**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по лабораторной работе №3**

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

**Тема: «Алгоритмы на графах»**

**Вариант 2**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_ Ковтун А.С.

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

Санкт-Петербург

2021

* **Постановка задачи и описание реализуемого класса и методов**

Дан список возможных авиарейсов в текстовом файле в формате:

Город отправления 1;Город прибытия 1;цена прямого перелета 1;цена обратного перелета 1

Город отправления 2;Город прибытия 2;цена перелета 2;цена обратного перелета 1

…

Город отправления N;Город прибытия N;цена перелета N;цена обратного перелета N

В случае, если нет прямого или обратного рейса, его цена будет указана как N/A (not available).

**Пример данных:**

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

**Задание**: найти наиболее эффективный по стоимости перелет из города i в город j

|  |  |
| --- | --- |
| **Описание** | **Оценка временной сложности** |
| Поиск в очереди | O(n) |
| Получение матрицы | O(1) |
| Создание матрицы | O(n^2) |
| Получение индексов | O(n) |
| Алгоритм Форда-Беллмана | O(n^3) |

* **Описание реализованных unit-тестов**

Реализованные Unit-тесты проверяют работу конструктора и методов; работа с файлами; проверка работы программы на полных и неполных графах.

* **Код программы**

|  |
| --- |
| Queue.h  #pragma once  #include "Elem.h"  template <class K, class V>  class Queue  {  private:  Elem<K, V>\* head; // old elem  Elem<K, V>\* tail; // new elem  void set\_head(Elem<K, V>\* key) { head = key; }  void set\_tail(Elem<K, V>\* key) { tail = key; }  public:  Queue()  {  head = nullptr;  tail = nullptr;  }  bool is\_empty()  {  if (tail == nullptr)  return true;  return false;  }  size\_t get\_size() // return a number of elements in queue  {  if (is\_empty())  return 0;  int size = 1;  for (Elem<K, V>\* now = tail; now->get\_next() != nullptr; now = now->get\_next())  size++;  return size;  }  Elem<K, V>\* top() // to know who will leave first  {  return head;  }  Elem<K, V>\* push(V data) // adds to the end  {  Elem<K, V>\* new\_elem = new Elem<K, V>;  new\_elem->set\_data(data);  new\_elem->set\_key(get\_size()); // key = index in array  if (is\_empty()) {  head = new\_elem;  tail = new\_elem;  }  else  {  new\_elem->set\_next(tail);  tail = new\_elem;  }  return new\_elem;  }  Elem<K, V>\* pop() {  Elem<K, V>\* to\_delete;  if (is\_empty())  throw out\_of\_range("the queue is empty");  else if (get\_size() == 1)  {  to\_delete = head;  head = tail = nullptr;  }  else  {  to\_delete = head;  for (Elem<K, V>\* now = tail; now->get\_next() != nullptr; now = now->get\_next())  head = now;  head->next = nullptr;  }  return to\_delete;  }  K search\_key(V value) {  Elem<K, V>\* current = tail;  while (current != nullptr)  {  if (current->get\_data() == value)  return current->get\_key();  current = current->get\_next();  }  return -1;  }  ~Queue() { while (!is\_empty()) pop(); }  }; |
| FB.cpp |
| #include "FB.h"  void FB::set\_size(size\_t size) { size = size; }  void FB::set\_cities(string\* cities) { towns = cities; }  void FB::set\_weights(double\*\* weights) { weights = weights; }  FB::FB()  {  size = 0;  towns = nullptr;  weights = nullptr;  }  FB::FB(string filename)  {  set\_size(0);  fstream in(filename);  if (!in.is\_open())  throw invalid\_argument("file doesn't exist");  Queue<size\_t, string> cities;  string city;  while (in.peek() != EOF)  {  city = "";  while (in.peek() != ';')  city += in.get();  if (cities.search\_key(city) == -1)  cities.push(city);  in.get();  city = "";  while (in.peek() != ';')  city += in.get();  if (cities.search\_key(city) == -1)  cities.push(city);  while ((in.peek() != '\n') && (in.peek() != EOF))  in.get();  if (in.peek() == '\n')  in.get();  }  set\_size(cities.get\_size());  towns = new string[get\_size()];  for (size\_t i = 0; i < get\_size(); i++)  towns[i] = cities.pop()->get\_data();  weights = new double\* [get\_size()];  for (size\_t i = 0; i < get\_size(); i++)  weights[i] = new double[get\_size()];  for (size\_t i = 0; i < get\_size(); i++)  {  for (size\_t j = 0; j < get\_size(); j++)  {  if (i == j)  weights[i][j] = 0;  else  weights[i][j] = INF;  }  }  in.close();  in.open(filename);  while (in.peek() != EOF)  {  size\_t index\_from, index\_to;  double forward\_cost, backward\_cost;  string info = "";  while (in.peek() != ';')  info += in.get();  index\_from = get\_city\_index(info);  in.get();  info = "";  