**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**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9301 |  | Синицкая В. А. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2021

Оглавление

[1 Постановка задачи 3](#_Toc69944291)

[2 Реализуемые классы и методы 3](#_Toc69944292)

[3 Оценка временной сложности каждого метода 5](#_Toc69944293)

[4 Описание реализованных Unit test 5](#_Toc69944294)

[5 Пример работы программы 5](#_Toc69944295)

[5.1 TEST\_METHOD(Matrix\_3x3\_1) 5](#_Toc69944296)

[5.2 TEST\_METHOD(Matrix\_5x5\_1) 6](#_Toc69944297)

[5.3 TEST\_METHOD(No\_way) 7](#_Toc69944298)

[6 Листинг 8](#_Toc69944299)

[6.1 Element.h 8](#_Toc69944300)

[6.2 Queue.h 8](#_Toc69944301)

[6.3 WeightsMatrix.h 10](#_Toc69944302)

[6.4 WeightsMatrix.cpp 10](#_Toc69944303)

[6.5 UnitTestFordBellman.cpp 14](#_Toc69944304)

# Постановка задачи

20 вариант => 2 вариант:

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.

# Реализуемые классы и методы

В проекте содержатся классы Element, Queue, WeightsMatrix.

Класс Element содержит поле next (указатель на следующий элемент), поле key и поле data. Класс написан как шаблонный.

Класс Queue имеет поля head и tail, обозначающие первый и последний элемент в очереди. В данной реализации метод push отличается тем, что элемент добавляется по значению для поля data, а поле key заполняется автоматически так, что получается нумерация как в обычных массивах C++. Класс дополнен методом search\_key, принимающим значение поля data и возвращающим соответствующий индекс или значение -1 в случае, если такого элемента не существует, в том числе если очередь пуста. Класс написан как шаблонный.

Класс WeightsMatrix имеет три поля: Size, Citites, Weights. Size обозначает количество указанных в файле различных городов, Citites хранит названия этих городов, Weights является матрицей весов графа перелетов между городами. Weights[i][j] хранит цену перелета из города Cities[i] в город Citites[j], но программа расширена и в вариант, что цена может быть отрицательной. Если направленного ребра из одного в другой город не существует, в матрице весов хранится значение INF — максимально допустимое значение для типа double, которым обозначены веса. Для всех полей прописаны методы get и set. Также прописаны методы get\_city\_index и FordBellman, конструктор WeightsMatrix, принимающий название файла, содержащего входные данные. get\_city\_index принимает на вход строку — название города, и возвращает его индекс в Citites, либо значение -1 в случае, если такого города там нет.

Конструктор WeightsMatrix по умолчанию запрещен (имеет модификатор доступа private), конструктор по названию файла открывает файл с входными данными или генерирует исключение. Сначала происходит первый проход по файлу, в ходе которого считываются и записываются в очередь названия всех различных городов. Далее, зная размер, заполняется поле Size, затем по очереди заполняется поле Cities. Дальше происходит второй проход по файлу, и заполняется матрица весов.

Метод FordBellman принимает на вход две строки, где первая обозначает город отправления, а вторая — прибытия. Находятся индексы городов в Citites или генерируются исключения. Создается копия матрицы весов, чтобы не испортить настоящую информацию. Искомая минимальная дистанция обозначается как INF, создается массив пометок, который в i ячейке хранит индекс города в Cities, из котрого проложен путь в Cities[i] город. Объявляется фаза, равная нулю. По алгоритму гарантируется, что за n-1 фазу он успешно завершает работу при отсутствии в графе отрицательных циклов (n — количество вершин графа, то есть количество городов). Путь от истока (города отправления) улучшается за счет поиска более ломаных путей к городу назначения. При отсутствии улучшения на какой-либо фазе алгоритм может быть завершен, так как очевидно, что в таком случае дальше улучшений не произойдет тоже, поэтому цикл поиска улучшений прерывается, если их не было. Если при выходе из цикла оказалось, что фаза = n, это говорит о том, что был найден цикл отрицательной длины, так как n-1 фаза была пройдена, а улучшения все равно нашлись. При нахождении отрицательного цикла генерируется исключение. Вообще говоря, алгоритм не находит все отрицательные циклы, те, которые были недостижимы при поиске нужного пути, не будут найдены. Иначе метод возвращает значение минимальной цены перелета из города отправления в город прибытия. Попутно создается файл, в который записывается порядок перелетов, откуда, куда и в какой последовательности нужно совершать перелет, чтобы достичь данной минимальной цены.

# Оценка временной сложности каждого метода

1. Queue Key\_Type search\_key(Value\_Type current\_value) — O(N).
2. Все методы set и get WeightsMatrix — O(1).
3. WeightsMatrix(string)(конструктор) — O(N^2), где N — количество символов входного файла.
4. WeightsMatrix size\_t get\_city\_index(string) — O(N).
5. WeightsMatrix double FordBellman(string, string) — O(N^3), где N = Size — количество различных городов (вершин графа).

# Описание реализованных Unit test

Была проверена работа всех методов, были проверены генерируемые исключения при запросе решить задачу для несуществующего города, для отрицательных циклов на искомом пути, была проверена работа на графах 3 на 3 и 5 на 5, содержащих отрицательные веса или без них, с отсутствующими ребрами или без таких. Была проверена работа конструктора.

# Пример работы программы

Примером работы послужат результаты выполнения некоторых тестов:

## TEST\_METHOD(Matrix\_3x3\_1)

На Рис. 5.1.1 и Рис. 5.1.2 показаны результаты теста 1.1 TEST\_METHOD(Matrix\_3x3\_1) для матрицы 3 на 3 без отрицательных весов, со всеми ребрами.

