

Алгоритъм на Хъфман

Изготвил: Алкан Мустафа, фн: 72018, Информационни
Системи, 3 курс, 1 група

Алгоритъмът на Хъфман е сравнително прост универсален алгоритъм за компресия без загуба на данни. Базира се на простата идея, че най-често срещаните символи в поредицата трябва да се записват с възможно най-малък брой битове. Така той построява нова азбука, която следва тази идея и след това превежда информацията в новата азбука. Кодирането е обратимо, т.е кодираната последователност може да се декомпресира – да се намери първоначалната поредица.

Реализацията на алгоритъма се състои от следните класове: `class HTree`, `class Comparator`, `class Encoder`, `class Decoder`, `class System`, `class BitSetHelper`(използва се само на едно място) и `class Menu`.

1. Class HTree

Класът HTree се състои от следните член-данни и методи:

```
C HTree.h M X  C HTree.cpp M
C HTree.h > ...
1  #pragma once
2  #include <iostream>
3  #include <queue>
4
5
6  struct Node{
7      char data;
8      size_t cnt;
9      Node* left;
10     Node* right;
11     Node(char newData, size_t newCnt) : data(newData), cnt(newCnt), left(nullptr), right(nullptr) {}
12 };
13
14 class HTree{
15     private:
16         Node* root;
17
18         void copy(Node*& current, Node* other); // used in copy constructors and operator=
19         void clear(Node*& current); // used in destructor
20         bool isLeaf(Node* current) const; // checks if current node is leaf
21         std::string getCharacterCodeHelper(Node* current, char searched, std::string encode); // gets the coding of entered character
22
23     public:
24         HTree();
25         HTree(char newData, size_t newCnt);
26         HTree(const HTree& first, const HTree& second);
27         HTree(const HTree& other);
28         HTree operator=(const HTree& other);
29         ~HTree();
30
31         size_t getCnt() const;
32
33         std::string getCharacterCode(char searched);
34
35         void printByLevels() const;
36     };

```

Има една единствена член-данна: указател към корена на дървото. Класът има дефолтен конструктор, конструктор с параметри, конструктор за копиране с два параметъра (два други HTree) и съответно с един параметър (друг HTree), оператор= и деструктор. Методът copy(Node*& current, Node* other) се използва в копи конструкторите и в оператор=, а методът clear(Node*& current) се използва в деструктора. getCnt()const връща общия брой на срещанията на всички букви в съответното дърво (ако root == nullptr, връща 0). getCharacterCode(char searched) връща в кодиран вид търсения символ (ако дървото не съдържа търсения символ, се връща празен низ), getCharacterCodeHelper метода се използва тук. printByLevels()const печата възлите на дървото по нива. Методът isLeaf(Node* current)const се използва в методите getCharacterCodeHelper и printByLevels()const, проверява дали подадения възел е листо.

2. Class Comparator

```

C Comparator.h X
C Comparator.h > ...
1  #pragma once
2  #include "HTree.h"
3
4  class Comparator{
5      public:
6          // needed to construct minHeap with Huffman trees
7          bool operator()(const HTree& firstTree, const HTree& secondTree){
8              return firstTree.getCnt() > secondTree.getCnt();
9          }
10 };

```

Използва се, за да може да се построи приоритетна опашка от HTree.

3. Class Encoder

Състои се от следните член-данни и методи:

```

C Encoder.h M X
C Encoder.h > ...
1  #pragma once
2  #include <string>
3  #include <unordered_map>
4  #include <queue>
5  #include <vector>
6  #include <cassert>
7  #include "HTree.h"
8  #include "Comparator.h"
9
10 class Encoder{
11     private:
12         size_t occurrences[128] = {0}; // 0 - 127 codes for every ASCII char
13         std::priority_queue<HTree, std::vector<HTree>, Comparator> queue;
14         HTree huffmanTree;
15         std::unordered_map<std::string, char> keyTable;
16
17         void constructHistogram(const std::string& message);
18         void clearHistogram();
19         void constructHuffmanTree();
20         void constructKeyTable();
21
22     public:
23         Encoder() = default;
24
25         std::string encodeMessage(const std::string& message);
26         std::unordered_map<std::string, char> getKeyTable();
27 };

```

Член-данните са: масив с размер 128(за символите от ASCII таблицата от 0-127, използва се за честотна таблица), приоритетна опашка(за построяване на дърво на Хъфман), дърво на хъфман(крайното дърво) и таблицата за кодиране. Методите constructHistogram построява честотната таблица, а методът clearHistogram я занулява(след края на целия процес). ConstructHuffmanTree() и constructKeyTable() съответно построяват дървото на Хъфман и таблицата за кодиране. Всички тези методи се изпълват в encodeMessage(const std::string& message), който връща като резултат кодираното съобщение, а методът getKeyTable() връща таблицата за кодиране.

