

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
ТЕМА: КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ, БДП, ХЕШ-ТАБЛИЦЫ,
СОРТИРОВКИ.

Студент гр. 9382

Кодуков А.В.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы:

Познакомиться с одним из часто используемых на практике алгоритмов кодирования данных.

Задание:*Индивидуальное задание 1:*

На вход подаётся файл с закодированным или незакодированным содержимым. Требуется раскодировать или закодировать содержимое файла определённым алгоритмом.

Кодирование: Фано-Шеннона

Описание алгоритма:*Кодирование Фано-Шеннона:*

Алгоритм использует избыточность сообщения, заключенную в неоднородном распределении частот символов. Также кодирование является префиксным, то есть никакой код не может быть префиксом другого.

Для кодирования файла необходимо два прохода. Первый необходим для подсчета частот всех символов. Затем символы сортируются по убыванию частоты. Далее необходимо построить дерево кодирования. Для этого массив символов делится пополам таким образом, чтобы обе части имели примерно одинаковую частоту. Эти части будут левым и правым поддеревом. Таки разбиение повторяется до тех пор, пока не дойдет до отдельных символов. Таким образом символ всегда является конечным листом дерева. Теперь если обозначить шаг влево по дереву как '0', а вправо как '1', то можно составить коды всех символов как путь до них по дереву. Так как символы хранятся исключительно в листьях, то никакой код не может быть префиксом другого.

Функции и структуры данных:Структуры данных:*Бинарное дерево:*

```
class Tree {
public:
    // Tree node structure
    struct node {
        Elem info;          // Node data
        Tree *lt, *rt;      // Node childs
    };
    ...
};
```

```

    }
private:
    node *Node = nullptr; // Tree root
...
}

```

Символы и частоты:

```

typedef std::map<unsigned char, long> ElemMap;
typedef std::vector<std::pair<unsigned char, long>> ElemArr;

```

Код символа:

```

struct CODE {
    bool Bits[50];
    int Len = 0;
};

```

Реализованные функции:

Построение дерева кодов:

Сигнатура: Tree *BuildCodeTree(ElemArr CurFreq)

Аргументы:

- CurFreq – текущий набор символов и их частот

Алгоритм:

- Массив пуст – вернуть пустой узел
- Массив содержит один символ – вернуть узел, построенный из этого элемента
- Посчитать среднюю частоту символов
- Разбить элементы массива на две примерно равных по частоте части
- Найти левое и правое поддереву как результат работы функции для первой и второй части получившегося разбиения.
- Заполнить и вернуть узел

Построение новых кодов символов:

Сигнатура: void BuildCodes(Tree *T)

Аргументы:

- T –дерево кодирования

Алгоритм:

- Левое и правое поддереву пусты – алгоритм дошел до отдельного символа. Записать накопившийся код в список.
- Левое поддереву не пусто – увеличить текущую длину кода, записать в текущий код '0', запустить функцию от левого поддереву.
- Правое поддереву – аналог. с записью '1'.

Сжатие файла

Сигнатура: `bool Press(const char *filename)`

Возвращаемое значение:

(bool) – Успешно ли кодирование

Аргументы:

- filename – имя файла

Алгоритм:

- Посчитать частоты символов
- Отсортировать символы по убыванию частоты
- Построить дерево кодирования
- Построить коды
- Вернуть файл в начало
- Записать метку
- Записать частоты
- Кодировать данные файлы, накапливая биты в специальном аккумуляторе

Разжатие файла

Сигнатура: `bool Depress()`

Возвращаемое значение:

(bool) – Успешно ли декодирование

Алгоритм:

- Проверить метку
- Считать частоты
- Построить дерево кодирования
- Побитово считывать закодированный файл и восстанавливать данные по дереву кодирования

Тестирование:

| № | Входные данные | Результат (коды и декодированный файл) | | |
|---|----------------|--|------|-----------------------|
| | | Частоты | Коды | Результат кодирования |
| 1 | a | a - 1 | a: 1 | 1 |
| 2 | aaaaaaaaa | a - 9 | a:1 | 1 1 1 1 1 1 1 1 1 |