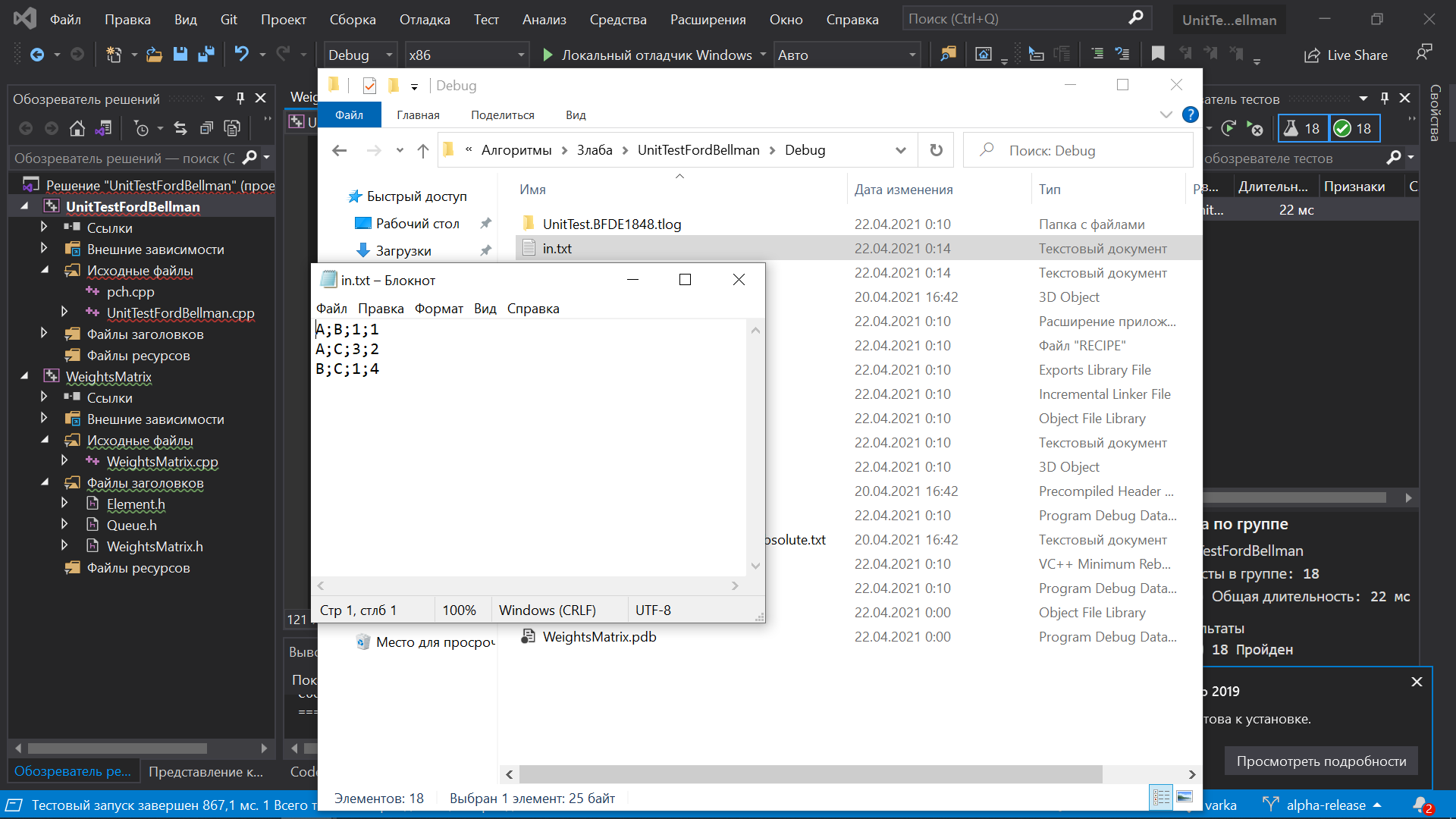


Рис. . — входные данные для примера 1.

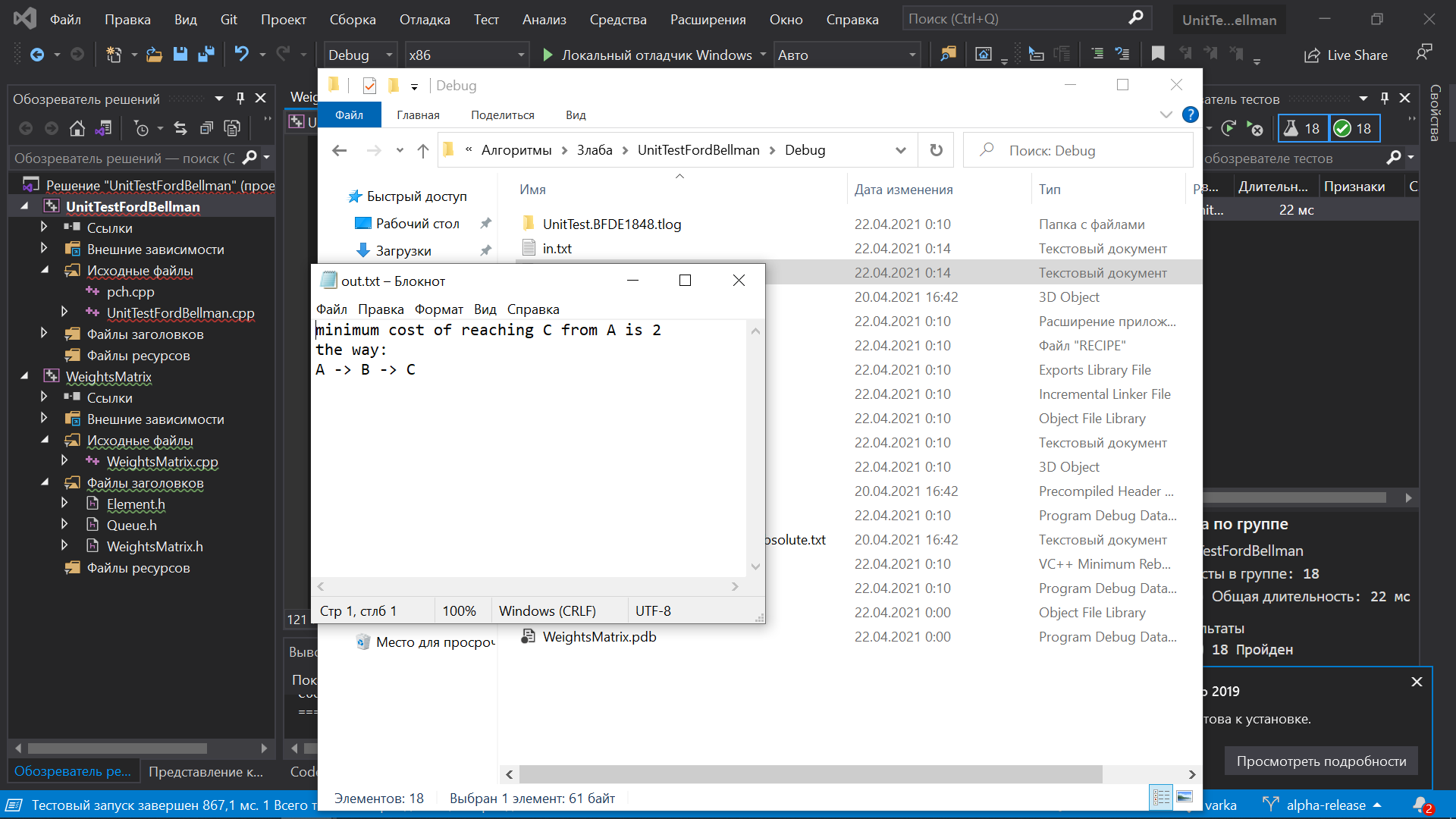


Рис. . — выходные данные для примера 1.

