>	nazwa pliku po dekompresji
<i>dictionary</i>	nazwa pliku zawierająca słownik

Definition at line 327 of file [Helpers.cpp](#).

4.3.1.10 HuffmanEncode()

```
void HuffmanEncode (
    string inputname,
    string outputname,
    string dicfile )
```

Funkcja kompresująca zawartość pliku. Tworzony jest zkompresowany plik, oraz słownik, który jest wymagany do dekompresji.

Parameters

<i>inputname</i>	nazwa pliku do kompresji
<i>outputname</i>	nazwa pliku po kompresji
<i>dicfile</i>	nazwa słownika

Definition at line 314 of file [Helpers.cpp](#).

4.3.1.11 load_dic_file()

```
pair< map< string, char >, int > load_dic_file (
    string dicFile )
```

Funkcja wczytuje pierwszą linijkę z pliku, w której zapisano ilość zer dodaną do ostatniego oktetu wiadomości. Następnie wczytujemy kolejne linijki zawierające znaki(zapisać jako int) oraz ich odpowiedniki w kodzie huffmana. Te wartości wstawiamy do mapy, która będzie użyta do odkodowania danych.

Parameters

<i>dicFile</i>	nazwa pliku słownika
----------------	----------------------

Returns

pair<map<string, char>, int> para zawierająca gotowy słownik, oraz liczbę zer dodaną do ostatniego oktetu

Definition at line 258 of file [Helpers.cpp](#).

4.3.1.12 output_dictionary()

```
template<typename K , typename V >
void output_dictionary (
    map< K, V > const & m,
    string dicfile,
    int zeros_added )
```

Funkcja wypisująca słownik(mapę) utworzony przez [create_dictionary\(\)](#) do pliku.

Parameters

<i>m</i>	mapa z której pobieramy wartości
<i>filename</i>	nazwa pliku
<i>zeros_added</i>	ilość zer dodana do ostatniego oktetu

Definition at line 156 of file [Helpers.cpp](#).

4.3.1.13 output_to_binary()

```
void output_to_binary (
    string huffman_coded_string,
    string outputfile )
```

Funkcja przyjmuje zakodowaną wiadomość jako string. Następnie w pętli będziemy w pętli dzielić tekst na oktety, które zostaną zapisane jako znaki wpliku wyjściowym(.bin). Do zamiany oktetu na char używamy [binary_to_decimal\(\)](#)

Parameters

<i>huffman_coded_string</i>	zakodowane dane wejściowe w postaci zer i jedynek
<i>filename</i>	nazwa pliku, do którego będziemy zapisywać dane w postaci binarnej

Definition at line 234 of file [Helpers.cpp](#).

4.3.1.14 setPathByFlag()

```
void setPathByFlag (
    string & flag,
    string & path,
    string & input_path,
    string & output_path,
    string & dic_path,
    string & mode )
```

Definition at line 43 of file [Helpers.cpp](#).

4.3.1.15 split_string()

```
pair< string, char > split_string (
    string napis )
```

Funkcja dzieli podany napis na dwie części po napotkaniu znaku spacji.

Parameters

<i>napis</i>	wejściowy string zawierający dwie wartości oddzielone spacją
--------------	--------------------------------------------------------------

Returns

pair<string, char> zwraca dwie wartości: kod huffmana, znak ASCII zapisany jako int

Definition at line 87 of file [Helpers.cpp](#).

4.3.1.16 validate_input_arguments()

```
int validate_input_arguments (
    int argc,
    vector< string > & arg )
```

Definition at line 16 of file [Helpers.cpp](#).

4.4 Helpers.h

[Go to the documentation of this file.](#)

```
00001 #ifndef  HELPERS_H
00002 #define  HELPERS_H
00003 #endif // ! HELPERS_H
00004
00005 #include <iostream>
00006 #include <vector>
00007 #include <map>
00008 #include <utility>
00009 #include <fstream>
00010 #include <bitset>
00011 #include <cstdlib>
00012 #include <string>
00013 using namespace std;
00014
00015 int validate_input_arguments(int argc, vector<string> &arg);
00016
00017 void setPathByFlag(string& flag, string& path, string& input_path, string& output_path, string&
    dic_path, string& mode);
00018
00023 struct WezelDrzewa
00024 {
00029     struct WezelDrzewa* parent = NULL;
00034     struct WezelDrzewa * left = NULL;
00039     struct WezelDrzewa * right = NULL;
00044     string ch;
00045     double probability = NULL;
00046 };
00047
00054 int binary_to_decimal(string& in);
00055
00062 string decimal_to_binary(int in);
00069 pair<string, char> split_string(string napis);
00079 bool compare_by_prob(const WezelDrzewa& a, const WezelDrzewa& b);
00086 map<char, double> calculate_probability(string inputname);
00087
00102 void create_dictionary(WezelDrzewa* tree, map<char, string>& dictionary, string prevstring);
00110 template <typename K, typename V>
00111 void output_dictionary(map<K, V> const& m, string dicfile, int zeros_added);
00118 WezelDrzewa create_tree(map<char, double> chars_with_prob);
00126 pair<string, int> coded_message(map<char, string> dictionary, string inputfile);
00133 void output_to_binary(string huffman_coded_string, string outputfile);
00140 pair<map<string, char>, int> load_dic_file(string dicFile);
00149 void decode_from_binary_to_text(map<string, char> decoDic, int number_of_zeros_to_remove, string
    inputname, string outputname);
00150
00158 void HuffmanEncode(string inputname, string outputname, string dicfile);
00166 void HuffmanDecode(string inputname, string outputname, string dictionary);
```

4.5 huffman-coding.cpp File Reference

```
#include "Helpers.h"
#include <iostream>
```

Functions

- int [main](#) (int argc, char **argv)

4.5.1 Function Documentation

4.5.1.1 main()

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

Definition at line 5 of file [huffman-coding.cpp](#).

4.6 huffman-coding.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Helpers.h"
00002 #include <iostream>
00003 using namespace std;
00004
00005 int main(int argc, char** argv)
00006 {
00007     string input_path, output_path, dic_path, mode;
00008     vector<string> arg(argv, argv + argc);
00009
00010
00011     if (validate_input_arguments(argc, arg) == -1) {
00012         return 0;
00013     }
00014
00015     setPathByFlag(arg[1], arg[2], input_path, output_path, dic_path, mode);
00016     setPathByFlag(arg[3], arg[4], input_path, output_path, dic_path, mode);
00017     setPathByFlag(arg[5], arg[6], input_path, output_path, dic_path, mode);
00018     setPathByFlag(arg[7], arg[8], input_path, output_path, dic_path, mode);
00019
00020
00021     if (mode == "k")
00022     {
00023         HuffmanEncode(input_path, output_path, dic_path);
00024     }
00025     else if(mode == "d")
00026     {
00027         HuffmanDecode(input_path, output_path, dic_path);
00028     }
00029     else
00030     {
00031         cout << "===== " <<
endl;
00032         cout << "Use correct format" << endl;
00033         cout << "For compressing data use:" << endl;
00034         cout << "huffman-coding -i input_filename -o output_filename -s dictionary_filename -t k" << endl;
00035         cout << "For decompressing data use:" << endl;
00036         cout << "huffman-coding -i input_filename -o output_filename -s dictionary_filename -t d" << endl;
00037         cout << "===== " <<
endl;
00038     }
00039     return 1;
00040 }
```