4. Class Decoder

Класът се състои от следните член-данни и методи:

```
C Decoder.h x
C Decoder.h > ...
1  #pragma once
2  #include <unordered_map>
3  #include <string>
4
5  class Decoder{
6  private:
7      std::unordered_map<std::string, char> keyTable;
8
9  public:
10     Decoder() = default;
11
12     void setKeyTable(const std::unordered_map<std::string, char>& newKeyTable){
13         if(newKeyTable.empty()){
14             // if empty key table is given, exception is thrown
15             throw std::invalid_argument("Empty Key Table");
16         }
17         keyTable = newKeyTable;
18     }
19
20     std::string decodeMessage(const std::string& message){
21         if(message.empty()){
22             return "";
23         }
24         std::string result = "";
25         std::string code = "";
26         // iterating through the given message
27         for(char character : message){
28             // adding the current character to a string
29             code.push_back(character);
30             if(keyTable.find(code) != keyTable.end()){ // if the code is found in the key table
31                 result.push_back(keyTable[code]); // the result char is added to the string
32                 code.clear();
33             }
34         }
35         return result;
36     }
37 };
```

Класът има една единствена член-данна: keyTable(текущата таблица за кодиране). Методът setKeyTable присвоява на keyTable новата стойност от newKeyTable(ако подадената таблица за кодиране е празна, се хвърля изключение). Методът decodeMessage декодира подаденото съобщение въз основа на текущата таблица за кодиране(ако подаденото съобщение е празно се връща празен низ).

5. Class System

Класът се състои от следните член-данни и методи:

```

C System.h x
C System.h > ...
1  #pragma once
2  #include <fstream>
3  #include <cassert>
4  #include <bitset>
5  #include "Encoder.h"
6  #include "Decoder.h"
7  #include "BitSetHelper.h"
8
9
10 class System{
11     private:
12         Encoder encode;
13         Decoder decode;
14         std::string inputFileName;
15         std::string outputFileName;
16         std::string message;
17         std::unordered_map<std::string, char> keyTable;
18         short workMode = -1; // default value for work mode
19         bool workWithBits = false; // default value to work with bits
20
21         void clearMessageAndTable(); // clears the current message and the current key table
22         bool doesExist(const std::string& fileName); // check if a given file name exists
23         void checkOpenInputFile(std::ifstream& file, const std::string& fileName); // check if given ifstream file is opened successfully
24         void checkOpenOutputFile(std::ofstream& file, const std::string& fileName); // check if given ofstream file is opened successfully
25         void readFile(std::ifstream& file); // reads the whole file of the given ifstream file (the file is opened successfully)
26         void readKeyTable(std::ifstream& file, std::string& data); // reads the key table from a given key table, with the help of additional
27         void writeToFileCompress(); // writing to file in compression mode(standard compression)
28         void readFromFileCompress(); // reading from file in compression mode
29         void writeToFileDecompress(); // writing to file in decompression mode(standard decompression)
30         void readFromFileDecompress(); // reading from file in decompression mode
31         void writeToBits(); // writing to file in compression mode(in bits mode)(used only in doctest)
32         void readFromBits(); // reading from file in decompression mode(in bits mode)(used only in doctest)
33         void printToScreen(); // printing the compressed(encoded) message to the screen(also in bits mode)
34
35         void compress(); // starting the compression mode
36         void decompress(); // starting the decompression mode
37         void debug(); // starting the debug mode(prints the result to screen)
38
39     public:
40         System() = default;
41
42         void setWorkWithBits(bool bitsMode); // setting if the program should work in bits mode
43         bool getWorkWithBits()const; // getting if the program works in bits mode
44         void setMode(short mode); // setting the workmode of the program(compression, decompression, debug)
45         int getMode()const; // getting the current workmode
46         void setInputFileName(const std::string& input); // setting the input file name
47         const std::string& getInputFileName()const; // getting the input file name
48         void setOutputFileName(const std::string& output); // setting the output file name
49         const std::string& getOutputFileName()const; // getting the output file name
50
51         void start(); // starting the program(throws exception if no work mode is set)
52     };