| | | | | |
|---|---|--|--|--|
| 3 | aa bbb cccc ddddd | d - 5; c - 4; - 3; b - 3; a - 2; | d:00 c:01 :10 b:111 a:111 | 111 111 10 110 110 110 10 01 01 01 01 10 00 00 00 00 00 |
| 4 | happy new year | -2; a-2; e-2; p-2; y-2; h-1; n-1; r-1; w-1; | :00 a:010 e:011 p:100 y:101 h:1100 n:1101 r:1110 w:1111 | 1100 010 100 100 101 00 1101 011 1111 00 101 011 010 1110 |
| 5 | abcdefghijklmnp qrstuvwxyz | a-1; b-1; c-1; d-1; e-1; f-1; g-1; h-1; i-1; j-1; k-1; l-1; m-1; n-1; o-1; p-1; q-1; r-1; s-1; t-1; u-1; v-1; w-1; x-1; y-1; z-1; | a:0000 b:00010 c:00011 d:0010 e:00110 f:00111 g:0100 h:01010 i:01011 j:01100 k:01101 l:01110 m:01111 n:1000 o:10010 p:10011 q:1010 r:10110 s:10111 t:1100 u:11010 v:11011 w:11100 x:11101 y:11110 z:11111 | 0000 00010 00011 0010 00110 00111 0100 01010 01011 01100 01101 01110 01111 1000 10010 10011 1010 10110 10111 1100 11010 11011 11100 11101 11110 11111 |
| 6 | WAR IS PEACE FREEDOM IS SLAVERY IGNORANCE IS STRENGTH | -9; E-7; R-5; S-5; A-4; I-4; N-3; C-2; G-2; O-2; T-2; | :000 E:001 R:010 S:011 A:100 I:1010 N:10110 C:10111 G:1100 O:11010 T:110110 | 111110 100 010 000 1010 011 000 111100 001 100 10111 001 000 111000 010 001 001 110111 11010 111011 000 1010 011 000 011 111010 100 111101 001 010 111111 000 1010 1100 10110 11010 010 100 10110 10111 001 000 1010 011 000 011 110110 010 001 10110 1100 110110 111001 000 |

| | | | | |
|---|--|---|---|--|
| | | D-1; F-1; H-1; L-1; M-1; P-1; V-1; W-1; Y-1; | D:110111 F:111000 H:111001 L:111010 M:111011 P:111100 V:111101 W:111110 Y:111111 | |
| 7 | Следование единому стилю форматировани я исходного кода на протяжении всего исходного кода программы | о-15; -11; и-9; а-7; н-7; д-6; е-5; р-5; г-4; м-4; с-4; в-3; т-3; к-2; л-2; п-2; х-2; я-2; С-1; ж-1; у-1; ф-1; ы-1; ю-1; | о:000 :001 и:010 а:0110 н:0111 д:1000 е:1001 р:1010 г:10110 м:10111 с:11000 в:11001 т:11010 к:11011 л:11100 п:111010 х:111011 я:111100 С:1111010 ж:1111011 у:1111100 ф:1111101 ы:1111110 ю:1111111 | 1111010 11100 1001 1000 000 11001 0110 0111 010 1001 001 1001 1000 010 0111 000 10111 1111100 001 11000 11010 010 11100 1111111 001 1111101 000 1010 10111 0110 11010 010 1010 000 11001 0110 0111 010 111100 001 010 11000 111011 000 1000 0111 000 10110 000 001 11011 000 1000 0110 001 0111 0110 001 111010 1010 000 11010 111100 1111011 1001 0111 010 010 001 11001 11000 1001 10110 000 001 010 11000 111011 000 1000 0111 000 10110 000 001 11011 000 1000 0110 001 111010 1010 000 10110 1010 0110 10111 10111 1111110 |

Вывод:

В результате выполнения работы был изучен и реализован алгоритм кодирования Фано-Шеннона.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

tree.h

```
#ifndef TREE_H_
#define TREE_H_

#include <cstdlib>
#include <fstream>
#include <iostream>
#include <queue>

typedef std::string Elem;

class Tree {
public:
    // Tree node structure
    struct node {
        Elem info; // Node data
        Tree *lt, *rt; // Node childs

        node() {
            lt = nullptr;
            rt = nullptr;
        }

        node(const Elem &x, Tree *lst, Tree *rst) {
            info = x;
            lt = lst;
            rt = rst;
        }
    };

private:
    node *Node = nullptr; // Tree root

public:
    Tree() {}

    Tree(node *N) { Node = N; }

    // Tree memory clear function
    void Clear() {
        if (Node != nullptr) {
            if (Node->lt != nullptr) Node->lt->Clear();
            if (Node->rt != nullptr) Node->rt->Clear();
            delete Node;
            Node = nullptr;
        }
    }

    ~Tree() { Clear(); }

    // Left child getting function
    Tree *Left() {
        if (Node == nullptr) { // No node
            std::cout << "Error: Left(null) \n";
            exit(1);
        } else
            return Node->lt;
    }
};
```

```

    }

    // Right child getting function
    Tree *Right() {
        if (Node == nullptr) {
            std::cout << "Error: Right(null) \n";
            exit(1);
        } else
            return Node->rt;
    }

    // Node pointer getting function
    node *NodePtr() {
        if (Node == nullptr) {
            std::cout << "Error: RootBT(null) \n";
            exit(1);
        } else
            return Node;
    }

    // Node data getting function
    Elem GetNode() {
        if (Node == nullptr) { // No node
            std::cout << "Error: RootBT(null) \n";
            exit(1);
        } else
            return Node->info;
    }
};

#endif // TREE_H_

