**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра CАПР**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 2**

Тема: Алгоритмы кодирования

|  |  |  |
| --- | --- | --- |
| Студент гр. 0309 |  | Головатюк К.А. |
| Преподаватель |  | Тутуева А.В |

Санкт-Петербург

2022

### Постановка задачи

1. Реализовать кодирование и декодирование по алгоритму Шеннона-Фано (2 вариант) входной строки, вводимой через консоль.

2. Посчитать объем памяти, который занимает исходная и закодированная

Строки.

3. Выводить на экран таблицу частот и кодов, результат кодирования и декодирования, коэффициент сжатия.

4. Стандартные структуры данных С++ использовать нельзя. Необходимо использовать структуры данных из предыдущих лабораторных работ.

Наличие unit-тестов является обязательным требованием

### Описание реализованных методов и оценка их временной сложности

Class ShenonFano

n – количество уникальных символов в строке, m – длинна строки.

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Оценка временной сложности |
| void create\_table() | Создание таблицы | O(m + 2^(n-1)) |
| string code() | Кодирование строки | O(m \* n) |
| string decode() | Декодирование строки | О(m \* n) |
| void print\_table() | Вывод в консоль | О(n) |

Class List

n – размер списка.

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Оценка временной сложности |
| void push(char c) | Добавление элемента в список | O(n) |
| void sort() | Сортировка списка | O(n^2) |
| string search(char c) | Поиск элемента по символу, возвращает код элемента | О(n) |

### Описание алгоритма Шеннона — Фано и структур данных

1. Символы сортируются по убыванию частоты.

2. Символы делятся на 2 группы, с примерной одинаковой частотой.

3. 1 – группе присваивается ‘0’, 2 – группе присваивается ‘1’.

4. Группы рекурсивно делятся, и им присваиваются двоичные цифры

5. Если группа состоит из 1 элемента, этот элемент становится листом бинарного дерева, который хранит в себе код, символ и частоту символа.

Алгоритм является не оптимальным, так для одной строки можно построить несколько кодов, и они будут отличаться по сжатию, но если построить все возможные коды, то среде них будут оптимальные.

Были использованы такие структуры как, бинарное дерево и список.

Список хранит в себе набор уникальных символов строки и их частоту. Бинарное дерево хранит в себе закодированные символы, код элемента является путем до этого элемента в бинарном дереве, такая структура позволяет эффективно раскодировать строку.

### Описание реализованных unit-тестов

Список тестов:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

### Пример работыИзображение выглядит как текст Автоматически созданное описание

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Вывод**

При реализации были улучшены навыки создания таких структур данных как, списки и бинарные деревья. Получены навыки кодирования и декодирования. Изучен алгоритм Шеннона-Фано.

### Листинг

Файл main.cpp:

#include <iostream>

#include "ShenonFano.h"

using namespace std;

int main()

{

string str = "";

getline(cin, str);

ShenonFano test(str);

test.create\_table();

test.print\_table();

cout << "code - " << test.code() << endl;

cout << "decode - " << test.decode() << endl;

cout << "size code string = " << test.take\_code().size() << endl;

cout << "size string = " << test.take\_decode().size() \* 8 << endl;

cout << "compression ratio = " << float(test.take\_decode().size() \* 8) / float(test.take\_code().size()) << endl;

}

Файл ShenonFano.h:

#pragma once

#include "List.h"

#include <iostream>

using namespace std;

class NodeTree {

public:

string symb;

string code;

int freq;

NodeTree\* left;

NodeTree\* right;

NodeTree() : symb(""), code(""), freq(0), left(nullptr), right(nullptr) {};

};

class ShenonFano {

public:

ShenonFano(string s) : str(s), decode\_str(""), root(nullptr), code\_str(""), table(nullptr) {};

void create\_table();

void print\_table();

string code();

string decode();

string take\_decode() { return decode\_str; };

string take\_code() { return code\_str; };

List\* take\_table() { return table; };

private:

void create\_code(string code, NodeTree\* node, int start, int end);

string str;

string decode\_str;

string code\_str;

List\* table;

NodeTree\* root;

};