## TEST\_METHOD(Matrix\_5x5\_1)

На Рис. 5.2.1 и Рис. 5.2.2 показаны результаты работы теста TEST\_METHOD(Matrix\_5x5\_1) для матрицы 5 на 5, где есть отрицательные веса, существуют не все направленные ребра.

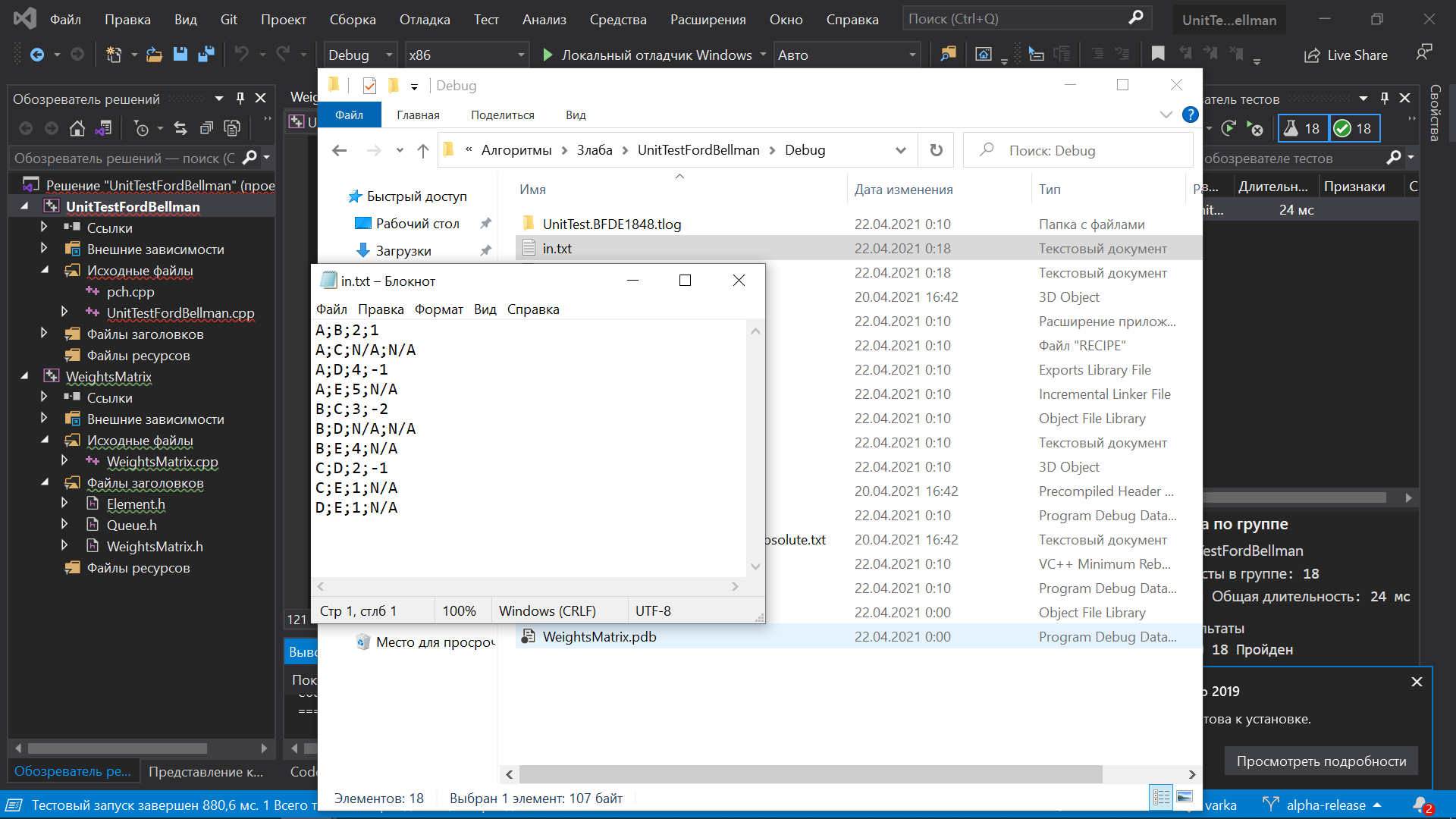


Рис. . — входные данные для примера 2.

