**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 |  | Сиротин П.О. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

Оглавление

[1. Постановка задачи. Краткое описание реализуемого алгоритма и используемых структур данных. Оценка временной сложности каждого метода 3](#_Toc74074392)

[2. Описание реализованных unit-тестов 4](#_Toc74074393)

[3. Пример работы. 4](#_Toc74074394)

[4. Листинг. 5](#_Toc74074395)

[5. Вывод 12](#_Toc74074396)

1. **Постановка задачи. Краткое описание реализуемого алгоритма и используемых структур данных. Оценка временной сложности каждого метода**

* 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***.
* Необходимо решить данную задачу через алгоритм Беллмана-Форда.

Классы lst\_node и ListL2 были взяты из предыдущей лабораторной работы, но были внесены некоторые изменения в их реализацию:

1. Был добавлен метод FindAnElement(), который возвращает или индекс элемента списка, если элемент был найден в списке, или -1, если элемент не был найден.
2. Были добавлены методы push\_back() и pop\_back(): по сути, это просто аналоги push() и pop() из прошлой работы, просто переписанные для работы с «хвостом» списка.
3. Класс stack превратился в двунаправленный список, поскольку так будет удобнее работать с множеством имён вершин.

template <class T>

Class lst\_node

|  |  |
| --- | --- |
| Элемент | Описание |
| T info | Информация внутри узла стека |
| lst\_node\* prev | Указатель на предыдущий элемент списка |
| lst\_node\* next | Указатель на следующий элемент списка |

template <class T>

Class ListL2

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент | Описание | | |
| lst\_node\* top | Указатель на первый элемент списка | | |
| lst\_node\* tail | Указатель на последний элемент списка | | |
| Метод | | Описание | Оценка временной сложности |
| void push\_front(T data) | | Вставка нового элемента в начало списка | О(1) |
| void push\_back(T data) | | Вставка нового элемента в конец списка | О(1) |
| T pop\_front() | | Удаление элемента из начала списка | О(1) |
| T pop\_back() | | Удаление элемента из конца списка | О(1) |
| int FindAnElement(T dataToFind) | | Поиск элемента в списке – если элемент найден, возвращается его индекс, иначе же возвращается -1 | О(n) |
| size\_t give\_size() | | Возвращает количество элементов списка | O(n) |
| void print() | | Вывод информации из элементов стека в консоль | О(n) |
| T what\_at(int index) | | Возвращает данные, находящиеся в элементе, имеющим данный индекс в списке | O(n) |

