**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ОТЧЁТ

**по лабораторной работе №2**

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

**Тема: «Алгоритмы кодирования»**

**Вариант 1.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Новокрещенов Д.К. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

## Постановка задачи и описание реализуемого класса и методов.

Для реализации алгоритма кодирования Хаффмана мне потребовалось реализовать класс Tree, а также использовать классы BiList и RedBlackTree из предыдущих работ.

struct Freq\_Syms

|  |  |
| --- | --- |
| **Компонент** | **Описание** |
| string value, code; | Сам символ и его код |
| unsigned amount = 0; | Частота встречаемости символа |

**classTree**

|  |  |  |
| --- | --- | --- |
| **Компонент** | **Описание** | |
| Tree\* left\_child; | Указатель на левое поддерево | |
| Tree\* right\_child; | Указатель на правое поддерево | |
| Freq\_Syms data; | Информация узла | |
| **Метод** | **Описание** | **Оценка временной сложности** |
| Tree\* build\_tree (BiList <Freq\_Syms> &); | Построение дерева Хаффмана | O(n) |
| Tree\* join(Tree\* attachable); | Объединение деревьев | O(1) |
| void get\_code (string, RedBlackTree<char, string>&); | Получение кода для символов | O(n) |
| void sort(BiList<Freq\_Syms>&); | Сортировка списка | O(n^2) |
| void sort(BiList<Tree\*>&); | Сортировка списка | O(n^2) |
| void print(BiList<Freq\_Syms>&); | Вывод таблицы частот | O(n) |
| string decoder(string); | Декодирование | O(n) |
| string Huffman(string); | Кодирование | O(n) |

## Описание реализованных unit-тестов.

|  |  |
| --- | --- |
| **Наименование теста** | **Описание** |
| simple | Проверка работы алгоритма на простом примере |
| medium | Проверка работы алгоритма на обычном примере |
| hard | Проверка работы алгоритма на сложном примере |