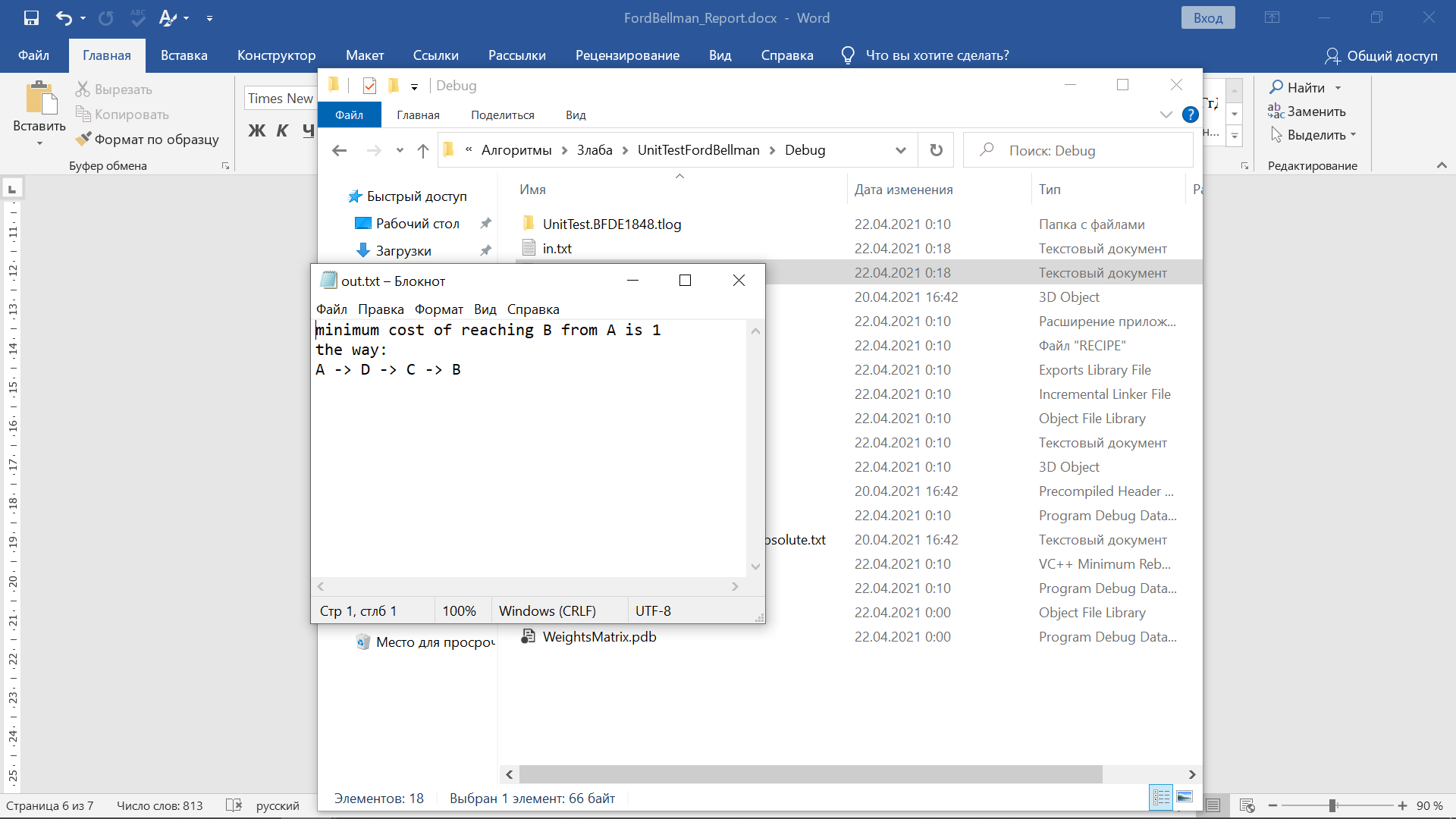


Рис. . — выходные данные для примера 2.

## TEST\_METHOD(No\_way)

На Рис. 5.3.1 и Рис. 5.3.2 показаны результаты работы теста 1.1 TEST\_METHOD(No\_way) для матрицы 3 на 3, где показано, что не существует искомого пути.

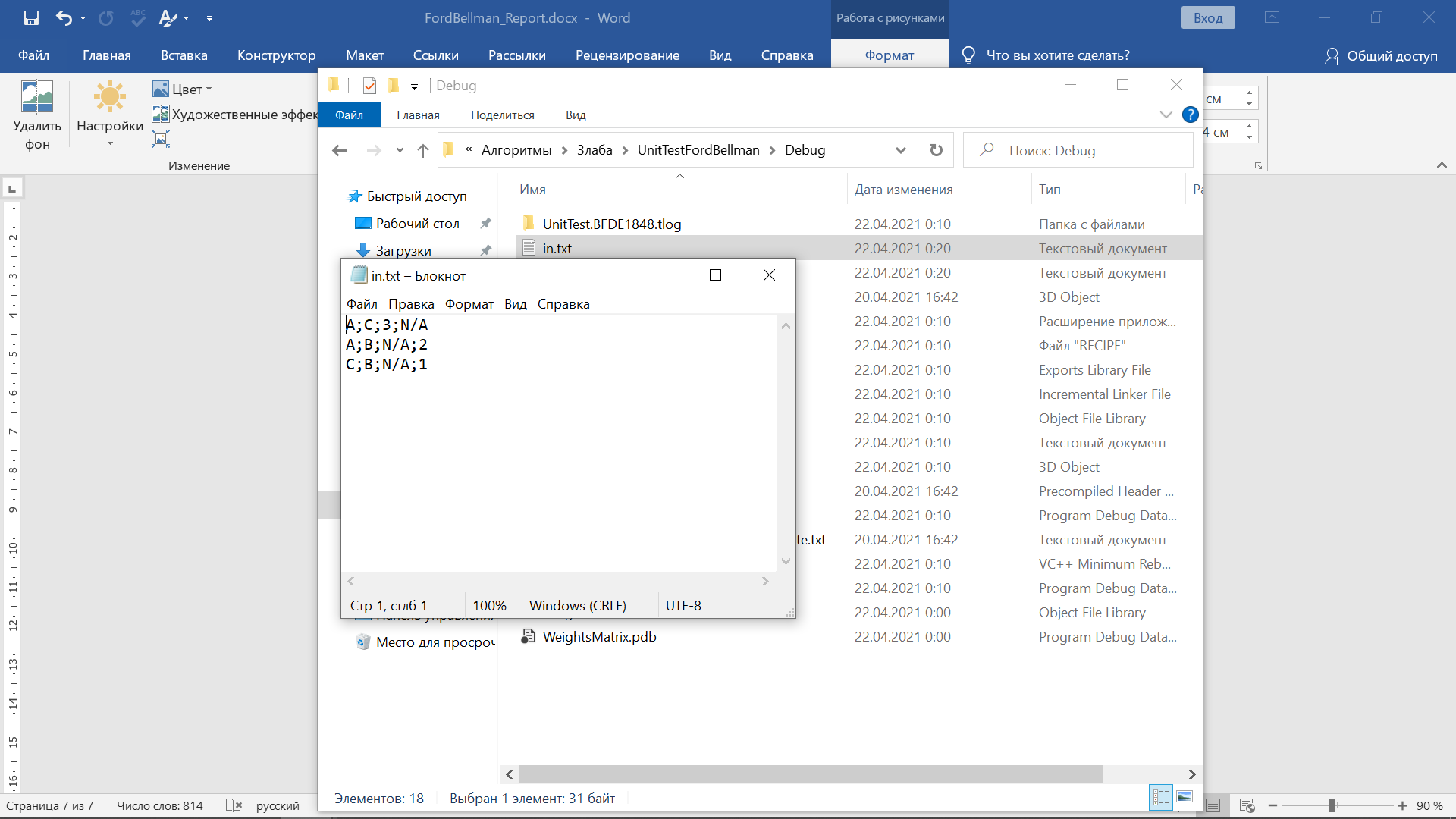


Рис. . — входные данные для примера 3.