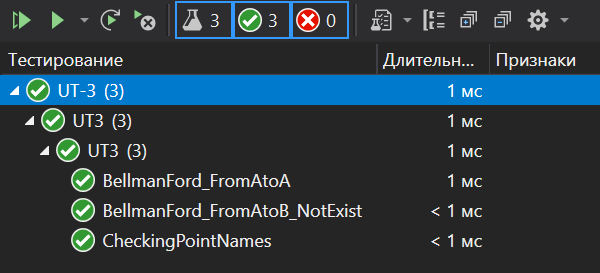
Реализованные функции

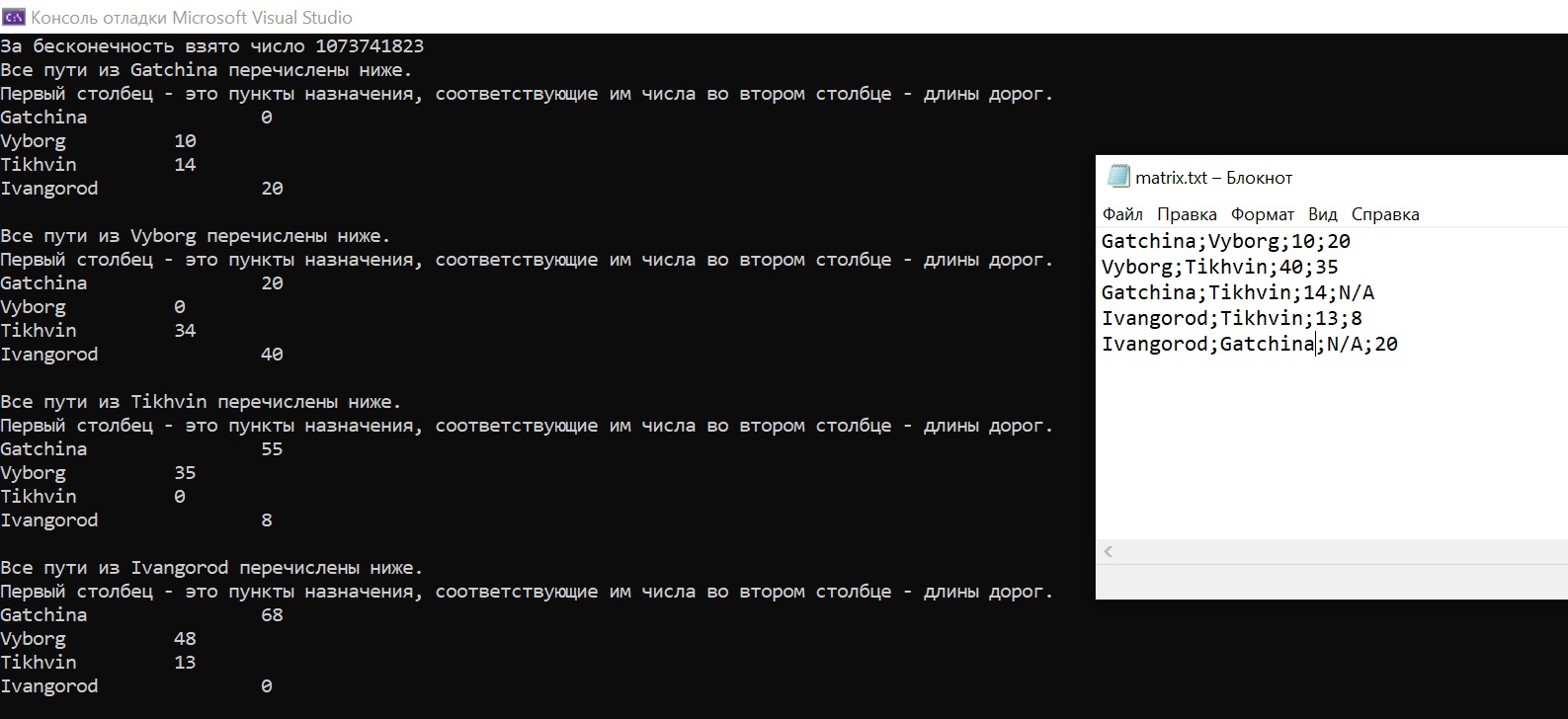
|  |  |  |
| --- | --- | --- |
| Функция | Описание | Оценка временной сложности |
| int\*\* ReadFromFile(ListL2<string>\* names, ifstream &file, unsigned& edges\_num) | Чтение данных из файла, составление матрицы для дальнейшего решения задачи и реализации алгоритма Беллмана-Форда. Кроме рёбер, сохраняются и названия пунктов, из которых составляется список с уникальными названиями | О(n) |
| void BellmanFord(ListL2<string>\* names, int\*\* greatMatrix, int source, int Edges, int\*& dis) | Алгоритм Беллмана-Форда. В конце алгоритма на экран выводятся все пути из начальной точки. | О(n3) |
| void Summing() | Процесс работы программы. На экран выводятся данные о путях из каждой точки (для удобного изучения информации) | О(n3) |

