**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.

**Class Graph**

|  |  |  |
| --- | --- | --- |
| **Компонент** | **Описание** | |
| BiList<string> Info; | Список, где по строкам хранится информация текстового файла | |
| BiList<string> Node; | Список, где хранится имя узлов графа | |
| int\*\* matrix; | Указатель на матрицу смежности | |
| size\_t size; | Размер матрицы смежности (size\*size) | |
| **Метод** | **Описание** | **Оценка временной сложности** |
| void input(string s); | Ввод информации из файла | О(n) |
| void summarize(); | Создание списка имён узлов и матрицы смежности | O() |
| size\_t BellFord(size\_t, size\_t); | Алгоритм Беллмана-Форда для поиска кратчайшего пути из i в j | O() |
| void clear(); | Удаление информации | O(n) |

## Листинги программы и тестов

|  |
| --- |
| List.h |
| #pragma once  #include <cstddef>  #include <string>  using namespace std;  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();  void swap(size\_t, size\_t);  int find(Type);  private:  struct Node  {  Type data;  Node\* prev = NULL;  Node\* next = NULL;  };  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()  {  return size;  }  //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;  }  template<class Type>  void BiList<Type>::swap(size\_t first, size\_t second)  {  Node\* temp\_f = head, \* temp\_s = head;  for (size\_t i = 0; i < first; i++)  temp\_f = temp\_f->next;  for (size\_t i = 0; i < second; i++)  temp\_s = temp\_s->next;  Type temp = temp\_f->data;  temp\_f->data = temp\_s->data;  temp\_s->data = temp;  }  template<class Type>  int BiList<Type>::find(Type info)  {  Node\* cur = head;  int i = 0;  while (i < size)  {  if (cur->data == info)  return i;  cur = cur->next;  i++;  }  return -1;  } |
| BellFord.h |
| #pragma once  #include <fstream>  #include "List.cpp"  class Graph  {  private:  BiList<string> Node, Info;  int\*\* matrix;  size\_t size;  void summarize();  public:  Graph();  ~Graph();  void input(string filename = "test.txt");  size\_t BellFord(size\_t, size\_t);  void clear();  }; |
| BellFord.cpp |
| #include "BellFord.h"  #include <iostream>  #define inf INT\_MAX/2-1  using namespace std;  Graph::Graph()  {  matrix = NULL;  size = 0;  }  Graph::~Graph()  {  clear();  }  void Graph::input(string filename)  {  Info.clear(); Node.clear();  if (filename == "")  filename = "test.txt";  string temp;  ifstream filereader;  filereader.open(filename.c\_str(), ios::in);  if (filereader.bad())  throw "Bad File Name";  while (!filereader.eof())  {  getline(filereader, temp);  Info.push\_back(temp);  }  summarize();  }  size\_t Graph::BellFord(size\_t first, size\_t second)  {  if (first == second)  return 0;  int\* dist = new int[Node.get\_size()];  int\* way = new int[Node.get\_size()];  for (size\_t i = 0; i < Node.get\_size(); i++)  {  if (i != first) dist[i] = matrix[first][i];  else dist[i] = 0;  way[i] = -1;  }  for (size\_t k = 0; k < Node.get\_size(); k++)  {  bool control = false;  for (size\_t i = 0; i < Node.get\_size(); i++)  {  for (size\_t j = 0; j < Node.get\_size(); j++)  {  if (dist[i] > (dist[j] + matrix[j][i]) && i!=j)  {  dist[i] = dist[j] + matrix[j][i];  way[k] = (int)j;  control = true;  }  }  }  if (!control) break;  }  if (dist[second] == inf)  cout << "Пути из " << Node.at(first) << " в " << Node.at(second) << " не существует" << endl;  else  {  cout << "Путь из " << Node.at(first) << " в " << Node.at(second) << " стоит " << dist[second] << endl;  cout << "Маршрут: " << Node.at(first);  for (size\_t i = 0; i < Node.get\_size(); i++)  {  if (way[i] == -1 || way[i] == second)  break;  cout << " -> " << Node.at(way[i]);  }  cout << "->" << Node.at(second) << endl;  }  size\_t result = dist[second];  delete[] dist; delete[] way;  return result;  }  void Graph::clear()  {  Node.clear();  Info.clear();  for (size\_t i = 0; i < size; i++)  {  delete[] matrix[i];  }  delete[] matrix;  matrix = NULL;  size = 0;  }  void Graph::summarize()  {  size\_t iter = Info.get\_size();  int\*\* tmp = new int\* [iter];  for (size\_t i = 0; i < iter; i++)  tmp[i] = new int[4];  for (size\_t i = 0; i < iter; i++)  {  string temp = Info.at(i), name = "";  string way[4] = { "","","","" };  size\_t k = 0;  for (size\_t j = 0; j < temp.size(); j++)  {  if (temp[j] == ';') k++;  else way[k].push\_back(temp[j]);  }  for (size\_t k = 0; k < 2; k++)  {  if(Node.find(way[k])==-1) Node.push\_back(way[k]);  //cout << way[k + 2] << endl ;  if (way[k + 2][0] == 'N')  tmp[i][k + 2] = inf;  else tmp[i][k + 2] = stoi(way[k + 2]);  }  tmp[i][0] = Node.find(way[0]);  tmp[i][1] = Node.find(way[1]);  }  size = Node.get\_size();  matrix = new int\* [size];  for (size\_t i = 0; i < size; i++)  {  matrix[i] = new int[size];  for (size\_t j = 0; j < Node.get\_size(); j++)  matrix[i][j] = inf;  }  for (size\_t i = 0; i < iter; i++)  {  matrix[tmp[i][0]][tmp[i][1]] = tmp[i][2];  matrix[tmp[i][1]][tmp[i][0]] = tmp[i][3];  delete[] tmp[i];  }  delete[] tmp;  } |