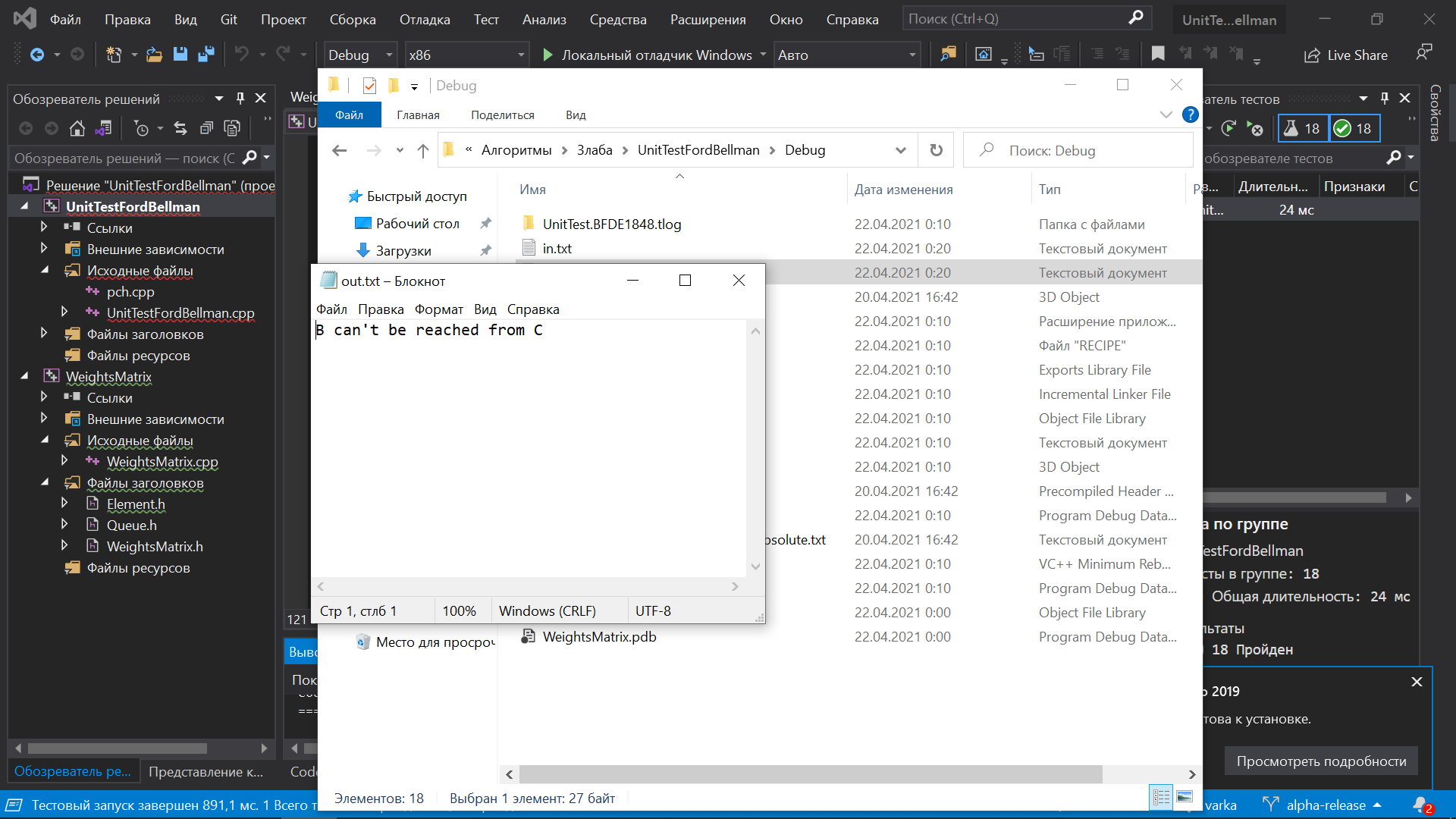


Рис. . — выходные данные для примера 3.

# Листинг

## Element.h

1. #pragma once
2. #include <iostream>
3. using namespace std;
4. template <class Key\_Type, class Value\_Type>
5. class Element
6. {
7. Element<Key\_Type, Value\_Type>\* next;
8. Value\_Type data;
9. Key\_Type key;
10. void set\_next(Element<Key\_Type, Value\_Type>\* new\_element) { next = new\_element; }
11. void set\_data(Value\_Type new\_data) { data = new\_data; }
12. void set\_key(Key\_Type new\_key) { key = new\_key; }
14. public:
15. Element() { }
16. Element<Key\_Type, Value\_Type>\* get\_next() { return next; }
17. Value\_Type get\_data() { return data; }
18. Key\_Type get\_key() { return key; }
19. ~Element() {}
20. template <class Key\_Type, class Value\_Type> friend class Queue;
21. };