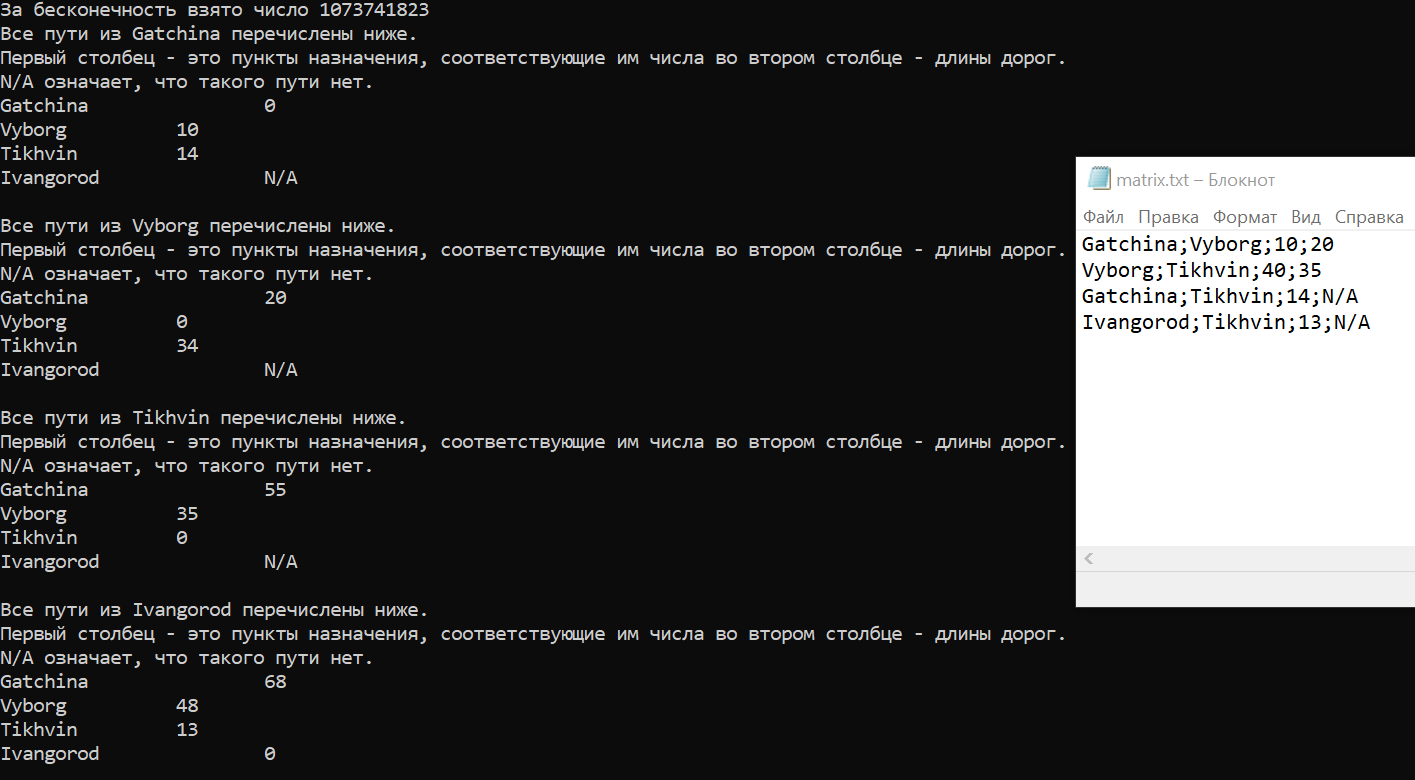
# Описание реализованных unit-тестов

|  |  |
| --- | --- |
| CheckingPointNames | Проверка функции ReadFromFile, правильно ли заносятся названия городов в список |
| BellmanFord\_FromAtoA | Проверка функции BellmanFord, равен ли путь из города А в город А нулю |
| BellmanFord\_FromAtoB\_NotExist | Проверка функции BellmanFord на несуществующий путь |