Файл List.h

#pragma once

#include <string>

using namespace std;

class Node {

public:

int freq;

char symb;

string code;

Node\* next;

Node\* prev;

Node(char c) : freq(1), code(""), symb(c), next(nullptr), prev(nullptr) {};

};

class List{

public:

List() : size(0), head(nullptr), tail(nullptr) {};

void push(char c);

void sort();

Node\* take\_head() { return this->head; };

Node\* take\_tail() { return this->tail; };

int take\_size() { return this->size; };

string search(char c);

private:

void swap(Node\* l, Node\* r);

int size;

Node\* head;

Node\* tail;

};

Файл ShenonFano.cpp:

#include "ShenonFano.h"

void ShenonFano::create\_table()

{

this->table = new List();

for (int i = 0; i < str.size(); i++) {

this->table->push(str[i]);

}

this->table->sort();

this->root = new NodeTree();

if (this->table->take\_size() == 1) {

this->root->left = new NodeTree();

this->root->left->code = "0";

this->root->left->symb = table->take\_head()->symb;

this->root->left->freq = table->take\_head()->freq;

table->take\_head()->code = "0";

return;

}

create\_code("", this->root, 1, this->table->take\_size());

}

void ShenonFano::create\_code(string code, NodeTree\* node, int start, int end)

{

if (node == nullptr) {

node = new NodeTree();

}

Node\* buff = this->table->take\_head();

for (int i = 1; i < start; i++) {

buff = buff->next;

}

if (start == end) {

node->code = code;

node->symb = buff->symb;

node->freq = buff->freq;

buff->code = code;

return;

}

Node\* cur = buff;

string str = "";

int ds = 0;

for (int i = start; i < end; i++) {

ds += buff->freq;

str = str + buff->symb;

buff = buff->next;

}

ds /= 2;

int S = 0;

int it = start;

int m = start;

buff = cur;

while ((S + buff->freq < ds) && (it < end)) {

S += buff->freq;

it++; m = it;

buff = buff->next;

}

node->symb = str;

node->left = new NodeTree();

node->right = new NodeTree();

create\_code(code + '0', node->left, start, m);

create\_code(code + '1', node->right, m+1 , end);

}

void ShenonFano::print\_table()

{

Node\* buff = this->table->take\_head();

while (buff)

{

cout << buff->symb << " - " << buff->code

<< " frequency = " << buff->freq << endl;

buff = buff->next;

}

}

string ShenonFano::code()

{

this->code\_str = "";

for (int i = 0; i < this->str.size(); i++) {

this->code\_str = this->code\_str + this->table->search(str[i]);

}

return this->code\_str;

}

string ShenonFano::decode()

{

this->decode\_str = "";

if (root == nullptr)

return decode\_str;

NodeTree\* buff = this->root;

for (int i = 0; i < this->code\_str.size(); i++) {

if (this->code\_str[i] == '0') {

buff = buff->left;

}

else {

buff = buff->right;

}

if ((buff->left == nullptr) && (buff->right == nullptr)) {

this->decode\_str = this->decode\_str + buff->symb;

buff = this->root;

}

}

return decode\_str;

}

Файл List.cpp

#include "List.h"

void List::push(char c)

{

if (this->head == nullptr) {

this->head = new Node(c);

this->tail = this->head;

this->size++;

return;

}

Node\* buff = this->head;

while (buff)

{

if (buff->symb == c) {

buff->freq++;

return;

}

buff = buff->next;

}

buff = this->tail;

Node\* add = new Node(c);

add->prev = buff;

buff->next = add;

this->tail = add;

this->size++;

}

void List::sort()

{

for (int i = 0; i < this->size; i++) {

Node\* left = this->head;

Node\* right = this->head->next;

for (int j = 0; j < this->size - 1; j++) {

if (left->freq < right->freq) {

swap(left, right);

left = right->next;

right = left->next;

continue;

}

left = left->next;

right = right->next;

}

}

}

void List::swap(Node\* l, Node\* r)

{

Node\* buff = r->next;

r->next = l;

l->next = buff;

buff = l->prev;

l->prev = r;

r->prev = buff;

if (r->prev)

r->prev->next = r;

else

this->head = r;

if (l->next)

l->next->prev = l;

else

this->tail = l;