```

Член данните на класа са: енкодер(за енкодиране на текущото съобщение), декоред(за декодиране на текущото съобщение, със съответната таблица за кодиране), два низа(един за името на входния файл и един за името на изходния файл, с които програмата ще работи), таблица за кодиране, една променлива за режим на работа(0 – за компресия, 1 – за декомпресия, 2 – в debug режим и -1 – за не настроен режим) и една булева променлива за това дали ще се работи с битове или стандартна компресия/декомпресия(използва се само в доктестовите). Методът clearMessageAndTable() изтрива текущото съобщение и текущата таблица за кодиране(извиква се след като програмата приключи работа или ако изникне проблем), doesExist(const std::string& fileName) проверява дали съществува файл с такова име, checkOpenInputFile и checkOpenOutputFile проверяват дали файловете

в са отворени успешно в съответните режими. Методът `readFile` чете целия файл символ по символ и го запазва в член-данната `message`, а методът `readKeyTable` чете файла и запазва резултата в член-данната `keyTable`. `WriteToFileCompress()`, `readFromFileCompress`, пишат във файл и четат от файл в режим на компресия. Съответно `writeToFileDecompress()` и `readFromFileDecompress()` пишат във файл и четат от файл в режим на декомпресия. Методите `writeToBits()` и `readFromBits()` пишат във файл и четат от файл в режим на компресия/декомпресия, но по малко по различен начин. `PrintToScreen()` печата резултата на екрана(използва се в debug режима). Методът `compress()` извиква `readFromFileCompress()` и `writeToFileCompress()` или `writeToBits()` в зависимост от стойността на член-данната `workWithBits`. `Decompress()` извиква `readFromFileDecompress()` или `readFromBits()` пак в зависимост от стойността на член-данната и `writeToFileDecompress()`. `Debug()` извиква методите `readFromFileCompress()` и `printToScreen()`, т.е чете файла в режим на компресия и печата компресирания резултат на екрана, както и степента на компресия. В методите `compress()`, `decompress()` и `debug()` се хвърлят изключения и съответно се хващат изключения, ако методите стигнат своя край стойностите `message` и `keyTable` се изтриват(ако се хвърли изключение, също се изтриват). Следващите методи са `setters & getters` съответно за `workWithBits`, за `workMode`, за `InputFileName`, за `OutputFileName`. Последният метод `start()` стартира програмата в зависимост от това какъв режим е избран, ако не е избран никакъв режим(`workMode == -1`), тогава се хвърля изключение.

6. Class Menu

Класът Menu се състои от следните член-данни и методи:

```
C Menu.h X
C Menu.h > ...
1  #pragma once
2  #include "System.h"
3
4
5  class Menu{
6  private:
7      System system;
8      std::string currentCommand;
9      std::string commandNames[10]; // static array of strings for
10     std::string commandExplanations[10]; // the supported 10 commands with their explanations
11
12     void loadCommands(); // loads the commandNames array and the commandExplanations array
13     void printCommands(); // print the supported commands with their explanations
14     // validating the entered command
15     void validateCommand(short& choice, std::string& keyword, std::string& secondPart);
16     // and entering the command cases to determine which command to call
17     void commandCases();
18
19     // setting the current working mode
20     void setCompressionMode();
21     void setDecompressionMode();
22     void setDebugMode();
23     // setting the current files to work with
24     void setInputFile(const std::string& input);
25     void setOutputFile(const std::string& output);
26     // getting the current working mode
27     void mode();
28     // starting the program
29     void start();
30     // getting the current files the problem works with
31     void files();
32
33 public:
34     Menu() = default;
35
36     // starts the whole program
37     void run();
38 }
```

Член-данните са: system, където е логиката за работата с файлове и като цяло за програмата, currentCommand за текущата въведена команда, commandNames и commandExplanations са масиви с размер 10, които пазят в себе си командите и техните обяснения, които програмата поддържа. Методите loadCommands() и printCommands() съответно зареждат имената на командите и техните обяснения в съответните масиви. ValidateCommand валидира текущата въведена команда, а после в commandCases() се решава кой метод да се извика. SetCompressionMode(), setDecompressionMode() и setDebugMode() съответно дават стойности на член-данната workMode на system(съответно 0, 1, 2). SetInputFile, setOutputFile дават стойности на inputFileName и outputFileName на system. Mode() и files() печатат съответно в какъв режим работи програмата и как се казва входния и изходния файл на програмата. Start() извиква методът start() на system. Единствения публичен метод run() стартира цялото CLI приложение.

