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

Кафедра САПР

**ОТЧЕТ**

**по лабораторной работе №2**

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

**Тема: «Алгоритмы кодирования»**

**Вариант 2**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гелета А.И.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А. В.

Санкт-Петербург

2021

# Постановка задачи

Реализовать кодирование и декодирование по алгоритму Шеннона-Фано входной строки, вводимой через консоль.

# Описание реализуемого класса и методов

class SFCode

Класс кодирования и декодирования по алгоритму Шеннона-Фано.

Поля:

Node\* root – корень дерева

char\* inputString – входная строка

int inputStringLength – длина входной строки

RedBlackTree<char, SymbCode\*>\* codes – коды элементов

bool\* codedString – закодированная строка

int codedStringLength – закодированная строка

Методы:

PriorityList<char>\* CountProbability(char \*, int\*) – считает количество появлений символов в строке

void buildTree(Node\*) - строит дерево

void Code(Node\*) – строит коды для каждого символа

bool\* CodeString() – кодирует строку

int GetCodeLength() – возвращает длину строки

char\* DecodeString() - - декодирование строки

void CodesOut() – вывод кодов в консоль

double Coeff() - возвращает коэффициент сжатия

class Node

Элемент дерева кодов.

Поля:

PriorityList<char>\* symbols – массив символов в этом элементе

int allSymsbols – количество символов

bool ifLeaf – является ли элемент листов

Node\* left – левый ребенок

Node\* right – правый ребенок

Node\* parent - родитель

class SymbCode

Класс кода одного элемента.

Поля:

bool\* code – сама код.

int length – длина кода.

# Оценка временной сложности каждого метода

|  |  |
| --- | --- |
| Метод | Сложность |
| buildTree+Code | O(n\*n) |
| CodeString | O(n\*logn) |
| DecodeString | O(n\*logn) |

# Описание реализованных unit-тестов

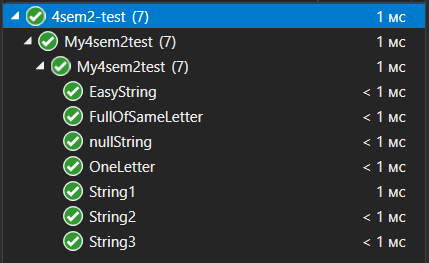
EasyString – кодирование простой строки (5 элементов).

FullOfSameLetter – кодирование строки, состоящий из одних и тех же символов

nullString – кодирование пустой строки

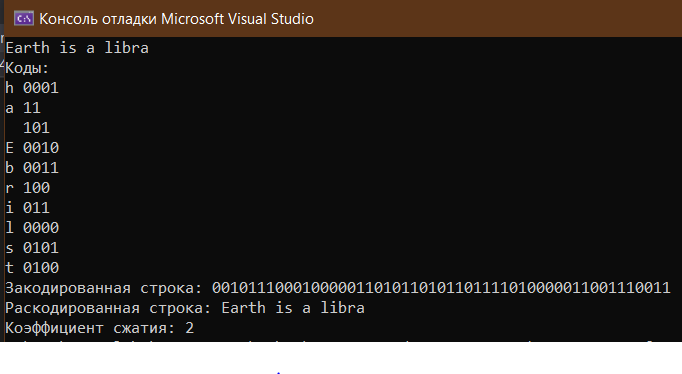
OneLetter – кодирование сроки, состоящей из одного символа

String1, String2, String3 – кодирование разных строк.

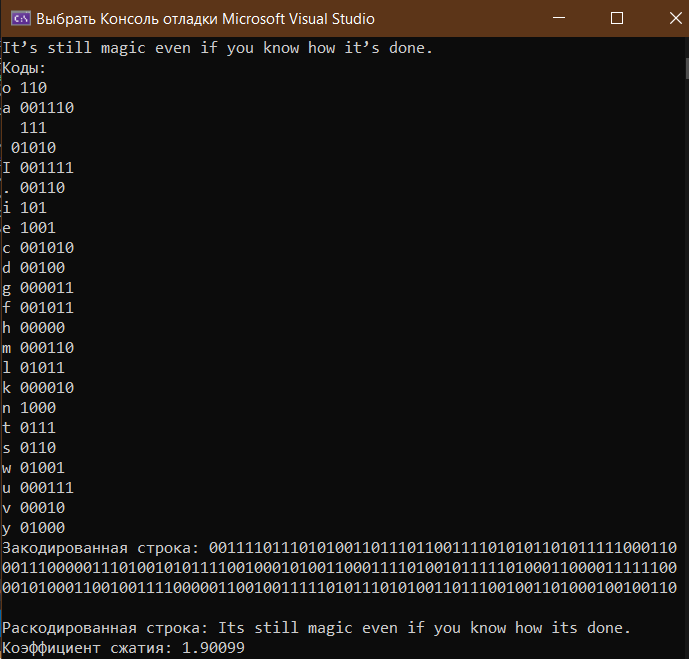


# Примеры работы программы

На рисунках ниже приведены примеры работы программы при вводе строки с клавиатуры. Выводятся коды, присвоенные каждому символу, закодированная строки, раскодирования строка и коэффициент сжатия.