}

string List::search(char c)

{

Node\* buff = this->head;

while (buff)

{

if (buff->symb == c) {

return buff->code;

}

buff = buff->next;

}

return "\_";

}

Файл test.cpp

#include "pch.h"

#include "ShenonFano.h"

#include <iostream>

#include <windows.h>

void PrintTabs(HANDLE hStdOut) {

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "[ ] ";

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

}

void TreeOut(ShenonFano test, HANDLE hStdOut) {

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "[ String] ";

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

cout << test.take\_decode() << endl;

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "[ Code] ";

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

cout << test.take\_code() << endl;

}

void ListOut(List test, HANDLE hStdOut) {

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "[ Symbols] ";

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

Node\* buff = test.take\_head();

while (buff)

{

cout << buff->symb << " ";

buff = buff->next;

}

cout << endl;

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "[ Freq] ";

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

buff = test.take\_head();

while (buff)

{

cout << buff->freq << " ";

buff = buff->next;

}

cout << endl;

}

TEST(ShenonFanoTest, create\_table) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

ShenonFano test("asdfsagasdklasfasaasfdaafas");

char symb[] = { 'a', 's', 'f', 'd', 'g', 'k', 'l' };

string code[] = { "00", "01", "10", "110", "1110", "11110", "11111" };

int freq[] = { 10,7,4,3,1,1,1 };

test.create\_table();

test.code();

test.decode();

TreeOut(test, hStdOut);

List\* table = test.take\_table();

Node\* buff = table->take\_head();

for (int i = 0; i < 7; i++) {

ASSERT\_EQ(buff->symb, symb[i]);

ASSERT\_EQ(buff->freq, freq[i]);

ASSERT\_STREQ(buff->code.c\_str(), code[i].c\_str());

buff = buff->next;

}

}

TEST(ShenonFanoTest, code) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

ShenonFano test("asdfsagasdklasfasaasfdaafas");

char str[] = "00011101001001110000111011110111110001100001000001101100000100001";

test.create\_table();

test.code();

test.decode();

TreeOut(test, hStdOut);

ASSERT\_STREQ(test.take\_code().c\_str(), str);

}

TEST(ShenonFanoTest, decode) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

ShenonFano test("asdfsagasdklasfasaasfdaafas");

char str[] = "asdfsagasdklasfasaasfdaafas";

test.create\_table();

test.code();

test.decode();

TreeOut(test, hStdOut);

ASSERT\_STREQ(test.take\_decode().c\_str(), str);

}

TEST(ListTest, push) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

List test;

string str = "1234444456";

char arr[] = { '1','2','3','4','5','6' };

int freq[] = { 1,1,1,5,1,1 };

for (int i = 0; i < str.size(); i++) {

test.push(str[i]);

}

ListOut(test, hStdOut);

Node\* buff = test.take\_head();

for (int i = 0; i < 6; i++) {

ASSERT\_EQ(buff->symb, arr[i]);

ASSERT\_EQ(buff->freq, freq[i]);

buff = buff->next;

}

}

TEST(ListTest, sort) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

List test;

string str = "1234444456";

char arr[] = { '4','1','2','3','5','6' };

int freq[] = { 5,1,1,1,1,1 };

for (int i = 0; i < str.size(); i++) {

test.push(str[i]);

}

test.sort();

ListOut(test, hStdOut);

Node\* buff = test.take\_head();

for (int i = 0; i < 6; i++) {

ASSERT\_EQ(buff->symb, arr[i]);

ASSERT\_EQ(buff->freq, freq[i]);

buff = buff->next;

}

}

TEST(ListTest, search) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, FOREGROUND\_GREEN | FOREGROUND\_BLUE

| FOREGROUND\_RED | FOREGROUND\_INTENSITY);

ShenonFano table("1234444456");

table.create\_table();

List\* test = table.take\_table();

string code[] = { "0","100","101","110","1110","1111" };

ListOut(\*test, hStdOut);

Node\* buff = test->take\_head();

for (int i = 0; i < 6; i++) {

ASSERT\_STREQ(buff->code.c\_str(), code[i].c\_str());

buff = buff->next;

}

}