```

fano.cpp

```

#include "fano.h"

// Types
typedef std::map<unsigned char, long> ElemMap;
typedef std::vector<std::pair<unsigned char, long>> ElemArr;

// Tree printing function
void Print(Tree *q, long n) {
    long i;
    if (q != nullptr) {
        Print(q->Right(), n + q->GetNode().size() * 2);
        for (i = 0; i < n; i++) std::cout << " ";
        std::cout << "\\\" << q->GetNode() << "\\\"\\n";
        Print(q->Left(), n + q->GetNode().size() * 2);
    }
}

Tree *BuildCodeTree(ElemArr CurFreq) {
    // Quit recursion
    if (CurFreq.size() == 0) return nullptr;
    // Symbol leaf case
    if (CurFreq.size() == 1) {
        std::string s;
        s.push_back(CurFreq.begin()->first);
        return new Tree(
            new Tree::node(s, nullptr, nullptr));
    }
    // Count average frequency
    long sum = 0;
    std::string nodestr;

```



```

    for (auto &f : CurFreq) {
        nodestr.push_back(f.first);
        sum += f.second;
    }
    long avg = sum / 2;
    // Splitting current array by frequency
    long cursum = 0;
    auto iter = CurFreq.begin();
    while (cursum < avg) {
        cursum += iter->second;
        iter++;
    }
    // Building left and right subtree
    ElemArr Left(CurFreq.begin(), iter);
    ElemArr Right(iter, CurFreq.end());
    return new Tree(new Tree::node(nodestr, BuildCodeTree(Left),
        BuildCodeTree(Right)));
}

// Code cuildeing definitions
struct CODE {
    bool Bits[50];
    int Len = 0;
};
CODE CurCode;
std::map<unsigned char, CODE> NewCodes;

void BuildCodes(Tree *T) {
    // No subtrees -> symbol found
    if (T->Left() == nullptr && T->Right() == nullptr) {
        unsigned char ch = T->GetNode()[0];
        if (CurCode.Len == 0) {
            CurCode.Bits[0] = 1;
            CurCode.Len = 1;
        }
        NewCodes.insert({ch, CurCode});
        return;
    }
    // Left subtree (0 to code)
    if (T->Left() != NULL) {
        CurCode.Bits[CurCode.Len] = 0;
        CurCode.Len++;
        BuildCodes(T->Left());
        CurCode.Len--;
    }
    // Right subtree (1 to code)
    if (T->Right() != NULL) {
        CurCode.Bits[CurCode.Len] = 1;
        CurCode.Len++;
        BuildCodes(T->Right());
        CurCode.Len--;
    }
}

int comp(const std::pair<unsigned char, long> *i,
        const std::pair<unsigned char, long> *j) {
    return j->second - i->second;
}

bool Press(const char *filename) {
    // Count frequency
    int ch;
    long long size1 = 0, size2 = 0;