while (in.peek() != ';')  info += in.get();  index\_to = get\_city\_index(info);  in.get(); // read ';'  if (in.peek() == 'N')  {  forward\_cost = INF; // unreal cost  while (in.get() != ';') {}  }  else  {  in >> forward\_cost;  in.get(); // read ';'  }  if (in.peek() == 'N')  {  backward\_cost = INF; // unreal cost  for (int i = 0; i < 3; i++)  in.get(); // read "N/A"  }  else  in >> backward\_cost;  if (in.peek() == '\n') // read '\n'  in.get();  if (forward\_cost < INF)  weights[index\_from][index\_to] = forward\_cost;  if (backward\_cost < INF)  weights[index\_to][index\_from] = backward\_cost;  // filled weights matrix  }  }  size\_t FB::get\_size() { return size; }  string\* FB::get\_cities() { return towns; }  double\*\* FB::get\_weights() { return weights; }  size\_t FB::get\_city\_index(string city)  {  for (size\_t i = 0; i < get\_size(); i++)  {  if (towns[i] == city)  return i;  }  return -1; // in case of not-existing city return unreal index  }  double FB::FordBellman(string from, string to)  {  ofstream out("out.txt");  size\_t f = get\_city\_index(from);  size\_t t = get\_city\_index(to);  if (f == -1)  {  out << "the source city doesn't exist" << endl;  throw invalid\_argument("the source city doesn't exist");  }  if (t == -1)  {  out << "the target city doesn't exist" << endl;  throw invalid\_argument("the target city doesn't exist");  }  double min\_cost = INF;  size\_t\* marks = new size\_t[get\_size()]{ f };  double\*\* W = new double\* [get\_size()];  for (size\_t i = 0; i < get\_size(); i++)  W[i] = new double[get\_size()];  for (size\_t i = 0; i < get\_size(); i++)  {  for (size\_t j = 0; j < get\_size(); j++)  W[i][j] = get\_weights()[i][j];  }  size\_t p;  for (p = 0; p < get\_size(); p++)  {  bool is\_improoved = false;  for (size\_t node = 0; node < get\_size(); node++)  {  for (size\_t to = 0; to < get\_size(); to++)  {  if (W[f][to] > W[f][node] + W[node][to])  {  W[f][to] = W[f][node] + W[node][to];  marks[to] = node;  is\_improoved = true;  }  }  }  if (!is\_improoved) // if on current phase wasn't any improvements  break;  }  if (p == get\_size()) // unreal, so exists a negative cycle  {  out << "There is a negative cycle, program stops working." << endl;  throw invalid\_argument("Found negative cycle");  }  min\_cost = W[f][t];  if (min\_cost >= INF)  {  out << to << " can't be reached from " << from << endl;  throw invalid\_argument("path doesn't exist");  }  out << "minimum cost of reaching " << to << " from " << from << " is " << W[f][t] << endl;  // going to restore the way and print it to the file  size\_t\* reversed\_way = new size\_t[get\_size()]{ f }; // an array saving indexes of cities in reversed order  size\_t city = t;  int i = 0;  while (city != f)  {  reversed\_way[i] = city;  city = marks[city];  i++;  }  reversed\_way[i] = f;  out << "the way:" << endl;  for (int k = i; k >= 0; k--)  {  out << get\_cities()[reversed\_way[k]];  if (k != 0)  out << " -> ";  }  out.close();  return W[f][t];  } |
| Elem.h |
| #pragma once  #include <iostream>  using namespace std;  template <class K, class V>  class Elem  {  Elem<K, V>\* next;  V data;  K key;  void set\_next(Elem<K, V>\* new\_element) { next = new\_element; }  void set\_data(V new\_data) { data = new\_data; }  void set\_key(K new\_key) { key = new\_key; }  public:  Elem() { }  Elem<K, V>\* get\_next() { return next; }  V get\_data() { return data; }  K get\_key() { return key; }  ~Elem() {}  template <class K, class V> friend class Queue;  }; |
| FB.h  #pragma once  #include"Queue.h"  #include <fstream>  #include <stdexcept>  #include <float.h>  #define INF (DBL\_MAX) // to declare infinity value for not-existing nodes  using namespace std;  class FB  {  size\_t size;  string\* towns;  double\*\* weights;  void set\_size(size\_t);  void set\_cities(string\*);  void set\_weights(double\*\*);  FB();  public:  FB(string); // input file name  size\_t get\_size();  string\* get\_cities();  double\*\* get\_weights();  size\_t get\_city\_index(string);  double FordBellman(string, string); // originating and arriving towns  }; |

* **Пример работы**

В первом файле находится информация о существующих и несуществующих перелётах и их стоимости, во втором файле находится информация о наименьшей цене перелёта.

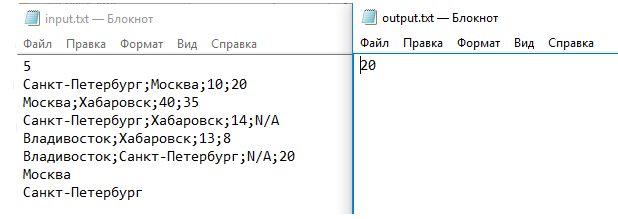


Рисунок 1. Файлы с данными работы программы.

Листинг 1 — файл исходных данных:

5

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

Москва

Санкт-Петербург

Листинг 2 — файл с результатами работы программы:

20