## Queue.h

1. #pragma once
2. #include "Element.h"
3. template <class Key\_Type, class Value\_Type>
4. class Queue
5. {
6. private:
7. Element<Key\_Type, Value\_Type>\* head; // the first in queue, leaves first
8. Element<Key\_Type, Value\_Type>\* tail; // the last, new in queue
9. void set\_head(Element<Key\_Type, Value\_Type>\* key) { head = key; }
10. void set\_tail(Element<Key\_Type, Value\_Type>\* key) { tail = key; }
11. public:
12. Queue()
13. {
14. head = nullptr;
15. tail = nullptr;
16. }
17. bool is\_empty() // returns true if queue is empty
18. {
19. if (tail == nullptr)
20. return true;
21. return false;
22. }
23. int size() // return a number of elements in queue
24. {
25. if (is\_empty())
26. return 0;
27. int size = 1;
28. for (Element<Key\_Type, Value\_Type>\* now = tail; now->get\_next() != nullptr; now = now->get\_next())
29. size++;
30. return size;
31. }
32. Element<Key\_Type, Value\_Type>\* top() // to know who will leave first
33. { return head; }
35. Element<Key\_Type, Value\_Type>\* push(Value\_Type data) // to add in the end of the queue the element with key-data and return it to user
36. {
37. Element<Key\_Type, Value\_Type>\* new\_element = new Element<Key\_Type, Value\_Type>;
38. new\_element->set\_data(data);
39. new\_element->set\_key(size()); // key = index in array
40. if (is\_empty())
41. head = tail = new\_element;
42. else
43. {
44. new\_element->set\_next(tail);
45. tail = new\_element;
46. }
47. return new\_element;
48. }
50. Element<Key\_Type, Value\_Type>\* pop() // to delete first in queue and return it to user
51. {
52. Element<Key\_Type, Value\_Type>\* to\_delete;
53. if (is\_empty())
54. throw out\_of\_range("the queue is empty");
55. else if (size() == 1)
56. {
57. to\_delete = head;
58. head = tail = nullptr;
59. }
60. else
61. {
62. to\_delete = head;
63. for (Element<Key\_Type, Value\_Type>\* now = tail; now->get\_next() != nullptr; now = now->get\_next())
64. head = now;
65. head->next = nullptr;
66. }
67. return to\_delete;
68. }
70. Key\_Type search\_key(Value\_Type current\_value)
71. {
72. Element<Key\_Type, Value\_Type>\* current = tail;
73. while (current != nullptr)
74. {
75. if (current->get\_data() == current\_value)
76. return current->get\_key();
77. current = current->get\_next();
78. }
79. return -1; // return unreal index because not found or empty queue and so not found too
80. }
82. ~Queue()
83. {
84. while (!is\_empty())
85. pop();
86. }
87. };

## WeightsMatrix.h

1. #pragma once
2. #include "Queue.h"
3. #include <fstream>
4. #include <stdexcept>
5. #include <float.h>
6. #define INF (DBL\_MAX) // to declare infinity value for not-existing nodes
7. using namespace std;
8. class WeightsMatrix
9. {
10. size\_t Size; // amount of cities, size^2 = size of weights matrix
11. string\* Cities; // an array of declared cities // indexes of cities are like their keys
12. double\*\* Weights; // a matrix of weights // indexes means cities, i - from what city, j - to what city
13. void set\_size(size\_t);
14. void set\_cities(string\*);
15. void set\_weights(double\*\*);
16. WeightsMatrix();
17. public:
18. WeightsMatrix(string); // parameter = name of the file with data in required format
19. size\_t get\_size();
20. string\* get\_cities();
21. double\*\* get\_weights();
22. size\_t get\_city\_index(string);
23. double FordBellman(string, string); // parameters = city from and to where between which the minimum distance is searched
24. };

## WeightsMatrix.cpp

1. #include "WeightsMatrix.h"
3. void WeightsMatrix::set\_size(size\_t size) { Size = size; }
4. void WeightsMatrix::set\_cities(string\* cities) { Cities = cities; }
5. void WeightsMatrix::set\_weights(double\*\* weights) { Weights = weights; }
6. WeightsMatrix::WeightsMatrix()
7. {
8. Size = 0;
9. Cities = nullptr;
10. Weights = nullptr;
11. }
12. WeightsMatrix::WeightsMatrix(string filename) // parameter = filename
13. {
14. set\_size(0);
15. fstream in(filename);
16. if (!in.is\_open())
17. throw invalid\_argument("file doesn't exist");
18. // firstly fill array of cities
19. // for that use queue
20. Queue<size\_t, string> cities;
21. string city;
22. while (in.peek() != EOF)
23. {
24. city = "";
25. while (in.peek() != ';') // read the name of the city from
26. city += in.get();
27. if (cities.search\_key(city) == -1) // if city is still not in queue of cities
28. cities.push(city); // add it to cities
29. in.get(); // read ';'
30. city = "";
31. while (in.peek() != ';') // read the name of the city to
32. city += in.get();
33. if (cities.search\_key(city) == -1)
34. cities.push(city);
35. while ((in.peek() != '\n') && (in.peek() != EOF))
36. in.get();
37. if (in.peek() == '\n') // read '\n'
38. in.get();
39. }
40. set\_size(cities.size());
41. Cities = new string[get\_size()]; //going to fill cities with cities from queue
42. for (size\_t i = 0; i < get\_size(); i++)
43. Cities[i] = cities.pop()->get\_data();