```

```

ElemMap Freq;
std::cout << "Pressing file " << filename << "\n";
setlocale(LC_CTYPE, ".1251");
std::ifstream infile(filename, std::ios::in);
if (!infile.is_open()) {
    std::cout << "Impossible to open file\n";
    return false;
}
std::ofstream outfile;
// Counting Frequencies
while ((ch = infile.get()) != EOF) {
    size1++;
    auto iter = Freq.find(ch);
    if (iter != Freq.end())
        (*iter).second++;
    else
        Freq.insert({ch, 1});
}
// Sorting frequencies
ElemArr CurFreq(Freq.begin(), Freq.end());
std::qsort(CurFreq.data(), CurFreq.size(),
           sizeof(std::pair<unsigned char, long>),
           (int (*)(const void *, const void *))comp);
// Building code tree
Tree *T = BuildCodeTree(CurFreq);
// Building new codes
BuildCodes(T);
// Passing through the file second time
infile.seekg(infile.beg);
outfile.open("Files/pressed.txt", std::ios::binary);
// Write label
outfile << "FN!";
outfile << (int)Freq.size();
std::cout << "Frequencies:\n";
// Write frequencies
for (auto &i : CurFreq) {
    std::cout << (unsigned char)i.first << "-" << i.second << ";";
    outfile << i.first;
    outfile.write(reinterpret_cast<char *>(&i.second), sizeof(long));
}
std::cout << "\n";
if (CurFreq.size() <= 15) {
    std::cout << "Tree:\n";
    Print(T, 0);
}
T->Clear();
std::cout << "Codes:\n";
for (auto &c : NewCodes) {
    std::cout << c.first << ":";
    for (int i = 0; i < c.second.Len; i++) std::cout << (int)c.second.Bits[i];
    std::cout << "\n";
}
// Coding input data to output file
unsigned char BitAccum = 0;
int BitPos = 7;
infile.close();
infile.open(filename);
while ((ch = infile.get()) != EOF) {
    for (int k = 0; k < NewCodes[ch].Len; k++) {
        BitAccum |= NewCodes[ch].Bits[k] << BitPos--;
        // Writing byte
        if (BitPos < 0) {
            size2++;

```

```

        outfile << BitAccum;
        BitAccum = 0;
        BitPos = 7;
    }
}
}
if (BitPos < 7) outfile << BitAccum, size2++;
std::cout << "Input size: " << size1
        << "\nPressed size(pure input data): " << size2 << "\n";

infile.close();
outfile.close();
return true;
}

// Decompress function
bool Depress() {
    std::ifstream infile("Files/pressed.txt", std::ios::binary);
    std::ofstream outfile;
    if (!infile.is_open()) {
        std::cout << "Impossible to open file\n";
        return false;
    }
    setlocale(LC_ALL, "Russian");
    // Check label
    char label[4];
    infile.read(label, 3);
    label[3] = '\0';
    if (strcmp(label, "FN!") != 0) {
        infile.close();
        std::cout << "Wrong pressed file\n";
        return false;
    }
    // Read frequencies
    int cnt;
    long size = 0;
    infile >> cnt;
    ElemMap Freq;
    for (int i = 0; i < cnt; i++) {
        unsigned char ch;
        unsigned long num = 0;
        ch = infile.get();
        unsigned char numstr[4];
        for (int i = 0; i < 4; i++) numstr[3 - i] = infile.get();
        for (int i = 0; i < 4; i++) {
            num <<= 8;
            num |= numstr[i];
        }
        Freq.insert({ch, num});
        size += num;
    }
    // Sort frequencies
    Tree *T, *Start;
    ElemArr CurFreq(Freq.begin(), Freq.end());
    std::qsort(CurFreq.data(), CurFreq.size(), sizeof(CurFreq[0]),
        (int (*)(const void *, const void *))comp);
    // Building code tree
    T = BuildCodeTree(CurFreq);
    Start = T;
    int ch;
    unsigned char BitAccum;
    int BitPos = -1, res = 0;
    bool isfirst = true, isstart = true;

```

```

// Reading coded data
outfile.open("Files/decompressed.txt");
long num = 0;
while (1) {
    // Leaf -> symbol
    if (!isfirst && T->Left() == NULL) {
        if (isstart && !res)
            break;
        outfile << (unsigned char)T->GetNode()[0];
        num++;
        if (num == size)
            break;
        T = Start;
        isstart = true;
    }
    if (isfirst) isfirst = false;
    // Get new byte
    if (BitPos < 0) {
        ch = infile.get();
        if (ch == EOF) break;
        BitAccum = ch;
        BitPos = 7;
    }
    // 0 - go left, 1 - go right
    res = (BitAccum >> BitPos--) & 1;
    if (res && (T->Right() != nullptr)) {
        T = T->Right();
        isstart = false;
    } else if (T->Left() != nullptr) {
        isstart = false;
        T = T->Left();
    }
}
infile.close();
outfile.close();

return true;
}

```

fano.h

```

#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <cstring>
#include <set>

#include "tree.h"

bool Press(const char *filename);

bool Depress();

```

main.cpp

```

#include <algorithm>
#include "fano.h"

int main() {
    std::string fname;

```

```

std::cout << "Input filename: ";
std::cin >> fname;
if (Press(fname.data())) {
    std::cout << "Compression complete!\n";
    if (Depress()) std::cout << "Decompression complete!\n";
    else
        std::cout << "Decompression error\n";
} else
    std::cout << "Compression error\n";
system("pause");
}

```