## Листинги программы и тестов

|  |
| --- |
| List.h |
| #pragma once  #include <cstddef>  template<class Type>  class BiList  {  public:  BiList();  ~BiList();  void push\_back(Type);  void push\_front(Type);  void pop\_back();  void pop\_front();  void insert(Type, size\_t);  Type at(size\_t);  void remove(size\_t);  size\_t get\_size();  void print\_to\_console();  void clear();  void set(size\_t, Type);  bool isEmpty();  void reverse();  private:  struct Node  {  Type data;  Node\* prev;  Node\* next;  };  Node\* head;  Node\* last;  size\_t size;  }; |
| List.cpp |
| #pragma once  #include "List.h"  #include <iostream>  template<class Type>  BiList<Type>::BiList()  {  head = NULL;  last = NULL;  size = 0;  }  template<class Type>  BiList<Type>::~BiList()  {  while (head)  {  last = head->next;  delete head;  head = last;  }  }  //The function inserts a new node at back of the list.  template<class Type>  void BiList<Type>::push\_back(Type value)  {  Node\* temp = new Node;  temp->next = NULL;  temp->data = value;  if (!this->isEmpty())  {  temp->prev = last;  last->next = temp;  last = temp;  }  else {  temp->prev = NULL;  head = last = temp;  }  size++;  }  //The function inserts a new node at front of the list.  template<class Type>  void BiList<Type>::push\_front(Type value)  {  Node\* temp = new Node;  temp->prev = NULL;  temp->data = value;  if (!this->isEmpty())  {  temp->next = head;  head->prev = temp;  head = temp;  }  else  {  temp->prev = NULL;  temp->next = NULL;  head = last = temp;  }  size++;  }  //The function deletes the last node in the list.  template<class Type>  void BiList<Type>::pop\_back()  {  if(size==0)  return;  if (size == 1)  {  delete last;  last = head = NULL;  size--;  return;  }  delete last->next;  last = last->prev;  last->next = NULL;  size--;  }  //The function deletes the first node in the list.  template<class Type>  void BiList<Type>::pop\_front()  {  if (size == 0)  return;  if (size == 1)  {  delete head;  head = last = NULL;  size--;  return;  }  delete head->prev;  head->prev = NULL;  head = head->next;  size--;  }  //The function inserts a new node after the node with the number "index".  template<class Type>  void BiList<Type>::insert(Type value, size\_t index)  {  if ((size == 0 && index==0) || index >= size)  throw "Incorrect index.";  if (index == 0)  {  this->push\_front(value);  return;  }  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  Node\* temp = new Node;  temp->data = value;  temp->next = cursor;  temp->prev = cursor->prev;  cursor->prev = temp;  temp->prev->next = temp;  size++;  }  //The function gets the value from the node with the number "index".  template<class Type>  Type BiList<Type>::at(size\_t index)  {  if (index >= size || index<0)  throw "Incorrect index.";  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  return cursor->data;  }  //The function deletes the node with the "index" number.  template<class Type>  void BiList<Type>::remove(size\_t index)  {  if (index >= size || index <= 0)  throw "Incorrect index.";  if (index == 0)  {  this->pop\_front();  return;  }  if (index == size-1)  {  this->pop\_back();  return;  }  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  Node\* temp = cursor;  cursor->prev->next = cursor->next;  cursor->next->prev = cursor->prev;  delete temp;  size--;  }  //The function gets the number of items in the list  template<class Type>  size\_t BiList<Type>::get\_size()  {  size\_t size\_of\_list = 0;  Node\* cursor = head;  if (cursor == NULL)  return 0;  while (cursor)  {  size\_of\_list++;  cursor = cursor->next;  }  return size\_of\_list;  }  //The function prTypes all list in console.  template<class Type>  void BiList<Type>::print\_to\_console()  {  Node\* cursor = head;  if (head)  {  while (cursor->next)  {  std::cout << cursor->data << "<-->";  cursor = cursor->next;  }  std::cout << cursor->data;  }  else throw "List is empty.";  }  //The function deletes all node in list.  template<class Type>  void BiList<Type>::clear()  {  while (head!=NULL)  this->pop\_front();  head = last = NULL;  }  //The function sets a new value in the node with the number "index"  template<class Type>  void BiList<Type>::set(size\_t index, Type value)  {  if (index >= size || index<0)  throw"Incorrect index.";  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  cursor->data = value;  }  //The function checks the list for emptiness.  template<class Type>  bool BiList<Type>::isEmpty()  {  if (head==NULL) return true;  return false;  }  //The function reverses a list.  template<class Type>  void BiList<Type>::reverse()  {  if (!head || !head->next)  return;  last = head;  Node\* temp = NULL;  Node\* current = head;  while (current != NULL)  {  temp = current->prev;  current->prev = current->next;  current->next = temp;  current = current->prev;  }  head = temp->prev;  } |
| RBTree.h |