# 



# Текст программы

|  |  |
| --- | --- |
|  | **SFCode.h** |
|  | #pragma once  #include "RBtree.h"  #include "PriorityList.h"  class SFCode  {  private:  class Node {  public:  PriorityList<char>\* symbols;  int allSymsbols;  bool ifLeaf;  Node\* left;  Node\* right;  Node\* parent;  Node() {  this->left = nullptr; this->right = nullptr; this->parent = nullptr;  this->symbols = new PriorityList<char>();  this->allSymsbols = 0;  this->ifLeaf = false;  }  };  class SymbCode {  public:  bool\* code;  int length;  SymbCode(bool\* code = nullptr, int length = 0) { this->code = code; this->length = length; }  };    Node\* root;  char\* inputString;  int inputStringLength;  RedBlackTree<char, SymbCode\*>\* codes; //key - symbol, value - code  bool\* codedString;  int codedStringLength;  PriorityList<char>\* CountProbability(char \*, int\*);  void buildTree(Node\*); //build codes tree  void Code(Node\*); //connect letters and codes  public:  SFCode(char\* string, int string\_length) {  this->codedString = NULL;  this->codedStringLength = 0;  this->inputString = string;  this->inputStringLength = string\_length;  if (string\_length == 0) { root = nullptr;}  else {  root = new Node();  root->symbols = CountProbability(string, &string\_length);  root->allSymsbols = string\_length;  if (root->symbols->get\_size() == 1) root->ifLeaf = true;  else { root->ifLeaf = false; buildTree(root); }  }  Code(root);  }  bool\* CodeString(); //get coded string  int GetCodeLength() { return codedStringLength; } //for test purposes  char\* DecodeString(); //get decoded string  void CodesOut(); //write codes in console  double Coeff() { return (double)sizeof(char) \* inputStringLength \* 8 / codedStringLength; } //how compressed it is  ~SFCode() {};  }; |
|  | **SFCode.cpp** |
|  | #include "SFCode.h"  #include"Stack.h"  PriorityList<char>\* SFCode::CountProbability(char\* string, int\* length) {  PriorityList<char>\* symb = new PriorityList<char>();  //symbols  int chars[256] = { 0 };  int count = 0;  for (int i = 0; (string[i] != '\n' && i < \*length); i++)  {  chars[string[i]]++; count++;  }  for (int i = 0; i < 256; i++) {  //if symbol exist in string then add it  if (chars[i] != 0) symb->push((char)i, chars[i]);  }  \*length = count;  return symb;  }  void SFCode::buildTree(Node\* current) {  if (current->symbols->get\_size() == 1) { current->ifLeaf = true; return; }  PriorityList<char>\* symb = new PriorityList<char>();  symb = current->symbols->Copy();  Node\* left = new Node();  Node\* right = new Node();  int probLeft, probRight;  //split in half  probLeft = symb->head\_priority(); left->symbols->push(symb->pop\_front(), probLeft);  probRight = symb->tail\_priority(); right->symbols->push(symb->pop\_back(), probRight);  //try to split in more or less same parts  while (!symb->isEmpty()) {  int probCur;  if (probLeft > probRight) {  //if left is bigger need to increase right  probCur = symb->tail\_priority();  right->symbols->push(symb->pop\_back(), probCur);  probRight += probCur;  }  else {  probCur = symb->head\_priority();  left->symbols->push(symb->pop\_front(), probCur);  probLeft += probCur;  }  }  //making nodes  left->allSymsbols = probLeft;  right->allSymsbols = probRight;  current->left = left;  current->right = right;  left->parent = current;  right->parent = current;  //call recursively for left and right  buildTree(left);  buildTree(right);  }  void SFCode::Code(Node\* root) {  if (root == nullptr){  this->codes = nullptr;  return;  }  Stack<Node\*>\* stack = new Stack<Node\*>();  stack->push(root);  RedBlackTree<char, SymbCode\*>\* codes = new RedBlackTree<char, SymbCode\*>();  bool\* curCode = new bool[root->allSymsbols / 2 + 1];  Node\* cur = root;  int curDepth = 0;  if (root->ifLeaf) { curDepth = 1; curCode[0] = 0; }  //go in tree like in dft  while (!stack->isEmpty()) {  if (cur->left != NULL) {  //go left  curCode[curDepth] = 1;  curDepth++;  if (cur->right != NULL) stack->push(cur->right);  cur = cur->left;  }  else if (cur->right != NULL) {  //go right  curCode[curDepth] = 0;  curDepth++;  cur = cur->right;  }  else {  //write code  bool\* finalCode = new bool[curDepth];  for (int i = 0; i < curDepth; i++) finalCode[i] = curCode[i];  SymbCode\* Code = new SymbCode(finalCode, curDepth);  codes->insert((int)cur->symbols->get\_front(), Code);  //fck go back  while (stack->get\_top() != cur->right) {  cur = cur->parent;  curDepth--;  