44. // filled Cities, now can declare the size of the weights matrix
45. Weights = new double\* [get\_size()];
46. for (size\_t i = 0; i < get\_size(); i++)
47. Weights[i] = new double[get\_size()];
48. // now going to initialize weights matrix with zeros for elements on the diagonal and with INF on other positions
49. for (size\_t i = 0; i < get\_size(); i++)
50. {
51. for (size\_t j = 0; j < get\_size(); j++)
52. {
53. if (i == j)
54. Weights[i][j] = 0;
55. else
56. Weights[i][j] = INF;
57. }
58. }
59. in.close();
60. in.open(filename);
61. while (in.peek() != EOF)
62. {
63. size\_t index\_from, index\_to; // indexes of cities in Cities
64. double forward\_cost, backward\_cost;
65. string info = "";
66. while (in.peek() != ';') // read city from
67. info += in.get();
68. index\_from = get\_city\_index(info);
69. in.get(); // read ';'
70. info = "";
71. while (in.peek() != ';') // read city to
72. info += in.get();
73. index\_to = get\_city\_index(info);
74. in.get(); // read ';'
75. if (in.peek() == 'N')
76. {
77. forward\_cost = INF; // unreal cost
78. while (in.get() != ';') {}
79. }
80. else
81. {
82. in >> forward\_cost;
83. in.get(); // read ';'
84. }
85. if (in.peek() == 'N')
86. {
87. backward\_cost = INF; // unreal cost
88. for (int i = 0; i < 3; i++)
89. in.get(); // read "N/A"
90. }
91. else
92. in >> backward\_cost;
93. if (in.peek() == '\n') // read '\n'
94. in.get();
95. if (forward\_cost < INF)
96. Weights[index\_from][index\_to] = forward\_cost;
97. if (backward\_cost < INF)
98. Weights[index\_to][index\_from] = backward\_cost;
99. // filled weights matrix
100. }
101. }
102. size\_t WeightsMatrix::get\_size() { return Size; }
103. string\* WeightsMatrix::get\_cities() { return Cities; }
104. double\*\* WeightsMatrix::get\_weights() { return Weights; }
106. size\_t WeightsMatrix::get\_city\_index(string city)
107. {
108. for (size\_t i = 0; i < get\_size(); i++)
109. {
110. if (Cities[i] == city)
111. return i;
112. }
113. return -1; // in case of not-existing city return unreal index
114. }
116. double WeightsMatrix::FordBellman(string source\_city, string target\_city)
117. {
118. ofstream out("out.txt");
119. size\_t source = get\_city\_index(source\_city);
120. size\_t target = get\_city\_index(target\_city);
121. if (source == -1)
122. {
123. out << "the source city doesn't exist" << endl;
124. throw invalid\_argument("the source city doesn't exist");
125. }
126. if (target == -1)
127. {
128. out << "the target city doesn't exist" << endl;
129. throw invalid\_argument("the target city doesn't exist");
130. }
131. double min\_cost = INF; // too big, unreal, cost of the flight between source and target cities
132. size\_t\* marks = new size\_t[get\_size()]{ source }; // remember indexes that means cities from what we came to indexed here city
133. double\*\* W = new double\* [get\_size()]; // create a copy of the original weights matrix to keep it safe
134. for (size\_t i = 0; i < get\_size(); i++)
135. W[i] = new double[get\_size()];
136. for (size\_t i = 0; i < get\_size(); i++)
137. {
138. for (size\_t j = 0; j < get\_size(); j++)
139. W[i][j] = get\_weights()[i][j];
140. }
141. size\_t phase = 0;
142. for (; phase < get\_size(); phase++) // guaranteed stops after size-1 times, so the last one shouldn't be reached because of variable is\_improoved
143. {
144. bool is\_improoved = false;
145. for (size\_t node = 0; node < get\_size(); node++)
146. {
147. for (size\_t to = 0; to < get\_size(); to++)
148. {
149. if (W[source][to] > W[source][node] + W[node][to])
150. {
151. W[source][to] = W[source][node] + W[node][to];
152. marks[to] = node;
153. is\_improoved = true;
154. }
155. }
156. }
157. if (!is\_improoved) // if on current phase wasn't any improvements
158. break;
159. }
160. if (phase == get\_size()) // unreal, so exists a negative cycle
161. {
162. out << "There is a negative cycle, program stops working." << endl;
163. throw invalid\_argument("Found negative cycle");
164. }
165. min\_cost = W[source][target];
167. if (min\_cost >= INF)
168. {
169. out << target\_city << " can't be reached from " << source\_city << endl;
170. throw invalid\_argument("path doesn't exist");
171. }
172. out << "minimum cost of reaching " << target\_city << " from " << source\_city << " is " << W[source][target] << endl;
173. // going to restore the way and print it to the file
174. size\_t\* reversed\_way = new size\_t[get\_size()]{ source }; // an array saving indexes of cities in reversed order
175. size\_t city = target;
176. int i = 0;
177. while (city != source)
178. {
179. reversed\_way[i] = city;
180. city = marks[city];
181. i++;
182. }
183. reversed\_way[i] = source;
184. out << "the way:" << endl;
185. for (int k = i; k >= 0; k--)
186. {
187. out << get\_cities()[reversed\_way[k]];
188. if (k != 0)
189. out << " -> ";
190. }
191. out.close();
192. return W[source][target];
193. }