## Примеры работы программы

|  |  |  |
| --- | --- | --- |
| Входные данные: | Граф: | Результат: |
| 1;2;2;2  1;3;3;N/A  2;4;1;N/A  3;4;5;N/A  1;4;7;N/A |  | Путь из 1 в 2 стоит 2  Маршрут: 1->2  Путь из 1 в 3 стоит 3  Маршрут: 1 -> 2->3  Путь из 1 в 4 стоит 3  Маршрут: 1 -> 2->4  Путь из 2 в 1 стоит 2  Маршрут: 2->1  Путь из 2 в 3 стоит 5  Маршрут: 2 -> 1->3  Путь из 2 в 4 стоит 1  Маршрут: 2 -> 1->4  Пути из 3 в 1 не существует  Пути из 3 в 2 не существует  Путь из 3 в 4 стоит 5  Маршрут: 3->4  Пути из 4 в 1 не существует  Пути из 4 в 2 не существует  Пути из 4 в 3 не существует |
| Санкт-Петербург; Москва;10;20  Москва; Хабаровск;40;35  Санкт-Петербург; Хабаровск;14;N/A  Владивосток; Хабаровск;13;8  Владивосток; Санкт-Петербург; N/A;20 |  | Путь из Санкт-Петербург в Москва стоит 10  Маршрут: Санкт-Петербург->Москва  Путь из Санкт-Петербург в Хабаровск стоит 14  Маршрут: Санкт-Петербург->Хабаровск  Путь из Санкт-Петербург в Владивосток стоит 20  Маршрут: Санкт-Петербург->Владивосток  Путь из Москва в Санкт-Петербург стоит 20  Маршрут: Москва->Санкт-Петербург  Путь из Москва в Хабаровск стоит 34  Маршрут: Москва -> Санкт-Петербург->Хабаровск  Путь из Москва в Владивосток стоит 40  Маршрут: Москва -> Санкт-Петербург->Владивосток  Путь из Хабаровск в Санкт-Петербург стоит 55  Маршрут: Хабаровск -> Москва->Санкт-Петербург  Путь из Хабаровск в Москва стоит 35  Маршрут: Хабаровск->Москва  Путь из Хабаровск в Владивосток стоит 8  Маршрут: Хабаровск -> Москва->Владивосток  Путь из Владивосток в Санкт-Петербург стоит 68  Маршрут: Владивосток -> Хабаровск -> Москва->Санкт-Петербург  Путь из Владивосток в Москва стоит 48  Маршрут: Владивосток -> Хабаровск->Москва  Путь из Владивосток в Хабаровск стоит 13  Маршрут: Владивосток->Хабаровск |

## Вывод

В ходе выполнения работы я реализовал алгоритм Беллмана-Форда для поиска кратчайшего пути в графе, на основе матрицы смежности.