if (cur == nullptr) break;  }  cur = stack->pop();  curCode[curDepth] = 0;  curDepth++;  }  }  this->codes = codes;  }  bool\* SFCode::CodeString() {  if (root == nullptr) return NULL;  if (codedString != NULL) return codedString; //if we did this already  PriorityList<char>\* symb = new PriorityList<char>();  symb = root->symbols->Copy();    int prior, codeLength = 0; char curSymb;  //counting length of the final code  while (!symb->isEmpty()) {  prior = symb->head\_priority();  curSymb = symb->pop\_front();  SymbCode\* curSymbCode = codes->find(curSymb);  codeLength += prior\*curSymbCode->length;  }  this->codedStringLength = codeLength;  codedString = new bool[codeLength];  int curi = 0, j = 0;  //find code  for (int j = 0; j < inputStringLength; j++) {  curSymb = inputString[j];  SymbCode\* curSymbCode = codes->find(curSymb);  for (int i = 0; i < curSymbCode->length; i++) {  codedString[curi] = curSymbCode->code[i]; curi++;  }  }  return codedString;  }  char\* SFCode::DecodeString() {  if (codedString == NULL) return NULL;  int curLength = 0;  Node\* cur = root;  //assume string is the same lenght as code  //it can't be bigger  char\* string = new char[codedStringLength];  for (int i = 0; curLength < codedStringLength && i < codedStringLength;) {  while (!cur->ifLeaf) {  if (codedString[i] == 1) {  //go left;  cur = cur->left;  }  else {  //go right  cur = cur->right;  }  i++;  }  string[curLength] = cur->symbols->get\_front();  curLength++;  cur = root;  }  //if string is smaller than code  for (int i = 0; i < curLength + 1; i++) cout << string[i];  char\* finalString = new char[curLength + 1];  for (int i = 0; i < curLength + 1; i++) finalString[i] = string[i];  return finalString;  }  void SFCode::CodesOut() {  //codes->print();  char\* chars = codes->get\_keys();  SymbCode\*\* Symbs = codes->get\_values();  for (int i = 0; i < codes->get\_size(); i++)  {  cout << endl << chars[i] << " ";  for (int j = 0; j < Symbs[i]->length; j++)  cout << Symbs[i]->code[j];  }  } |
|  | **4sem2-test.cpp** |
|  | #include "pch.h"  #include "CppUnitTest.h"  #include"..\4sem2\SFCode.h"  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace My4sem2test  {  TEST\_CLASS(My4sem2test)  {  public:    TEST\_METHOD(EasyString)  {  char\* string = new char[5]{ 'h','e','l','l','o' };  SFCode\* test = new SFCode(string, 5);  bool\* code = test->CodeString();  char\* decode = test->DecodeString();  for (size\_t i = 0; i < 5; i++)  {  Assert::AreEqual(string[i], decode[i]);  }  }  TEST\_METHOD(OneLetter)  {  char\* string = new char[1]{ 'h' };  SFCode\* test = new SFCode(string, 1);  bool\* code = test->CodeString();  Assert::AreEqual(test->GetCodeLength(), 1);  char\* decode = test->DecodeString();  for (size\_t i = 0; i < 1; i++)  {  Assert::AreEqual(string[i], decode[i]);  }  }  TEST\_METHOD(FullOfSameLetter)  {  char\* string = new char[5]{ '!','!','!','!','!' };  SFCode\* test = new SFCode(string, 5);  bool\* code = test->CodeString();  Assert::AreEqual(test->GetCodeLength(), 5);  char\* decode = test->DecodeString();  for (size\_t i = 0; i < 5; i++)  {  Assert::AreEqual(string[i], decode[i]);  }  }  TEST\_METHOD(String1)  {  char\* string = new char[56]{ 'F','i','v','e',' ','e','x','c','l','a','m','a','t','i','o','n',' ','m','a','r','k','s',',',' ','t','h','e',' ','s','u','r','e',' ','s','i','g','n',' ','o','f',' ','a','n',' ','i','n','s','a','n','e',' ','m','i','n','d','.'};  SFCode\* test = new SFCode(string, 56);  bool\* code = test->CodeString();  char\* decode = test->DecodeString();  for (size\_t i = 0; i < 56; i++)  {  Assert::AreEqual(string[i], decode[i]);  }  }  TEST\_METHOD(String2)  {  char\* string = new char[10]{ 'a','o', 'a','o', 'a','o', 'a','o', 'a','o'};  SFCode\* test = new SFCode(string, 10);  bool\* code = test->CodeString();  char\* decode = test->DecodeString();  for (size\_t i = 0; i < 10; i++)  {  Assert::AreEqual(string[i], decode[i]);  }  }  TEST\_METHOD(String3)  {  char\* string = new char[16]{ 'E','a','r','t','h',' ','i','s',' ','a',' ','l','i','b','r','a' };  SFCode\* test = new SFCode(string, 16);  bool\* code = test->CodeString();  char\* decode = test->DecodeString();  for (size\_t i = 0; i < 16; i++)  {  Assert::AreEqual(string[i], decode[i]);  }  }  TEST\_METHOD(nullString)  {  SFCode\* test = new SFCode(NULL, 0);  bool\* code = test->CodeString();  Assert::AreEqual(NULL, test->DecodeString());  }  };  } |