| #pragma once  #include "List.h"  #include "List.cpp"  #define NULL 0  enum Color {RED, BLACK};  template<class Key\_Type, class Value\_Type>  class RedBlackTree  {  private:  class Node  {  public:  Key\_Type Key;  Value\_Type Value;  Node\* parent;  Node\* left;  Node\* right;  Color COLOR;  Node(Key\_Type Key, Value\_Type Value, Node\* parent = NULL, Node\* left = NULL, Node\* right = NULL, Color COLOR = RED);  ~Node();  };  Node\* root;  Node\* nil;  size\_t size;  void recovery(Node\*);  void leftRotate(Node\*);  void rightRotate(Node\*);  void recoveryRemove(Node\*);    void iteration\_over(Node\* actual, BiList<Key\_Type>\* keys, BiList<Value\_Type>\* values);  public:  RedBlackTree();  ~RedBlackTree();  void insert(Key\_Type, Value\_Type);  void remove(Key\_Type);  Value\_Type find(Key\_Type);  void clear();  BiList<Key\_Type>\* get\_keys();  BiList<Value\_Type>\* get\_values();  void print();  size\_t get\_size();  }; |
| RBTree.cpp |
| #pragma once  #include <iostream>  #include "RBTree.h"  #include "List.h"  using namespace std;  //конструктор для узла дерева  template<class Key\_Type, class Value\_Type>  RedBlackTree<Key\_Type, Value\_Type>::Node::Node(Key\_Type Key, Value\_Type Value, Node\* parent, Node\* left, Node\* right, Color COLOR)  {  this->Key = Key;  this->Value = Value;  this->left = left;  this->right = right;  this->parent = parent;  this->COLOR = COLOR;  }  //деструктор узла дерева  template<class Key\_Type, class Value\_Type>  RedBlackTree<Key\_Type, Value\_Type>::Node::~Node()  {  }  //восстановление после вставки  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::recovery(Node\* new\_Node)  {  Node\* parent, \* grandParent;  while (new\_Node->parent->COLOR == RED)  {  parent = new\_Node->parent;  grandParent = parent->parent;  if (grandParent->left == parent)  {  if (grandParent->right->COLOR == RED) //случай 1  {  grandParent->right->COLOR = BLACK;  grandParent->left->COLOR = BLACK;  grandParent->COLOR = RED;  new\_Node = grandParent;  }  else  {  if (parent->right == new\_Node) //случай 2  {  new\_Node = parent;  leftRotate(new\_Node);  }  //случай 3  new\_Node->parent->COLOR = BLACK;  new\_Node->parent->parent->COLOR = RED;  rightRotate(new\_Node->parent->parent);  }  }  else  {  if (grandParent->left && grandParent->left->COLOR == RED) //случай 1  {  grandParent->right->COLOR = BLACK;  grandParent->left->COLOR = BLACK;  grandParent->COLOR = RED;  new\_Node = grandParent;  }  else  {  if (parent->left == new\_Node) //случай 2  {  new\_Node = parent;  rightRotate(new\_Node);  }  //случай 3  new\_Node->parent->COLOR = BLACK;  new\_Node->parent->parent->COLOR = RED;  leftRotate(new\_Node->parent->parent);  }  }  }  root->COLOR = BLACK;  }  //левый поворот  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::leftRotate(Node\* actual)  {  if (actual->right == nil) return;  Node\* temp = actual->right;  if (temp->left != nil)  {  actual->right = temp->left;  temp->left->parent = actual;  }  else  actual->right = nil;  if (temp != nil)  temp->parent = actual->parent;  if (actual->parent != nil)  {  if (actual == actual->parent->left)  actual->parent->left = temp;  else  actual->parent->right = temp;  }  else  {  temp->parent = nil;  root = temp;  }  temp->left = actual;  if (actual != nil)  actual->parent = temp;  }  //правый поворот  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::rightRotate(Node\* actual)  {  Node\* temp = actual->left;  actual->left = temp->right;  if (temp->right != nil)  temp->right->parent = actual;  if (temp != nil)  temp->parent = actual->parent;  if (actual->parent != nil)  {  if (actual == actual->parent->right)  actual->parent->right = temp;  else  actual->parent->left = temp;  }  else  {  root = temp;  }  temp->right = actual;  if (actual != nil) actual->parent = temp;  }  //восстановление после удаления элемента  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::recoveryRemove(Node\* x\_Node)  {  Node\* brother;  while (x\_Node != root && x\_Node->COLOR == BLACK)  {  if (x\_Node == x\_Node->parent->left)  {  brother = x\_Node->parent->right;  if (brother->COLOR == RED)  {  brother->COLOR = BLACK;  x\_Node->parent->COLOR = RED;  leftRotate(x\_Node->parent);  brother = x\_Node->parent->right;  }  if (brother->left->COLOR == BLACK && brother->right->COLOR == BLACK)  {  brother->COLOR = RED;  x\_Node = x\_Node->parent;  }  else  {  if (brother->right->COLOR == BLACK)  {  brother->left->COLOR = BLACK;  brother->COLOR = RED;  rightRotate(brother);  brother = x\_Node->parent->right;  }  brother->COLOR = x\_Node->parent->COLOR;  x\_Node->parent->COLOR = BLACK;  brother->right->COLOR = BLACK;  leftRotate(x\_Node->parent);  x\_Node = root;  }  }  else  {  brother = x\_Node->parent->left;  if (brother->COLOR == RED)  {  brother->COLOR = BLACK;  x\_Node->parent->COLOR = RED;  rightRotate(x\_Node->parent);  brother = x\_Node->parent->left;  }  if (brother->right->COLOR == BLACK && brother->left->COLOR == BLACK)  {  brother->COLOR = RED;  x\_Node = x\_Node->parent;  }  else  {  if (brother->left->COLOR == BLACK)  {  brother->right->COLOR = BLACK;  brother->COLOR = RED;  leftRotate(brother);  brother = x\_Node->parent->left;  }  brother->COLOR = x\_Node->parent->COLOR;  x\_Node->parent->COLOR = BLACK;  brother->left->COLOR = BLACK;  rightRotate(x\_Node->parent);  x\_Node = root;  }  }  }  x\_Node->COLOR = BLACK;  }  //конструктор для пустого дерева  template<class Key\_Type, class Value\_Type>  RedBlackTree<Key\_Type, Value\_Type>::RedBlackTree()  {  