## UnitTestFordBellman.cpp

1. #include "pch.h"
2. #include "CppUnitTest.h"
3. #include "../WeightsMatrix/WeightsMatrix.h"
5. using namespace Microsoft::VisualStudio::CppUnitTestFramework;
7. namespace UnitTestFordBellman
8. {
9. TEST\_CLASS(UnitTestFordBellman)
10. {
11. public:
12. TEST\_METHOD(search\_existing\_key\_in\_queue)
13. {
14. Queue<int, string> queue;
15. queue.push("A");
16. queue.push("B");
17. queue.push("C");
18. Assert::AreEqual(queue.search\_key("A"), 0);
19. Assert::AreEqual(queue.search\_key("B"), 1);
20. Assert::AreEqual(queue.search\_key("C"), 2);
21. }
23. TEST\_METHOD(search\_not\_existing\_key\_in\_queue)
24. {
25. Queue<int, string> queue;
26. queue.push("A");
27. queue.push("B");
28. queue.push("C");
29. Assert::AreEqual(queue.search\_key("D"), -1);
30. }
32. TEST\_METHOD(constructor\_3x3)
33. {
34. string filename = "in.txt";
35. string input = "A;B;N/A;1\nA;C;3;2\nB;C;1;4";
36. ofstream out(filename);
37. out << input;
38. out.close();
39. WeightsMatrix W(filename);
40. Assert::AreEqual(W.get\_size(), (size\_t)3);
41. Assert::AreEqual(W.get\_cities()[0], (string)"A");
42. Assert::AreEqual(W.get\_cities()[1], (string)"B");
43. Assert::AreEqual(W.get\_cities()[2], (string)"C");
44. Assert::AreEqual(W.get\_weights()[0][0], (double)0);
45. Assert::AreEqual(W.get\_weights()[0][1], INF);
46. Assert::AreEqual(W.get\_weights()[0][2], (double)3);
47. Assert::AreEqual(W.get\_weights()[1][0], (double)1);
48. Assert::AreEqual(W.get\_weights()[1][1], (double)0);
49. Assert::AreEqual(W.get\_weights()[1][2], (double)1);
50. Assert::AreEqual(W.get\_weights()[2][0], (double)2);
51. Assert::AreEqual(W.get\_weights()[2][1], (double)4);
52. Assert::AreEqual(W.get\_weights()[2][2], (double)0);
53. }
55. TEST\_METHOD(get\_existing\_city\_index)
56. {
57. string filename = "in.txt";
58. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
59. ofstream out(filename);
60. out << input;
61. out.close();
62. WeightsMatrix W(filename);
63. Assert::AreEqual(W.get\_city\_index("B"), (size\_t)1);
64. }
66. TEST\_METHOD(get\_not\_existing\_city\_index)
67. {
68. string filename = "in.txt";
69. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
70. ofstream out(filename);
71. out << input;
72. out.close();
73. WeightsMatrix W(filename);
74. Assert::AreEqual(W.get\_city\_index("D"), (size\_t)(-1));
75. }
77. TEST\_METHOD(Matrix\_3x3\_1)
78. {
79. string filename = "in.txt";
80. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
81. ofstream out(filename);
82. out << input;
83. out.close();
84. WeightsMatrix W(filename);
85. double answer = W.FordBellman("A", "C");
86. Assert::AreEqual(answer, (double)2);
87. }
88. TEST\_METHOD(Matrix\_3x3\_2)
89. {
90. string filename = "in.txt";
91. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
92. ofstream out(filename);
93. out << input;
94. out.close();
95. WeightsMatrix W(filename);
96. double answer = W.FordBellman("C", "B");
97. Assert::AreEqual(answer, (double)3);
98. }
100. TEST\_METHOD(Matrix\_3x3\_3)
101. {
102. string filename = "in.txt";
103. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
104. ofstream out(filename);
105. out << input;
106. out.close();
107. WeightsMatrix W(filename);
108. double answer = W.FordBellman("C", "A");
109. Assert::AreEqual(answer, (double)2);
110. }
112. TEST\_METHOD(Matrix\_3x3\_4)
113. {
114. string filename = "in.txt";
115. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
116. ofstream out(filename);
117. out << input;
118. out.close();
119. WeightsMatrix W(filename);
120. double answer = W.FordBellman("B", "A");
121. Assert::AreEqual(answer, (double)1);
122. }
124. TEST\_METHOD(Matrix\_3x3\_5)
125. {
126. string filename = "in.txt";
127. string input = "A;B;1;1\nA;C;3;2\nB;C;1;4";
128. ofstream out(filename);
129. out << input;
130. out.close();
131. WeightsMatrix W(filename);
132. double answer = W.FordBellman("B", "C");
133. Assert::AreEqual(answer, (double)1);
134. }
136. TEST\_METHOD(Matrix\_5x5\_1)
137. {
138. string filename = "in.txt";
139. string input = "A;B;2;1\nA;C;N/A;N/A\nA;D;4;-1\nA;E;5;N/A\nB;C;3;-2\nB;D;N/A;N/A\nB;E;4;N/A\nC;D;2;-1\nC;E;1;N/A\nD;E;1;N/A";
140. ofstream out(filename);
141. out << input;
142. out.close();
143. WeightsMatrix W(filename);
144. double answer = W.FordBellman("A", "B");
145. Assert::AreEqual(answer, (double)1);
146. }
148. TEST\_METHOD(Matrix\_5x5\_2)
149. {
150. string filename = "in.txt";
151. string input = "A;B;2;1\nA;C;N/A;N/A\nA;D;4;-1\nA;E;5;N/A\nB;C;3;-2\nB;D;N/A;N/A\nB;E;4;N/A\nC;D;2;-1\nC;E;1;N/A\nD;E;1;N/A";
152. ofstream out(filename);
153. out << input;
154. out.close();
155. WeightsMatrix W(filename);
156. double answer = W.FordBellman("A", "C");
157. Assert::AreEqual(answer, (double)3);
158. }
160. TEST\_METHOD(Matrix\_5x5\_3)
161. {
162. string filename = "in.txt";
163. string input = "A;B;2;1\nA;C;N/A;N/A\nA;D;4;-1\nA;E;5;N/A\nB;C;3;-2\nB;D;N/A;N/A\nB;E;4;N/A\nC;D;2;-1\nC;E;1;N/A\nD;E;1;N/A";
164. ofstream out(filename);
165. out << input;
166. out.close();
167. WeightsMatrix W(filename);
168. double answer = W.FordBellman("A", "D");
169. Assert::AreEqual(answer, (double)4);
170. }
172. TEST\_METHOD(Matrix\_5x5\_4)
173. {
174. string filename = "in.txt";
175. string input = "A;B;2;1\nA;C;N/A;N/A\nA;D;4;-1\nA;E;5;N/A\nB;C;3;-2\nB;D;N/A;N/A\nB;E;4;N/A\nC;D;2;-1\nC;E;1;N/A\nD;E;1;N/A";
176. ofstream out(filename);
177. out << input;
178. out.close();
179. WeightsMatrix W(filename);
180. double answer = W.FordBellman("A", "E");
181. Assert::AreEqual(answer, (double)4);
182. }
184. TEST\_METHOD(No\_way)
185. {
186. string filename = "in.txt";
187. string input = "A;C;3;N/A\nA;B;N/A;2\nC;B;N/A;1";
188. ofstream out(filename);
189. out << input;
190. out.close();
191. WeightsMatrix W(filename);
192. double\*\* bla = W.get\_weights();
193. string\* cities = W.get\_cities();
194. try
195. {
196. double answer = W.FordBellman("C", "B");
197. }
198. catch (const exception& message)
199. {
200. Assert::AreEqual(message.what(), "path doesn't exist");
201. }
202. }
204. TEST\_METHOD(Negative\_cycle)
205. {
206. string filename = "in.txt";
207. string input = "A;B;1;3\nA;C;2;N/A\nB;C;-1;-2";
208. ofstream out(filename);
209. out << input;
210. out.close();
211. WeightsMatrix W(filename);
212. double\*\* bla = W.get\_weights();
213. string\* cities = W.get\_cities();
214. try
215. {
216. double answer = W.FordBellman("A", "C");
217. }
218. catch (const exception& message)
219. {
220. Assert::AreEqual(message.what(), "Found negative cycle");
221. }
222. }
224. TEST\_METHOD(No\_such\_source)
225. {
226. string filename = "in.txt";
227. string input = "A;B;2;1\nA;C;N/A;N/A\nA;D;4;-1\nA;E;5;N/A\nB;C;3;-2\nB;D;N/A;N/A\nB;E;4;N/A\nC;D;2;-1\nC;E;1;N/A\nD;E;1;N/A";
228. ofstream out(filename);
229. out << input;
230. out.close();
231. WeightsMatrix W(filename);
232. try
233. {
234. double answer = W.FordBellman("F", "A");
235. }
236. catch (const exception& message)
237. {
238. Assert::AreEqual(message.what(), "the source city doesn't exist");
239. }
240. }
242. TEST\_METHOD(No\_such\_target)
243. {
244. string filename = "in.txt";
245. string input = "A;B;2;1\nA;C;N/A;N/A\nA;D;4;-1\nA;E;5;N/A\nB;C;3;-2\nB;D;N/A;N/A\nB;E;4;N/A\nC;D;2;-1\nC;E;1;N/A\nD;E;1;N/A";
246. ofstream out(filename);
247. out << input;
248. out.close();
249. WeightsMatrix W(filename);
250. try
251. {
252. double answer = W.FordBellman("A", "F");
253. }
254. catch (const exception& message)
255. {
256. Assert::AreEqual(message.what(), "the target city doesn't exist");
257. }
258. }
260. };
261. }