7. Стартиране на приложението

```
Supported commands :
c[ompress] - parameter showing that the program will work in compression mode
d[ecompress] - parameter showing that the program will work in decompression mode
debug - parameter showing that the program will work in debug mode(prints the result)
i <filename> - parameter showing the input file name the program will work with
o <filename> - parameter showing the output file name the program will work with
m[ode] - prints the current working mode of the program
start - starts the program with current mode and input and output files
files - prints the input file name and output file the that the program is working with
help - prints again all commands
exit - ends the program
Enter command :
```

След стартиране на приложението ни се печатат всички команди, които приложението поддържа(само веднъж се печатат освен ако изрично не се въведе командата help, която служи точно за повторно печатане на всички команди).Като въведем съответно „с“, „d“ или „debug“ поставяме приложението в режим на работа

```
Enter command : c
Mode : Compression mode
Enter command : d
Mode : Decompression mode
Enter command : debug
Mode : Debug mode
Enter command :
```

С командите „i (име на файл)“ и „o (име на файл)“ избираме имената на файловете, с които ще работи нашата програма

```
Enter command : i test.txt
Input file set
Enter command : o test-result.txt
Output file set
Enter command :
```

Съответно командите „m“ и „files“ печатат в какъв режим и с кои файлове работи програмата

```
Enter command : m
Mode : Compression mode
Enter command : files
Input file : test.txt
Output file : test-result.txt
Enter command : █
```


Командата „help“ печата повторно всички команди

```
Enter command : help
Supported commands :
c[ompress] - parameter showing that the program will work in compression mode
d[ecompress] - parameter showing that the program will work in decompression mode
debug - parameter showing that the program will work in debug mode(prints the result)
i <filename> - parameter showing the input file name the program will work with
o <filename> - parameter showing the output file name the program will work with
m[ode] - prints the current working mode of the program
start - starts the program with current mode and input and output files
files - prints the input file name and output file the that the program is working with
help - prints again all commands
exit - ends the program
Enter command : █
```

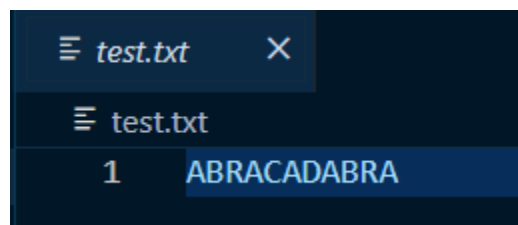
Последната команда „exit“ прекратява изпълнението на програмата

```
Exit - ends the program
Enter command : exit
Exiting
END
```

8. Примерна работа в режим Компресия

```
Enter command : c
Mode : Compression mode
Enter command : i test.txt
Input file set
Enter command : o test-result.txt
Output file set
Enter command : start
Enter command : █
```

Слагаме в режим компресия, въвеждаме имената на файловете и въвеждаме старт. Това е съобщението на входния файл:



The screenshot shows a file editor window with a title bar containing a menu icon, the filename 'test.txt', and a close button. The editor area displays the text 'ABRACADABRA' on a single line, which is highlighted in blue. A line number '1' is visible in the left margin.

Това е енцирираното съобщение в изходния файл:

```
test-result.txt X
test-result.txt
1 01101110100010101101110
```

И това е таблицата за кодиране в другия създаден изходен файл:

```
test-result.txttable.txt M X
test-result.txttable.txt
1 101=D
2 111=R
3 100=C
4 0=A
5 110=B
6
```

