**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе № 5**

**по дисциплине «Программирование»**

**Тема: кодирование методом Фано - Шеннона**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Быков И. В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

1. **Цель работы.**

Научиться применять бинарные деревья для решения задач кодирования (сжатия) и поиска.

1. **Постановка задачи.**

Реализация кодирования методом Фано – Шеннона.

1. **Основные теоретические положения.**

Алгоритм Шеннона — Фано — один из первых алгоритмов сжатия, который впервые сформулировали американские учёные Шеннон и Роберт Фано. Данный метод сжатия имеет большое сходство с алгоритмом Хаффмана, который появился на несколько лет позже и является логическим продолжением алгоритма Шеннона. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Шеннона — Фано префиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Основные этапы:

* Символы первичного алфавита m1 выписывают по убыванию вероятностей.
* Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
* В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
* Полученные части рекурсивно делятся и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Алгоритм вычисления кодов.

Код Шеннона — Фано строится с помощью дерева. Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева; такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины. Ветви кодового дерева размечаем символами 1 и 0, как в случае кода Хаффмана.

При построении кода Шеннона — Фано разбиение множества элементов может быть произведено, вообще говоря, несколькими способами. Выбор разбиения на уровне n может ухудшить варианты разбиения на следующем уровне (n + 1) и привести к неоптимальности кода в целом. Другими словами, оптимальное поведение на каждом шаге пути ещё не гарантирует оптимальности всей совокупности действий. Поэтому код Шеннона — Фано не является оптимальным в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона — Фано, и все они могут дать различные результаты. Если построить все возможные коды Шеннона — Фано для данного распределения вероятностей, то среди них будут находиться и все коды Хаффмана, то есть оптимальные коды.

1. **Спецификация программы.**

Программа предназначена для кодирования строки методом Фано – Шеннона.

Программа написана на языке C++ с использованием компилятора mingw32- g++. Входными данными для программы являются:

массив символов char – сообщение для кодирования (вводится с клавиатуры)

Выходные данные:

массив типа char – закодированная строка.

массив типа char – исходная строка.

построенное дерево.

1. **Пример диалога с пользователем.**

Enter the string: a

Tree:

1

Code: 0

Decode: a

Repeat? 1 - yes; 0 - no. 1

Enter the string: abcdefg

Tree:

1

3

1

2

1

7

1

2

1

4

1

2

1

Code: 00010011100101110111

Decode: abcdefg

Repeat? 1 - yes; 0 - no. 1

Enter the string: aaaaabbcccccjdddddd

Tree:

6

11

5

19

5

8

2

3

1

Code: 01010101011101101010101010111000000000000

Decode: aaaaabbcccccjdddddd

Repeat? 1 - yes; 0 - no. 0

1. **Реализация.**

Структура описания символа:

struct Symbol {

char c;

int weight;

};

Структура описания кодового дерева:

struct CodeTree {

Symbol s;

CodeTree\* parent;

CodeTree\* left;

CodeTree\* right;

};

Базовые функции для операций над деревом:

CodeTree\* make\_leaf(const Symbol& s); //

CodeTree\* make\_node(int weight, CodeTree\* left, CodeTree\* right);

bool is\_null(const CodeTree\* node);

bool is\_leaf(const CodeTree\* node);

bool is\_root(const CodeTree\* node);

char\* encode(const CodeTree\* tree, const char\* message); - кодирование

char\* decode(const CodeTree\* tree, const char\* code); - декодирование

void destroy(CodeTree\* tree);

void print\_tree(CodeTree\* tree, int level, std::ostream& out);

Базовые функции реализации алгоритма Фано – Шеннона.