if(typeid(Value\_Type).name() == typeid(string).name())  nil = new Node(0, " ", NULL, NULL, NULL, BLACK);  else  nil = new Node(0, 0, NULL, NULL, NULL, BLACK);  root = nil;  size = 0;  }  //деструктор для дерева  template<class Key\_Type, class Value\_Type>  RedBlackTree<Key\_Type, Value\_Type>::~RedBlackTree()  {  clear();  }  //вставка нового узла  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::insert(Key\_Type Key, Value\_Type Value)  {  Node\* new\_Node = new Node(Key, Value, nil, nil, nil);  if (size == 0)  {  root = new\_Node;  root->COLOR = BLACK;  size++;  return;  }  Node\* temp = root;  new\_Node->COLOR = RED;  while (1)  {  if (Key > temp->Key)  {  if (temp->right == nil)  {  temp->right = new\_Node;  new\_Node->parent = temp;  break;  }  temp = temp->right;  }  else  {  if (temp->left == nil)  {  temp->left = new\_Node;  new\_Node->parent = temp;  break;  }  temp = temp->left;  }  }  recovery(new\_Node);  size++;  }  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::remove(Key\_Type Key)  {  if (root == nil)  throw "Tree is empty";  Node\* NodeToDelete = root;  while (NodeToDelete->Key != Key) //поиск элемента  {  if (Key > NodeToDelete->Key)  NodeToDelete = NodeToDelete->right;  else  NodeToDelete = NodeToDelete->left;  if (NodeToDelete == nil)  throw "No such element";  }  Node\* x\_Node, \*w\_Node;  Color origCOLOR = NodeToDelete->COLOR;  if (NodeToDelete->left == nil)  {  x\_Node = NodeToDelete->right;  if (NodeToDelete->parent == nil)  root = x\_Node;  else if (NodeToDelete->parent->left == NodeToDelete)  NodeToDelete->parent->left = x\_Node;  else  NodeToDelete->parent->right = x\_Node;  x\_Node->parent = NodeToDelete->parent;  }  else if (NodeToDelete->right == nil)  {  x\_Node = NodeToDelete->left;  if (NodeToDelete->parent == nil)  root = x\_Node;  else if (NodeToDelete->parent->left == NodeToDelete)  NodeToDelete->parent->left = x\_Node;  else  NodeToDelete->parent->right = x\_Node;  x\_Node->parent = NodeToDelete->parent;  }  else  {  Node\* temp = NodeToDelete;  w\_Node = NodeToDelete->right;  while (w\_Node->left != nil)  w\_Node = w\_Node->left;  origCOLOR = w\_Node->COLOR;  NodeToDelete = w\_Node;  x\_Node = NodeToDelete->right;  if (NodeToDelete->parent == temp)  x\_Node->parent = NodeToDelete;  else  {  if (NodeToDelete->parent == nil)  {  root = x\_Node;  NodeToDelete->right->parent = nil;  }  else if (NodeToDelete->parent->right == NodeToDelete)  NodeToDelete->parent->right = x\_Node;  else  NodeToDelete->parent->left = x\_Node;  x\_Node->parent = NodeToDelete->parent;  NodeToDelete->right = temp->right;  NodeToDelete->right->parent = NodeToDelete;  }  if (temp->parent == nil)  root = NodeToDelete;  else if (temp == temp->parent->left)  temp->parent->left = NodeToDelete;  else  temp->parent->right = NodeToDelete;  NodeToDelete->parent = temp->parent;  NodeToDelete->left = temp->left;  NodeToDelete->left->parent = NodeToDelete;  NodeToDelete->COLOR = temp->COLOR;  }  if (origCOLOR == BLACK)  recoveryRemove(x\_Node);  size--;  }  template<class Key\_Type, class Value\_Type>  Value\_Type RedBlackTree<Key\_Type, Value\_Type>::find(Key\_Type Key)  {  if (root == nil)  throw "Tree is empty";  Node\* nodeToFind = root;  while (nodeToFind->Key != Key) //поиск элемента  {  if (Key > nodeToFind->Key)  nodeToFind = nodeToFind->right;  else  nodeToFind = nodeToFind->left;  if (nodeToFind == nil)  throw "No such element";  }  return nodeToFind->Value;  }  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::iteration\_over(Node\* actual, BiList<Key\_Type> \*keys, BiList<Value\_Type> \*values)  {  if (keys == NULL)  {  if (actual->left!=nil)  iteration\_over(actual->left, NULL, values);  if (actual->right!=nil)  iteration\_over(actual->right, NULL, values);  values->push\_back(actual->Value);  }  else  {  if (actual->left!=nil)  iteration\_over(actual->left, keys, NULL);  if (actual->right!=nil)  iteration\_over(actual->right, keys, NULL);  keys->push\_back(actual->Key);  }  }  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::clear()  {  while (root != nil)  remove(root->Key);  }  template<class Key\_Type, class Value\_Type>  BiList<Key\_Type>\* RedBlackTree<Key\_Type, Value\_Type>::get\_keys()  {  BiList<Key\_Type>\* result = new BiList<Key\_Type>;  iteration\_over(root, result, NULL);  return result;  }  template<class Key\_Type, class Value\_Type>  BiList<Value\_Type>\* RedBlackTree<Key\_Type, Value\_Type>::get\_values()  {  BiList<Value\_Type>\* result = new BiList<Value\_Type>;  iteration\_over(root, NULL, result);  return result;  }  template<class Key\_Type, class Value\_Type>  void RedBlackTree<Key\_Type, Value\_Type>::print()  {  BiList<Key\_Type>\* keys = get\_keys();  BiList<Value\_Type>\* values = get\_values();  for (size\_t i = 0; i < size; i++)  cout << keys->at(i) << "||" << values->at(i) << endl;  cout << endl;  }  template<class Key\_Type, class Value\_Type>  size\_t RedBlackTree<Key\_Type, Value\_Type>::get\_size()  {  return size;  } |
| BinaryTree.h |