9. Примерна работа в режим Декомпресия

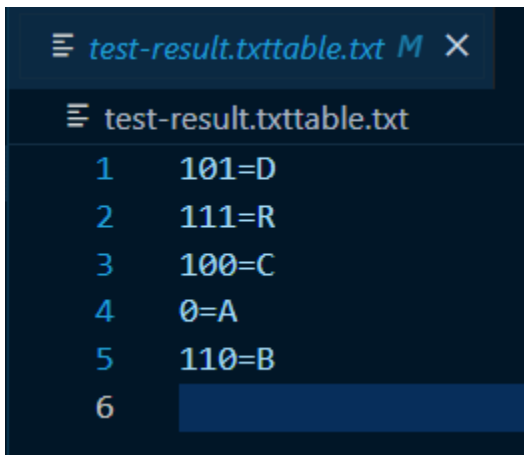
```
Enter command : d
Mode : Decompression mode
Enter command : i test-result.txt
Input file set
Enter command : o test.txt
Output file set
Enter command : start
Enter command :
```

Слагаме програмата в режим декомпресия, въвеждаме име на входния файл и на изходния файл и пишем старт.

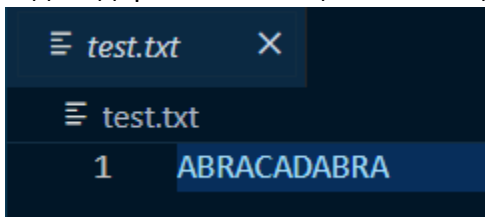
Енцирираното съобщение във входния файл:

```
test-result.txt X
test-result.txt
1 01101110100010101101110
```

И таблицата за кодиране на входния файл:



И декодираното съобщение в изходния файл:



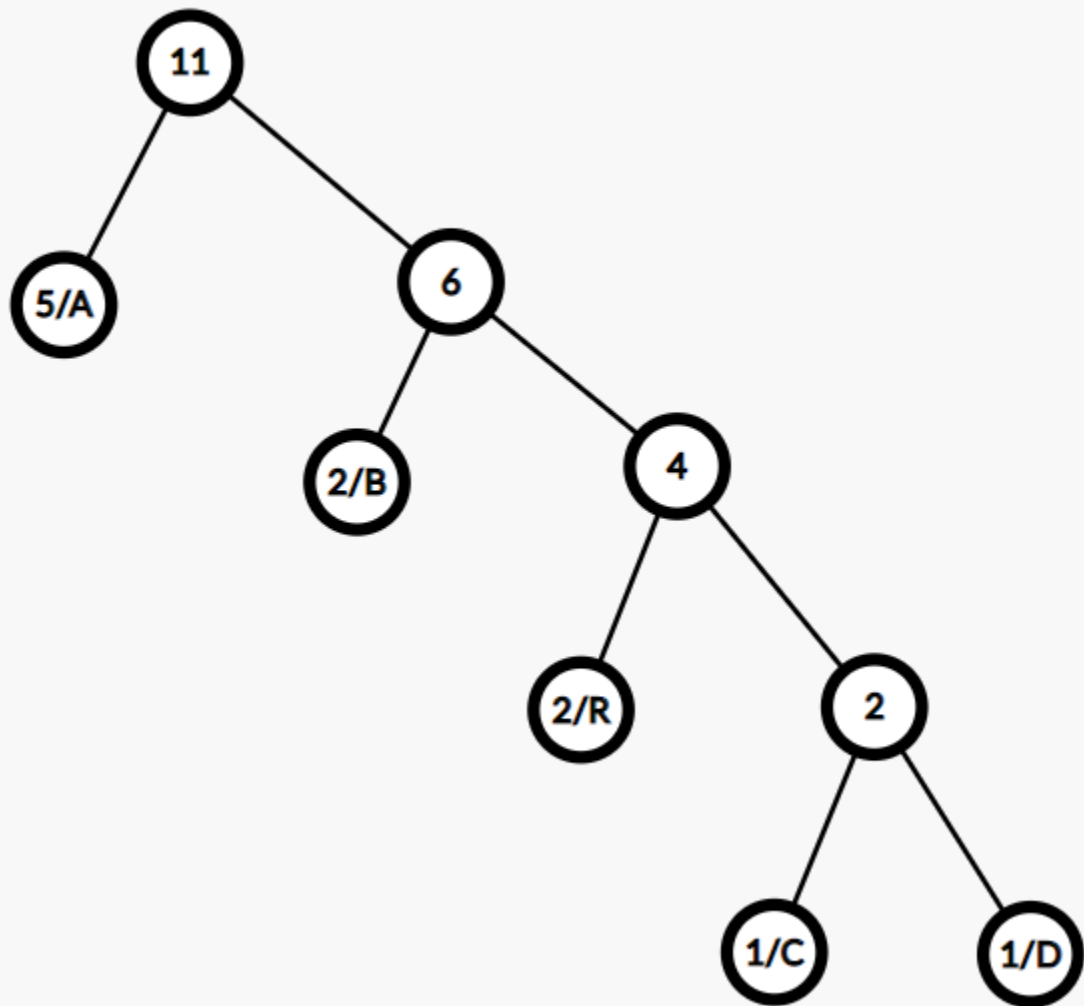
10. Примерна работа в режим Debug

```
Enter command : debug
Mode : Debug mode
Enter command : i test.txt
Input file set
Enter command : start
01101110 10001010 1101110  ->
110 138 110
Original data size : 88
Compressed data size : 23
Compression level : 26% of original : 88
Enter command :
```