CodeTree\* fanno\_shannon(const char\* message);

// Описание: получаем на вход заданную строку, считаем вхождения символов и записываем в symbols, сортируем по весам. Возвращаем fanno\_shannon(symbols, len)

CodeTree\* fanno\_shannon(const Symbol\* symbols, int len);

// Описание: считаем сумму всех весов, возвращаем fanno\_shannon(symbols, 0, len, sum)

CodeTree\* fanno\_shannon(const Symbol\* symbols, int l, int r, int sum)

// Описание: коротко говоря, здесь создается дерево

1. **Тестирование.**

|  |  |
| --- | --- |
| code | encode |
| a | 0 |
| aaaaaaaa | 00000000 |
| ab | 01 |
| abb | 100 |
| abcdef | 0001001110110111 |
| abbcccddddeeeee | 111110110101010010101010000000000 |

**Вывод****.**

В результате выполнения лабораторной работы мною были получены основные сведения о кодировании методом Фано - Шеннона. Мною были изучены базовые функции обработки бинарных деревьев, методы кодироания и декодирования сообщений.

**Приложение А. Исходный код.**

**main.cpp**

#include "fh.h"

int main()

{

int flag = 0;

while (true)

{

CodeTree\* ct = nullptr;

char \*c;

c = new char[MAX\_CODE\_LEN];

char \*\_encode, \*\_decode;

\_encode = new char[MAX\_CODE\_LEN];

std::cout << "Enter the string: ";

std::cin >> c;

ct = fanno\_shannon(c);

\_encode = encode(ct, c);

\_decode = decode(ct, \_encode);

std::cout << "Tree:\n";

print\_tree(ct, 0, std::cout);

std::cout << "Code: ";

std::cout << \_encode << std::endl;

std::cout << "Decode: ";

std::cout << \_decode << std::endl;

destroy(ct);

delete[] c;

std::cout << "Repeat? 1 - yes; 0 - no. ";

std::cin >> flag;

std::cout << std::endl;

if (flag == 0)

return 0;

}

//system("pause");

}

fh.cpp

#include "fh.h"

#include <algorithm>

#include <climits>

#include <cstring>

static int middle(const Symbol\* symbols, int l, int sum, int& lsum, int& rsum);

CodeTree\* fanno\_shannon(const Symbol\* symbols, int l, int r, int sum)

{

if (l >= r) return nullptr;

if (r - l == 1) return make\_leaf(symbols[l]);

int lsum, rsum;

int m = middle(symbols, l, sum, lsum, rsum);

CodeTree\* ltree = fanno\_shannon(symbols, l, m + 1, lsum);

CodeTree\* rtree = fanno\_shannon(symbols, m + 1, r, rsum);

CodeTree\* node = make\_node(sum, ltree, rtree);

ltree->parent = node;

rtree->parent = node;

return node;

}

CodeTree\* fanno\_shannon(const Symbol\* symbols, int len)

{

int sum = 0;

for (int i = 0; i < len; ++i)

sum += symbols[i].weight;

return fanno\_shannon(symbols, 0, len, sum);

}

CodeTree\* fanno\_shannon(const char\* message)

{

Symbol symbols[UCHAR\_MAX];

for (int i = 0; i < UCHAR\_MAX; ++i) {

symbols[i].c = i + CHAR\_MIN;

symbols[i].weight = 0;

}

int size = strlen(message);

for (int i = 0; i < size; ++i)

symbols[message[i] - CHAR\_MIN].weight++;

std::sort(symbols, symbols + UCHAR\_MAX, symbol\_greater);

int len = 0;

while (symbols[len].weight > 0 && len < UCHAR\_MAX) len++;

return fanno\_shannon(symbols, len);

}

int middle(const Symbol\* symbols, int l, int sum, int& lsum, int& rsum)

{

int m = l;

lsum = symbols[m].weight;

rsum = sum - lsum;

int delta = lsum - rsum;

while (delta + symbols[m + 1].weight < 0) {

m++;

lsum += symbols[m].weight;

rsum -= symbols[m].weight;

delta = lsum - rsum;

}

return m;

}

CodeTree.cpp

#include "CodeTree.h"

#include <climits>

#include <cstring>

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

bool symbol\_less(const Symbol& l, const Symbol& r)

{

return l.weight < r.weight;

}

bool symbol\_greater(const Symbol& l, const Symbol& r)

{

return l.weight > r.weight;

}

CodeTree\* make\_leaf(const Symbol& s)

{

return new CodeTree{ s, nullptr, nullptr, nullptr };