| #pragma once  #include <string>  #include "RBTree.h"  #include "List.h"  using namespace std;  struct Freq\_Syms {  string value, code;  unsigned amount = 0;  };  class Tree  {  private:  Tree\* left\_child;  Tree\* right\_child;  Freq\_Syms data;    Tree\* build\_tree(BiList<Freq\_Syms>&);  Tree\* join(Tree\* attachable);  void get\_code(string, RedBlackTree<char, string>&);  void sort(BiList<Freq\_Syms>&);  void sort(BiList<Tree\*>&);  void print(BiList<Freq\_Syms>&);  string decoder(string);  public:  Tree(Freq\_Syms data = {"","",0}, Tree\* left = NULL, Tree\* right = NULL);  ~Tree();  string Huffman(string);  }; |
| BinaryTree.cpp |
| #pragma once  #include "BinaryTree.h"  #include "RBTree.h"  #include "RBTree.cpp"  Tree::Tree(Freq\_Syms data, Tree\* left, Tree\* right)  {  this->data = data;  left\_child = left;  right\_child = right;  }  Tree::~Tree()  {  }  Tree\* Tree::build\_tree(BiList<Freq\_Syms>& list\_of\_syms)  {  BiList<Tree\*> new\_tree;  for (size\_t i = 0; i < list\_of\_syms.get\_size(); i++)  {  Freq\_Syms temp = list\_of\_syms.at(i);  new\_tree.push\_back(new Tree(temp));  }  while (new\_tree.get\_size() > 1)  {  Tree\* n = new\_tree.at(0);  new\_tree.pop\_front();  new\_tree.push\_front(n->join(new\_tree.at(0)));  new\_tree.remove(1);  sort(new\_tree);  }  return new\_tree.at(0);  }  Tree\* Tree::join(Tree\* attachable)  {  Freq\_Syms new\_data = {this->data.value + attachable->data.value, "" ,this->data.amount + attachable->data.amount};  return new Tree(new\_data, this, attachable);  }  void Tree::get\_code(string code, RedBlackTree<char, string>& map)  {  this->data.code = code;  if (left\_child != 0)  {  left\_child->get\_code((code + "0"), map);  }  if (right\_child != 0)  {  right\_child->get\_code((code + "1"), map);  }  if (left\_child==NULL && right\_child==NULL)  {  map.insert(this->data.value[0], code);  return;  }  }  string Tree::Huffman(string message)  {  BiList<Freq\_Syms> freq\_table;  //Подсчёт символов  for (size\_t i = 0; i < message.size(); i++)  {  string value;  value.push\_back(message[i]);  bool check = true;  for (size\_t j = 0; j < freq\_table.get\_size(); j++)  {  if (freq\_table.get\_size() == 0)  break;  Freq\_Syms temp = freq\_table.at(j);  if (temp.value == value)  {  temp.amount++;  freq\_table.set(j, temp);  check = false;  break;  }  }  if (check)  {  freq\_table.push\_back({ value, "", 1 });  }  }  sort(freq\_table); print(freq\_table);  //Построение бинарного дерева Хаффмана  Tree\* code = build\_tree(freq\_table);  //Ассоциативный массив символ код  RedBlackTree<char, string> map;  code->get\_code("", map);  cout << endl;  map.print();  //Кодирование  string result, result\_real;  for (size\_t i = 0; i < message.size(); i++)  {  result = result + map.find(message[i]) + " ";  result\_real = result\_real + map.find(message[i]);  }  cout << endl << result << endl;  cout << "Исходное сообщение: " << message.size() \* 8 << ", закодированное: " << result.size() - message.size() << endl;  cout << "Результат декодирования: ";  //Декодирование по дереву Хаффмана  code->decoder(result\_real);  return result\_real;  }  void Tree::sort(BiList<Freq\_Syms> &freq\_table) //Простая сортировка пузырьком  {  bool check = true;  while (check)  {  for (size\_t i = 0; i < freq\_table.get\_size()-1; i++)  {  if (freq\_table.at(i).amount > freq\_table.at(i + 1).amount)  {  Freq\_Syms temp = freq\_table.at(i);  freq\_table.set(i, freq\_table.at(i + 1));  freq\_table.set(i + 1, temp);  check = false;  }  }  if (!check)  check = true;  else check = false;  }  }  void Tree::sort(BiList<Tree\*>& freq\_table) //Простая сортировка пузырьком  {  bool check = true;  while (check)  {  for (size\_t i = 0; i < freq\_table.get\_size() - 1; i++)  {  if (freq\_table.at(i)->data.amount > freq\_table.at(i + 1)->data.amount)  {  Tree\* temp = freq\_table.at(i);  freq\_table.set(i, freq\_table.at(i + 1));  freq\_table.set(i + 1, temp);  check = false;  }  }  if (!check)  check = true;  else check = false;  }  }  string Tree::decoder(string message)  {  string result;  Tree\* temp = this;  size\_t i = 0;  while (i!=message.size()-1)  {  Tree\* cursor = temp;  while (cursor->left\_child!=0 && cursor->right\_child!=0)  {  if (message[i] == '1')  cursor = cursor->right\_child;  else  cursor = cursor->left\_child;  i++;  }  result = result + cursor->data.value;  }  cout << result << endl;  return result;  }  void Tree::print(BiList<Freq\_Syms>& table)  {  for (size\_t i = 0; i < table.get\_size(); i++)  {  Freq\_Syms print = table.at(i);  cout << "Value: " << print.value << " Amount: " << print.amount << endl;  }  } |
| UnitRBTreeTest.