Слагаме програмата на режим debug, въвеждаме име на входен файл(име на изходен файл не е нужен, защото резултатът се изписва на екрана и не се запазва във файл). На екрана се печата кодираното съобщение, разделено на блокове от по 8 бита и след това конвертирано в число, изписва се разметър на оригиналното съобщение, размерът на компресираното съобщение и степента на компресия.

11. Построяване на дърво на Хъфман

Ако искаме да кодираме низа „ABRACADABRA“, то дървото на Хъфман по въведения алгоритъм ще изглежда по този начин:



Където листата са изобразени по този начин: (брой срещания) / символ. Ако отпечатаме елементите на дървото по нива, то резултатът ще е такъв:

```
CNT : 11

DATA : A CNT : 5
CNT : 6

DATA : B CNT : 2
CNT : 4

DATA : R CNT : 2
CNT : 2

DATA : C CNT : 1
DATA : D CNT : 1
```

Съответно кодираното съобщение ще има вида:

```
≡ test-result.txt X
≡ test-result.txt
1 01101110100010101101110
```

А таблицата за кодиране ще изглежда така:

```
≡ test-result.txttable.txt M X
≡ test-result.txttable.txt
1 | 101=D
2 | 111=R
3 | 100=C
4 | 0=A
5 | 110=B
6 |
```

12. Class BitSetHelper

```

C BitSetHelper.h X
C BitSetHelper.h
1  #pragma once
2  #include <string>
3  #include <bitset>
4
5  class BitSetHelper{
6      public:
7          // used only in one method in System class
8          // because the bitset<size> must be known in compilation time
9          // and in our case the maxium size is 8 bits
10         std::string getString(int enteredNum, unsigned long number){
11             switch (enteredNum){
12                 case 1:
13                     return std::bitset<1>(number).to_string();
14                 case 2:
15                     return std::bitset<2>(number).to_string();
16                 case 3:
17                     return std::bitset<3>(number).to_string();
18                 case 4:
19                     return std::bitset<4>(number).to_string();
20                 case 5:
21                     return std::bitset<5>(number).to_string();
22                 case 6:
23                     return std::bitset<6>(number).to_string();
24                 case 7:
25                     return std::bitset<7>(number).to_string();
26                 default:
27                     return std::bitset<8>(number).to_string();
28             }
29         }
30     };

```

Класът BitSetHelper има само един метод, който приема като параметър(enteredNum и number), тоест представя числото number в бинарен вид с размерност enteredNum

13. WorkWithBits Компресиране и Декомпресиране

При това компресиране, ако пак имаме входен файл:

```

≡ testbits X
tests > SystemTest > ≡ testbits
1  ABRACADABRA

```

Файлът с компресираното съобщение ще изглежда във вида:

```
testbits-result X
tests > SystemTest > testbits-result
1 110 138 110
```

Тоест енкодираното съобщение от 0 и 1 е разделено на блокове от по 8 бита и са превърнати в десетичното им представяне.

Таблицата за кодиране е същата като при нормалната компресия, само че преди таблицата се записва и размерът на последния блок от 8 бита(дали последният блок е допълнен до 8 бита или не), ако е допълнен, то просто тези 8 бита се превръщат в блок от съответния размер(тук се използва класът BitSetHelper):

```
testbits-resulttable.txt X
tests > SystemTest > testbits-resulttable.txt
1 7
2 101=D
3 111=R
4 100=C
5 110=B
6 0=A
7
```

Съответно декомпресията чете енкодираните числа от входния файл, чете размера на последния блок и чете таблицата за кодиране от съответния файл и декодира съобщението в изходен файл.