}

CodeTree\* make\_node(int weight, CodeTree\* left, CodeTree\* right)

{

Symbol s{ 0, weight };

return new CodeTree{ s, nullptr, left, right };

}

bool is\_leaf(const CodeTree\* node)

{

return ((node->left == nullptr) && (node->right == nullptr));

}

bool is\_null(const CodeTree\* node)

{

return (node == nullptr);

}

bool is\_root(const CodeTree\* node)

{

return node->parent == nullptr;

}

static void fill\_symbols\_map(const CodeTree\* node, const CodeTree\*\* symbols\_map);

char\* encode(const CodeTree\* tree, const char\* message)

{

char\* code = new char[MAX\_CODE\_LEN];

int len = strlen(message);

const CodeTree\*\* symbols\_map = new const CodeTree\*[UCHAR\_MAX];

for (int i = 0; i < UCHAR\_MAX; ++i)

{

symbols\_map[i] = nullptr;

}

fill\_symbols\_map(tree, symbols\_map);

int index = 0;

char path[UCHAR\_MAX];

for (int i = 0; i < len; ++i)

{

const CodeTree\* node = symbols\_map[message[i] - CHAR\_MIN];

int j = 0;

while (!is\_root(node))

{

if (node->parent->left == node)

path[j++] = '0';

else

path[j++] = '1';

node = node->parent;

}

while (j > 0) code[index++] = path[--j];

}

if (index != 0)

code[index] = 0;

else

{

for (int i = 0; i < strlen(message); i++)

code[i] = '0';

code[strlen(message)] = 0;

}

delete[] symbols\_map;

return code;

}

char\* decode(const CodeTree\* tree, const char\* code)

{

char\* message = new char[MAX\_CODE\_LEN];

int index = 0;

int len = strlen(code);

const CodeTree\* v = tree;

if (is\_leaf(v))

{

for (int i = 0; i < len; ++i)

{

message[index++] = v->s.c;

v = tree;

}

}

else

for (int i = 0; i < len; ++i)

{

if (code[i] == '0')

v = v->left;

else

v = v->right;

if (is\_leaf(v))

{

message[index++] = v->s.c;

v = tree;

}

}

message[index] = 0;

return message;

}

void destroy(CodeTree\* tree)

{

if (tree == nullptr) return;

destroy(tree->left);

destroy(tree->right);

delete tree;

tree = nullptr;

}

void fill\_symbols\_map(const CodeTree\* node, const CodeTree\*\* symbols\_map)

{

if (is\_leaf(node))

symbols\_map[node->s.c - CHAR\_MIN] = node;

else {

fill\_symbols\_map(node->left, symbols\_map);

fill\_symbols\_map(node->right, symbols\_map);

}

}

void print\_tree(CodeTree\* tree, int level, std::ostream& out)

{

if (tree)

{

print\_tree(tree->left, level + 1, out);

for (int i = 0; i < level; i++) out << " ";

out << tree->s.weight << std::endl;

print\_tree(tree->right, level + 1, out);

}

}

CodeTree.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#define MAX\_CODE\_LEN 1000

struct Symbol {

char c;

int weight;

};

bool symbol\_less(const Symbol& l, const Symbol& r);

bool symbol\_greater(const Symbol& l, const Symbol& r);

struct CodeTree {

Symbol s;

CodeTree\* parent;

CodeTree\* left;

CodeTree\* right;

};

CodeTree\* make\_leaf(const Symbol& s);

CodeTree\* make\_node(int weight, CodeTree\* left, CodeTree\* right);

bool is\_null(const CodeTree\* node);

bool is\_leaf(const CodeTree\* node);

bool is\_root(const CodeTree\* node);

char\* encode(const CodeTree\* tree, const char\* message);

char\* decode(const CodeTree\* tree, const char\* code);

void destroy(CodeTree\* tree);

void print\_tree(CodeTree\* tree, int level, std::ostream& out);

fh.h

#pragma once

#include "CodeTree.h"

CodeTree\* fanno\_shannon(const char\* message);

CodeTree\* fanno\_shannon(const Symbol\* symbols, int len);