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "../Lab2/BinaryTree.cpp"  #include "../Lab2/List.cpp"  #include "../Lab2/RBTree.cpp"  using namespace std;  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UnitTest  {  TEST\_CLASS(BiListTest)  {  public:  BiList<int> Empty;  BiList<int> NotEmpty;  TEST\_METHOD\_INITIALIZE(SetUp)  {  for (size\_t i = 0; i < 3; i++)  NotEmpty.push\_back(i + 1);  }  TEST\_METHOD(IsEmpty\_for\_NotEmpty)  {  Assert::AreEqual(NotEmpty.isEmpty(), false);  }  TEST\_METHOD(IsEmpty\_for\_Empty)  {  Assert::AreEqual(Empty.isEmpty(), true);  }  TEST\_METHOD(Get\_Size\_for\_Empty)  {  size\_t real\_size = 0;  Assert::AreEqual(Empty.get\_size(), real\_size);  }  TEST\_METHOD(Get\_Size\_for\_NotEmpty)  {  size\_t real\_size = 3;  Assert::AreEqual(NotEmpty.get\_size(), real\_size);  }  TEST\_METHOD(at\_correct\_index)  {  Assert::AreEqual(NotEmpty.at(2), 3);  }  TEST\_METHOD(at\_incorrect\_index)  {  try  {  Empty.at(4);  }  catch (const char\* error)  {  Assert::AreEqual(error, "Incorrect index.");  }  }  TEST\_METHOD(Push\_Back\_NotEmpty)  {  NotEmpty.push\_back(5);  Assert::AreEqual(NotEmpty.at(3), 5);  }  TEST\_METHOD(Push\_Front\_NotEmpty)  {  NotEmpty.push\_front(0);  Assert::AreEqual(NotEmpty.at(0), 0);  }  TEST\_METHOD(remove\_correct\_index)  {  int temp = NotEmpty.at(2);  NotEmpty.remove(1);  Assert::AreEqual(NotEmpty.at(1), temp);  }  TEST\_METHOD(Pop\_Back\_for\_NotEmpty)  {  size\_t real\_size = NotEmpty.get\_size() - 1;  NotEmpty.pop\_back();  Assert::AreEqual(NotEmpty.get\_size(), real\_size);  }  TEST\_METHOD(Pop\_Back\_for\_Empty)  {  Empty.pop\_back();  Assert::AreEqual(Empty.isEmpty(), true);  }  TEST\_METHOD(Pop\_Front\_for\_NotEmpty)  {  size\_t real\_size = NotEmpty.get\_size() - 1;  NotEmpty.pop\_front();  Assert::AreEqual(NotEmpty.get\_size(), real\_size);  }  TEST\_METHOD(insert\_correct\_index)  {  NotEmpty.insert(4, 1);  Assert::AreEqual(NotEmpty.at(1), 4);  }  TEST\_METHOD(insert\_incorrect\_index)  {  try  {  Empty.insert(2, 0);  }  catch (const char\* error)  {  Assert::AreEqual("Incorrect index.", error);  }  }  TEST\_METHOD(remove\_incorrect\_index)  {  try  {  Empty.remove(2);  }  catch (const char\* error)  {  Assert::AreEqual("Incorrect index.", error);  }  }  TEST\_METHOD(reverse\_for\_NotEmpty)  {  int last\_inf = NotEmpty.at(NotEmpty.get\_size() - 1);  NotEmpty.reverse();  Assert::AreEqual(NotEmpty.at(0), last\_inf);  }  TEST\_METHOD(Pop\_Front\_for\_Empty)  {  Empty.pop\_front();  Assert::AreEqual(Empty.isEmpty(), true);  }  TEST\_METHOD(set\_correct\_index)  {  NotEmpty.set(2, 3);  Assert::AreEqual(NotEmpty.at(2), 3);  }  TEST\_METHOD(set\_incorrect\_index)  {  try  {  NotEmpty.set(10000000, 2);  }  catch (const char\* error)  {  Assert::AreEqual(error, "Incorrect index.");  }  }  TEST\_METHOD(Test\_Clear\_NotEmpty)  {  NotEmpty.clear();  size\_t real\_size = 0;  Assert::AreEqual(NotEmpty.get\_size(), real\_size);  }  TEST\_METHOD(Test\_Clear\_Empty)  {  Empty.clear();  size\_t real\_size = 0;  Assert::AreEqual(Empty.get\_size(), real\_size);  }  TEST\_METHOD(Push\_Back\_Empty)  {  Empty.push\_back(5);  Assert::AreEqual(Empty.at(0), 5);  Empty.pop\_back();  }  TEST\_METHOD(Push\_Front\_Empty)  {  Empty.push\_front(0);  Assert::AreEqual(Empty.at(0), 0);  }  };  TEST\_CLASS(UnitRBTreeTest)  {  public:  RedBlackTree<int, string> TestTree;  TEST\_METHOD\_INITIALIZE(SetUp)  {  int keys[10] = { 12,-11,45,76,34,-9,15,-87,97,111 };  string values[10] = { "cat","dog" ,"pig" ,"Kolpakov" ,"tree" ,"house" ,"beat" ,"meat" ,"SOS" ,"stackoverflow" };  for (size\_t i = 0; i < 10; i++)  TestTree.insert(keys[i], values[i]);  }  TEST\_METHOD(Search\_real)  {  string answer = "Kolpakov";  Assert::AreEqual(TestTree.find(76), answer);  }  TEST\_METHOD(Extra\_Search)  {  int keys[10] = { 12,-11,45,76,34,-9,15,-87,97,111 };  string values[10] = { "cat","dog" ,"pig" ,"Kolpakov" ,"tree" ,"house" ,"beat" ,"meat" ,"SOS" ,"stackoverflow" };  for (size\_t i = 0; i < 10; i++)  Assert::AreEqual(TestTree.find(keys[i]), values[i]);  }  TEST\_METHOD(Search\_unreal)  {  try {  TestTree.find(2020);  }  catch (const char\* error)  {  Assert::AreEqual(error, "No such element");  }  }  TEST\_METHOD(insert\_test)  {  string answer = "SPb";  TestTree.insert(1, answer);  Assert::AreEqual(TestTree.find(1), answer);  }  TEST\_METHOD(remove\_test\_real)  {  try {  string answer = "SPb";  TestTree.remove(1);  TestTree.find(1);  }  catch (const char\* error)  {  Assert::AreEqual(error, "No such element");  }  }  TEST\_METHOD(get\_keys\_test)  {  BiList<int>\* keys = TestTree.get\_keys();  try {  for (size\_t i = 0; i < 10; i++)  {  TestTree.remove(keys->at(i));  }  }  catch (const char\* error)  {  Assert::AreEqual(1, 0);  }  Assert::AreEqual(TestTree.get\_size(), (size\_t)0);  }  TEST\_METHOD(get\_value\_test)  {  size\_t check = 0;  string value[10] = { "cat","dog" ,"pig" ,"Kolpakov" ,"tree" ,"house" ,"beat" ,"meat" ,"SOS" ,"stackoverflow" };  BiList<string>\* values = TestTree.get\_values();  for (size\_t i = 0; i < 10; i++)  {  string temp = values->at(i);  for (size\_t j = 0; j < 10; j++)  {  if (temp == value[j])  check++;  }  }  Assert::AreEqual(check, (size\_t)10);  }  TEST\_METHOD(clear\_test)  {  TestTree.clear();  Assert::AreEqual(TestTree.get\_size(), (size\_t)0);  }  };  TEST\_CLASS(Huffman)  {  TEST\_METHOD(Simple)  {  Tree\* Test = new Tree();  string test = "abc", result = Test->Huffman(test), check = "10110";  Assert::AreEqual(result, check);  }  TEST\_METHOD(Medium)  {  Tree\* Test = new Tree();  string test = "wasdwasdwasd", result = Test->Huffman(test), check = "101100011011000110110001";  Assert::AreEqual(result, check);  }  TEST\_METHOD(Hard)  {  Tree\* Test = new Tree();  string test = "========== cборка: успешно: 0 ==========", result = Test->Huffman(test), check = "11111111110100011000111000000100001010110100001010011011011000011001011110011111011100000000010100111010101111111111";  Assert::AreEqual(result, check);  }  };  } |