# Пример работы.







# Листинг.

|  |
| --- |
| ListL2.h |
| #pragma once  #include <stdexcept>  #include <iostream>  using namespace std;  template <class T>  class ListL2  {  private:  class lst\_node  {  public:  lst\_node(T info, lst\_node\* next = nullptr, lst\_node\* prev = nullptr)  {  this->info = info;  this->next = next;  this->prev = prev;  }  lst\_node()  {  this->prev = nullptr;  this->next = nullptr;  }  ~lst\_node() {};  T info;  lst\_node\* prev;  lst\_node\* next;  };  public:  lst\_node\* top;  lst\_node\* tail;  ListL2()  {  top = nullptr;  tail = nullptr;  }  void push\_back(T data)  {  if (top == nullptr)  {  top = tail = new lst\_node(data);  }  else  {  lst\_node\* temp = new lst\_node(data, nullptr, tail);  tail->next = temp;  tail = temp;  }  }  void push\_front(T data)  {  if (top)  {  lst\_node\* lasthead = top;  top = new lst\_node(data, top);  lasthead->prev = top;  }  else  {  top = new lst\_node(data, top);  }  }  T pop\_front()  {  if (!top)  throw out\_of\_range("An error occured.");  if (top->next == nullptr)  {  T tmp = top->info;  tail = nullptr;  top = nullptr;  delete top;  delete tail;  return tmp;  }  else  {  lst\_node\* node\_temp = top;  T tmp = node\_temp->info;  top = top->next;  top->prev = nullptr;  node\_temp->next = nullptr;  delete node\_temp;  return tmp;  }  }  T pop\_back()  {  if (!tail)  throw out\_of\_range("An error occured.");  if (tail->prev == nullptr)  {  T tmp = tail->info;  tail = nullptr;  top = nullptr;  delete top;  delete tail;  return tmp;  }  else  {  lst\_node\* node\_temp = tail;  T tmp = node\_temp->info;  tail = tail->prev;  tail->next = nullptr;  node\_temp->prev = nullptr;  delete node\_temp;  return tmp;  }  }  void print()  {  lst\_node\* current = top;  if (top)  {  while (current->next)  {  cout << current->info << " ";  current = current->next;  }  cout << current->info;  cout << endl;  }  else throw out\_of\_range("An error occured: list is empty.");  }  T what\_at(int index)  {  if (top)  {  lst\_node\* current = top;  int i = 0;  while (i != index)  {  if (current->next)  current = current->next;  else  if (i == index)  return current->info;  i++;  }  return current->info;  }  else throw "An error occured.";  }  int FindAnElement(T dataToFind)  {  int i = 0;  if (top != nullptr)  {  lst\_node\* current = top;  while(current)  {  if (dataToFind == current->info)  {  return i;  }  current = current->next;  i++;  }  }  return -1;  }  size\_t give\_size()  {  size\_t i = 0;  lst\_node\* cur = top;  while (cur)  {  cur = cur->next;  i++;  }  return i;  }  ~ListL2()  {  T tmp;  while (top)  {  tmp = pop\_front();  }  }  }; |
| BellmanFordAlgo.h |
| #pragma once  #include <fstream>  #include <iostream>  #include <stdexcept>  #include <string>  #include "ListL2.h"  using namespace std;  int\*\* ReadFromFile(ListL2<string>\* names, ifstream &file, unsigned& edges\_num) // e.g. we have lines like "Gatchina;SaintPetersburg;20;10"  {  // in "names" we will store our names  string temp, usersText;  int count, i = 0, j = 0, edges\_tmp = 0;  while (!file.eof())  {  getline(file, usersText);  count = 0; // in every line we have only 4 positions - town1, town2, way1, way2  for (size\_t k = 0; k < usersText.size(); k++)  {  temp = "";  while ((usersText[k] != ';') && (usersText[k] != ',') && (k != usersText.size())) // user can divide it by , or ;  temp += usersText[k++]; // building a string by 1 symbol at iteration  switch (count)  {  case 0:  {  if (names->FindAnElement(temp) == -1)  names->push\_back(temp);  break;  }  case 1:  {  if (names->FindAnElement(temp) == -1)  {  names->push\_back(temp);  //names->print();  }  break;  }  case 2:  {  if (temp != "N/A")  edges\_tmp++;  break;  }  case 3:  {  if (temp != "N/A")  edges\_tmp++;  break;  }  }  count++;  }  }  file.close();  file.open("matrix.txt");  // every {u, v, w} in greatMatrix is just an edge of graph  // it can be read like "from u to v way costs w"  int\*\* greatMatrix = new int\* [edges\_tmp];  for (int i = 0; i < edges\_tmp; i++)  greatMatrix[i] = new int[3];  //for (int i = 0; i < edges\_tmp; i++)  // for (int j = 0; j < 3; j++)  // greatMatrix[i][j] = 0; // 0 in matrix means N/A    while (!file.eof())  {  getline(file, usersText);  count = 0; // in every line we have only 4 positions - town1, town2, way1, way2  for (size\_t k = 0; k < usersText.size(); k++)  {  temp = "";  while ((usersText[k] != ';') && (usersText[k] != ',') && (k != usersText.size())) // user can divide it by , or ;  temp += usersText[k++]; // building a string by 1 symbol at iteration  switch (count)  {  case 0:  {  if (names->FindAnElement(temp) != -1)  i = names->FindAnElement(temp);  break;  }  case 1:  {  if (names->FindAnElement(temp) != -1)  j = names->FindAnElement(temp);  break;  }  case 2: // greatMatrix is an adjacency matrix; greatMatrix[i][j] == greatMatrix[j][i]  {  if (temp != "N/A") // 0 in matrix means N/A  {  greatMatrix[edges\_num][0] = i;  greatMatrix[edges\_num][1] = j;  greatMatrix[edges\_num][2] = atoi(temp.c\_str()); // str.c\_str() to make string a const char\* value  edges\_num++;  }  break;  }  case 3: // greatMatrix is an adjacency matrix; greatMatrix[i][j] == greatMatrix[j][i]  {  if (temp != "N/A") // 0 in matrix means N/A  {  greatMatrix[edges\_num][0] = j;  greatMatrix[edges\_num][1] = i;  greatMatrix[edges\_num][2] = atoi(temp.c\_str()); // str.c\_str() to make string a const char\* value  edges\_num++;  }  break;  } // a little of explaining: we have line "Moscow;Khabarovsk;10;20"  // case 2 and case 3 say that we are building an array of edges  // first edge is {0, 1, 10} and it means that from Moscow to Khabarovsk the way costs 10  // second edge is {1, 0, 20} and it means that from Moscow to Khabarovsk the way costs 20  }  count++;  }  }  file.close();  return greatMatrix;  }  void BellmanFord(ListL2<string>\* names, int\*\* greatMatrix, int source, int Edges, int\*& distances)  {  for (int i = 0; i < names->give\_size(); i++)  distances[i] = INT\_MAX / 2; // INT\_MAX/2 == infinity  distances[source] = 0; // our start  for (int i = 0; i < names->give\_size() - 1; i++)  {  for (int j = 0; j < Edges; j++)  {  if ((distances[greatMatrix[j][0]] != INT\_MAX) && (distances[greatMatrix[j][0]] + greatMatrix[j][2] < distances[greatMatrix[j][1]]))  distances[greatMatrix[j][1]] = distances[greatMatrix[j][0]] + greatMatrix[j][2];  }  }  // now we are checking if there's some negative weight cycle  for (int i = 0; i < Edges; i++)  {  int x = greatMatrix[i][0];  int y = greatMatrix[i][1];  int weight = greatMatrix[i][2];  if ((distances[x] != INT\_MAX/2) && (distances[x] + weight < distances[y]))  cout << "Граф содержит отрицательный цикл." << endl;  }  cout << "Все пути из " << names->what\_at(source) << " перечислены ниже." << endl;  cout << "Первый столбец - это пункты назначения, соответствующие им числа во втором столбце - длины дорог." << endl;  cout << "N/A означает, что такого пути нет." << endl;  for (int i = 0; i < names->give\_size(); i++)  if (distances[i] != INT\_MAX / 2)  cout << names->what\_at(i) << "\t\t" << distances[i] << endl;  else cout << names->what\_at(i) << "\t\t" << "N/A" << endl;  cout << endl;  }  void Summing()  {  // source is the index of city from which  int\*\* matrixEdges;  unsigned edges = 0;  ListL2<string>\* names = new ListL2<string>();  ifstream file;  file.open("matrix.txt");  if (!file)  {  throw "No such file in directory.";  }  else matrixEdges = ReadFromFile(names, file, edges);  file.close();  int\* dis = new int[names->give\_size()];  for (int i = 0; i < names->give\_size(); i++)  BellmanFord(names, matrixEdges, i, edges, dis);  for (unsigned i = 0; i < edges; i++)  delete[] matrixEdges[i];  delete[] matrixEdges;  } |
| UT-3.cpp |
| #include "CppUnitTest.h"  #include <stdexcept>  #include "../Term4\_LabWork3/ListL2.h"  #include "../Term4\_LabWork3/BellmanFordAlgo.h"  using namespace std;  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UT3  {  TEST\_CLASS(UT3)  {  public:  TEST\_METHOD(CheckingPointNames)  {  int\*\* matrixEdges;  unsigned edges = 0;  ListL2<string>\* names = new ListL2<string>();  ifstream file;  file.open("C:\\Users\\Paul\\source\\repos\\UT-3\\matrix.txt");  matrixEdges = ReadFromFile(names, file, edges);  file.close();  Assert::IsTrue("Gatchina" == names->what\_at(0));  Assert::IsTrue("Vyborg" == names->what\_at(1));  Assert::IsTrue("Tikhvin" == names->what\_at(2));  Assert::IsTrue("Ivangorod" == names->what\_at(3));  }  TEST\_METHOD(BellmanFord\_FromAtoA)  {  int\*\* matrixEdges;  unsigned edges = 0;  ListL2<string>\* names = new ListL2<string>();  ifstream file;  file.open("C:\\Users\\Paul\\source\\repos\\UT-3\\matrix.txt");  matrixEdges = ReadFromFile(names, file, edges);  file.close();  int\* dis = new int[names->give\_size()];  BellmanFord(names, matrixEdges, 0, edges, dis);  Assert::IsTrue(dis[0] == 0);  for (unsigned i = 0; i < names->give\_size(); i++)  delete[] matrixEdges[i];  delete[] matrixEdges;  delete[] dis;  }  //Gatchina;Vyborg;10;20  //Vyborg;Tikhvin;40;35  //Gatchina;Tikhvin;14;N/A  //Ivangorod;Tikhvin;13;N/A  TEST\_METHOD(BellmanFord\_FromAtoB\_NotExist)  {  int\*\* matrixEdges;  unsigned edges = 0;  ListL2<string>\* names = new ListL2<string>();  ifstream file;  file.open("C:\\Users\\Paul\\source\\repos\\UT-3\\matrix.txt");  matrixEdges = ReadFromFile(names, file, edges);  file.close();  int\* dis = new int[names->give\_size()];  BellmanFord(names, matrixEdges, 0, edges, dis);  Assert::IsTrue(dis[3] == INT\_MAX/2);  for (unsigned i = 0; i < names->give\_size(); i++)  delete[] matrixEdges[i];  delete[] matrixEdges;  delete[] dis;  }  };  } |

# Вывод

Я изучил и научился реализовывать на графах алгоритм Беллмана-Форда и работать с ним.