## Примеры работы программы

|  |
| --- |
| Мальчик Дзиро мечтает о полетах и красивых самолетах, способных обогнать ветер. Вот только пилотом ему не стать - он с рождения близорук. Но Дзиро не расстается с мечтой о небе, он начинает придумывать идеальный самолет и со временем становится одним из лучших авиаконструкторов мира. На пути к успеху он не только встретит много интересных людей, переживет Великое землетрясение в Токио и жестокие войны, но и обретет любовь своей жизни - прекрасную Наоко. |
| Value: М Amount: 1  Value: ш Amount: 1  Value: Т Amount: 1  Value: Д Amount: 2  Value: г Amount: 2  Value: В Amount: 2  Value: - Amount: 2  Value: Н Amount: 3  Value: ю Amount: 3  Value: , Amount: 4  Value: . Amount: 4  Value: ж Amount: 4  Value: я Amount: 4  Value: ч Amount: 5  Value: д Amount: 5  Value: й Amount: 5  Value: з Amount: 6  Value: ы Amount: 6  Value: б Amount: 6  Value: х Amount: 7  Value: ь Amount: 8  Value: п Amount: 8  Value: у Amount: 9  Value: к Amount: 13  Value: м Amount: 13  Value: в Amount: 13  Value: л Amount: 14  Value: р Amount: 19  Value: а Amount: 22  Value: с Amount: 22  Value: н Amount: 25  Value: и Amount: 30  Value: т Amount: 32  Value: е Amount: 41  Value: о Amount: 44  Value: Amount: 71  Д||10101100  В||10000010  Н||0101001  М||010100010  а||0001  Т||01010000  г||10101101  в||01111  е||1111  ж||1010101  и||1001  з||010101  д||000000  б||000011  к||01011  м||01110  л||10001  о||001  р||11101  п||101001  н||0110  у||111001  т||1011  ч||1110001  ш||010100011  х||100001  ь||101000  ы||000010  я||1110000  ,||1010111  ||110  .||1010100  -||10000011  ю||1000000  с||0100  й||000001  010100010 0001 10001 101000 1110001 1001 01011 110 10101100 010101 1001 11101 001 110 01110 1111 1110001 1011 0001 1111 1011 110 001 110 101001 001 10001 1111 1011 0001 100001 110 1001 110 01011 11101 0001 0100 1001 01111 000010 100001 110 0100 0001 01110 001 10001 1111 1011 0001 100001 1010111 110 0100 101001 001 0100 001 000011 0110 000010 100001 110 001 000011 001 10101101 0110 0001 1011 101000 110 01111 1111 1011 1111 11101 1010100 110 10000010 001 1011 110 1011 001 10001 101000 01011 001 110 101001 1001 10001 001 1011 001 01110 110 1111 01110 111001 110 0110 1111 110 0100 1011 0001 1011 101000 110 10000011 110 001 0110 110 0100 110 11101 001 1010101 000000 1111 0110 1001 1110000 110 000011 10001 1001 010101 001 11101 111001 01011 1010100 110 0101001 001 110 10101100 010101 1001 11101 001 110 0110 1111 110 11101 0001 0100 0100 1011 0001 1111 1011 0100 1110000 110 0100 110 01110 1111 1110001 1011 001 000001 110 001 110 0110 1111 000011 1111 1010111 110 001 0110 110 0110 0001 1110001 1001 0110 0001 1111 1011 110 101001 11101 1001 000000 111001 01110 000010 01111 0001 1011 101000 110 1001 000000 1111 0001 10001 101000 0110 000010 000001 110 0100 0001 01110 001 10001 1111 1011 110 1001 110 0100 001 110 01111 11101 1111 01110 1111 0110 1111 01110 110 0100 1011 0001 0110 001 01111 1001 1011 0100 1110000 110 001 000000 0110 1001 01110 110 1001 010101 110 10001 111001 1110001 010100011 1001 100001 110 0001 01111 1001 0001 01011 001 0110 0100 1011 11101 111001 01011 1011 001 11101 001 01111 110 01110 1001 11101 0001 1010100 110 0101001 0001 110 101001 111001 1011 1001 110 01011 110 111001 0100 101001 1111 100001 111001 110 001 0110 110 0110 1111 110 1011 001 10001 101000 01011 001 110 01111 0100 1011 11101 1111 1011 1001 1011 110 01110 0110 001 10101101 001 110 1001 0110 1011 1111 11101 1111 0100 0110 000010 100001 110 10001 1000000 000000 1111 000001 1010111 110 101001 1111 11101 1111 1010101 1001 01111 1111 1011 110 10000010 1111 10001 1001 01011 001 1111 110 010101 1111 01110 10001 1111 1011 11101 1110000 0100 1111 0110 1001 1111 110 01111 110 01010000 001 01011 1001 001 110 1001 110 1010101 1111 0100 1011 001 01011 1001 1111 110 01111 001 000001 0110 000010 1010111 110 0110 001 110 1001 110 001 000011 11101 1111 1011 1111 1011 110 10001 1000000 000011 001 01111 101000 110 0100 01111 001 1111 000001 110 1010101 1001 010101 0110 1001 110 10000011 110 101001 11101 1111 01011 11101 0001 0100 0110 111001 1000000 110 0101001 0001 001 01011 001 1010100  Исходное сообщение: 3656, закодированное: 2032  Сжатие: 55.5799  Результат декодирования: Мальчик Дзиро мечтает о полетах и красивых самолетах, способных обогнать ветер. Вот только пилотом ему не стать - он с рождения близорук. Но Дзиро не расстается с мечтой о небе, он начинает придумывать идеальный самолет и со временем становится одним из лучших авиаконструкторов мира. На пути к успеху он не только встретит много интересных людей, переживет Великое землетрясение в Токио и жестокие войны, но и обретет любовь своей жизни - прекрасную Наоко. |
| Планирую переводиться в ваш университет, расскажите кто уже переводился, как сам процесс перевода проходит? Что надо знать перед тем как перевестись? анонимно пожалуйста |
| Value: П Amount: 1  Value: ю Amount: 1  Value: ш Amount: 1  Value: ц Amount: 1  Value: х Amount: 1  Value: Ч Amount: 1  Value: з Amount: 1  Value: й Amount: 1  Value: я Amount: 2  Value: , Amount: 2  Value: ? Amount: 2  Value: л Amount: 3  Value: ь Amount: 3  Value: ж Amount: 3  Value: м Amount: 3  Value: у Amount: 4  Value: д Amount: 6  Value: к Amount: 6  Value: н Amount: 7  Value: в Amount: 7  Value: п Amount: 8  Value: и Amount: 9  Value: р Amount: 10  Value: т Amount: 11  Value: с Amount: 11  Value: о Amount: 12  Value: а Amount: 13  Value: е Amount: 17  Value: Amount: 22  П||0000010  а||1110  д||10111  в||11111  ж||101101  е||001  Ч||0000111  и||0101  к||11010  м||110110  о||1100  н||11110  л||010011  й||0000101  з||0000100  р||0110  т||0111  с||1010  х||0000110  у||01000  ь||101100  ш||0000000  я||1101110  ,||1101111  ?||010010  ||100  ю||0000011  ц||0000001  п||0001  0000010 010011 1110 11110 0101 0110 01000 0000011 100 0001 001 0110 001 11111 1100 10111 0101 0111 101100 1010 1101110 100 11111 100 11111 1110 0000000 100 01000 11110 0101 11111 001 0110 1010 0101 0111 001 0111 1101111 100 0110 1110 1010 1010 11010 1110 101101 0101 0111 001 100 11010 0111 1100 100 01000 101101 001 100 0001 001 0110 001 11111 1100 10111 0101 010011 1010 1101110 1101111 100 11010 1110 11010 100 1010 1110 110110 100 0001 0110 1100 0000001 001 1010 1010 100 0001 001 0110 001 11111 1100 10111 1110 100 0001 0110 1100 0000110 1100 10111 0101 0111 010010 100 0000111 0111 1100 100 11110 1110 10111 1100 100 0000100 11110 1110 0111 101100 100 0001 001 0110 001 10111 100 0111 001 110110 100 11010 1110 11010 100 0001 001 0110 001 11111 001 1010 0111 0101 1010 101100 010010 100 1110 11110 1100 11110 0101 110110 11110 1100 100 0001 1100 101101 1110 010011 01000 0000101 1010 0111 1110  Исходное сообщение: 1352, закодированное: 731  Сжатие: 54.068  Результат декодирования: Планирую переводиться в ваш университет, расскажите кто уже переводился, как сам процесс перевода проходит? Что надо знать перед тем как перевестись? анонимно пожалуйста |
| Реализовать кодирование и декодирование по алгоритму Хаффмана / Шеннона-Фано (1 и 2 вариант соответственно) входной строки, вводимой через консоль |
| Value: Р Amount: 1  Value: п Amount: 1  Value: г Amount: 1  Value: у Amount: 1  Value: Х Amount: 1  Value: / Amount: 1  Value: Ш Amount: 1  Value: - Amount: 1  Value: Ф Amount: 1  Value: ( Amount: 1  Value: 1 Amount: 1  Value: 2 Amount: 1  Value: ) Amount: 1  Value: х Amount: 1  Value: , Amount: 1  Value: ч Amount: 1  Value: з Amount: 2  Value: ь Amount: 2  Value: ф Amount: 2  Value: й Amount: 2  Value: л Amount: 3  Value: м Amount: 3  Value: к Amount: 4  Value: с Amount: 4  Value: д Amount: 5  Value: р Amount: 6  Value: т Amount: 7  Value: е Amount: 9  Value: в Amount: 9  Value: и Amount: 11  Value: а Amount: 12  Value: н Amount: 12  Value: Amount: 18  Value: о Amount: 19  Ф||0101010  Р||0010010  Ш||0010100  в||0111  а||1110  Х||0010110  д||11000  и||1101  з||001110  е||0110  к||01000  н||1111  м||110011  л||110010  п||0010011  о||101  й||001101  г||0010000  с||01001  т||0001  ф||001100  ч||0101101  х||0101111  у||0010001  ||100  (||0101011  ,||0101100  )||0101110  /||0010111  2||0101001  1||0101000  -||0010101  ь||001111  р||0000  0010010 0110 1110 110010 1101 001110 101 0111 1110 0001 001111 100 01000 101 11000 1101 0000 101 0111 1110 1111 1101 0110 100 1101 100 11000 0110 01000 101 11000 1101 0000 101 0111 1110 1111 1101 0110 100 0010011 101 100 1110 110010 0010000 101 0000 1101 0001 110011 0010001 100 0010110 1110 001100 001100 110011 1110 1111 1110 100 0010111 100 0010100 0110 1111 1111 101 1111 1110 0010101 0101010 1110 1111 101 100 0101011 0101000 100 1101 100 0101001 100 0111 1110 0000 1101 1110 1111 0001 100 01001 101 101 0001 0111 0110 0001 01001 0001 0111 0110 1111 1111 101 0101110 100 0111 0101111 101 11000 1111 101 001101 100 01001 0001 0000 101 01000 1101 0101100 100 0111 0111 101 11000 1101 110011 101 001101 100 0101101 0110 0000 0110 001110 100 01000 101 1111 01001 101 110010 001111  Исходное сообщение: 1168, закодированное: 636  Сжатие: 54.4521  Результат декодирования: Реализовать кодирование и декодирование по алгоритму Хаффмана / Шеннона-Фано (1 и 2 вариант соответственно) входной строки, вводимой через консоль |

## Вывод

В ходе выполнения работы я реализовал алгоритм кодирования Хаффмана.