Задача 5

#include <iostream>

#include <sstream>

#include <list>

#include <array>

#include <vector>

**const** std::string constSpace = " "; //табуляция

**struct** Edge

{

**int** adjVertex;

**double** weight;

};

**class** Graph

{

**private**:

**int** v\_;

std::vector<std::list<Edge>> adj;

**public**:

Graph(**int** v);

**void** addEdge(**int** v, **int** adjVertex, **double** w);

**void** printShortPaths(**const** std::vector<**double**> &dist);

**void** printShortPathsTree(**const** std::vector<**int**> &parent);

**void** visit(std::vector<**bool**> &visited, Graph &gr, **int** v, std::string &space);

**void** bellmanFord(**int** source);

//факультативная задача

**bool** isCorrectShortPaths(**const** std::vector<**double**> &a, **int** source);

};

Graph::Graph(**int** v)

{

**this**->v\_ = v;

adj.resize(v\_);

}

**void** Graph::addEdge(**int** v, **int** adjVertex, **double** w)

{

adj[v].push\_back({adjVertex, w});

}

// Функция printShortPaths выводит в консоль кратчайшие пути из начальной вершины

**void** Graph::printShortPaths(**const** std::vector<**double**> &dist)

{

std::cout << "The weight of a shortest path from vertex 0 to vertex" << '\n';

**for** (**int** i = 0; i < v\_; i++)

{

**if** (dist[i] == INT\_MAX)

{

std::cout << i << " is inf" << '\n';

}

**else**

{

std::cout << i << " is " << dist[i] << '\n';

}

}

}

// Функция printShortPathsTree строит дерево кратчайших путей и затем, при помощи ф-и // visit выводит его в консоль.

**void** Graph::printShortPathsTree(**const** std::vector<**int**> &parent)

{

std::cout << "\nThe shortest-path tree:" << '\n';

Graph gr(parent.size());

**for** (**int** v = 0 ; v < parent.size(); v++)

{

**if** (parent[v] != -1)

{

gr.addEdge(parent[v], v, 0);

}

}

std::vector<**bool**> visited(parent.size(), **false**);

std::string space = "";

visit(visited, gr, 0, space);

}

//Функция visit осуществляет обход в глубину по дереву кратчайших путей.

**void** Graph::visit(std::vector<**bool**> &visited, Graph &gr, **int** v, std::string &space)

{

visited[v] = **true**;

std::cout << space << v << '\n';

**if** (gr.adj[v].empty())

{

**return**;

}

space += constSpace;

**for** (**auto** vertex: gr.adj[v])

{

**if** (v == 0)

{

space = constSpace;

}

**if** (!visited[vertex.adjVertex])

{

visit(visited, gr, vertex.adjVertex, space);

}

}

}

**void** Graph::bellmanFord(**int** source)

{

std::vector<**double**> dist(v\_, INT\_MAX);

std::vector<**int**> parent(v\_, -1);

dist[source] = 0;

**for** (**int** i = 0; i < v\_; i++)

{

**for** (**auto** j: adj[i])

{

**int** v = j.adjVertex;

**double** w = j.weight;

**if** (dist[i] != INT\_MAX && dist[i] + w < dist[v])

{

dist[v] = dist[i] + w;

parent[v] = i; //устанавливаем родителя "i" для вершины "v".

}

}

}

**for** (**int** i = 0; i < v\_; i++)

{

**for** (**auto** j: adj[i])

{

**int** v = j.adjVertex;

**double** w = j.weight;

**if** (dist[i] != INT\_MAX && dist[i] + w < dist[v])

{

std::cout << "A negative-weight cycle is reachable from the source vertex!";

**return**; //найдено ребро отрицательного веса.

}

}

}

printShortPaths(dist); //вывод кратчайших путей в консоль

printShortPathsTree(parent); //вывод дерева кратчайших путей в консоль

**return**;

}

**int** main()

{

**int** size = 0;

std::cin >> size;

std::cin.ignore();

Graph graph(size);

**for** (**int** i = 0; i < size; i++)

{

std::string str;

getline(std::cin, str);

std::stringstream stream(str);

std::string v;

stream >> v;

std::string u;

std::string w;

**int** v1 = stoi(v);

**while** (u != "-1")

{

stream >> u;

**if** (u != "-1")

{

stream >> w;

**int** u1 = stoi(u);

**double** w1 = stod(w);

graph.addEdge(v1, u1, w1);

}

}

}

std::cout << '\n';

graph.bellmanFord(0);

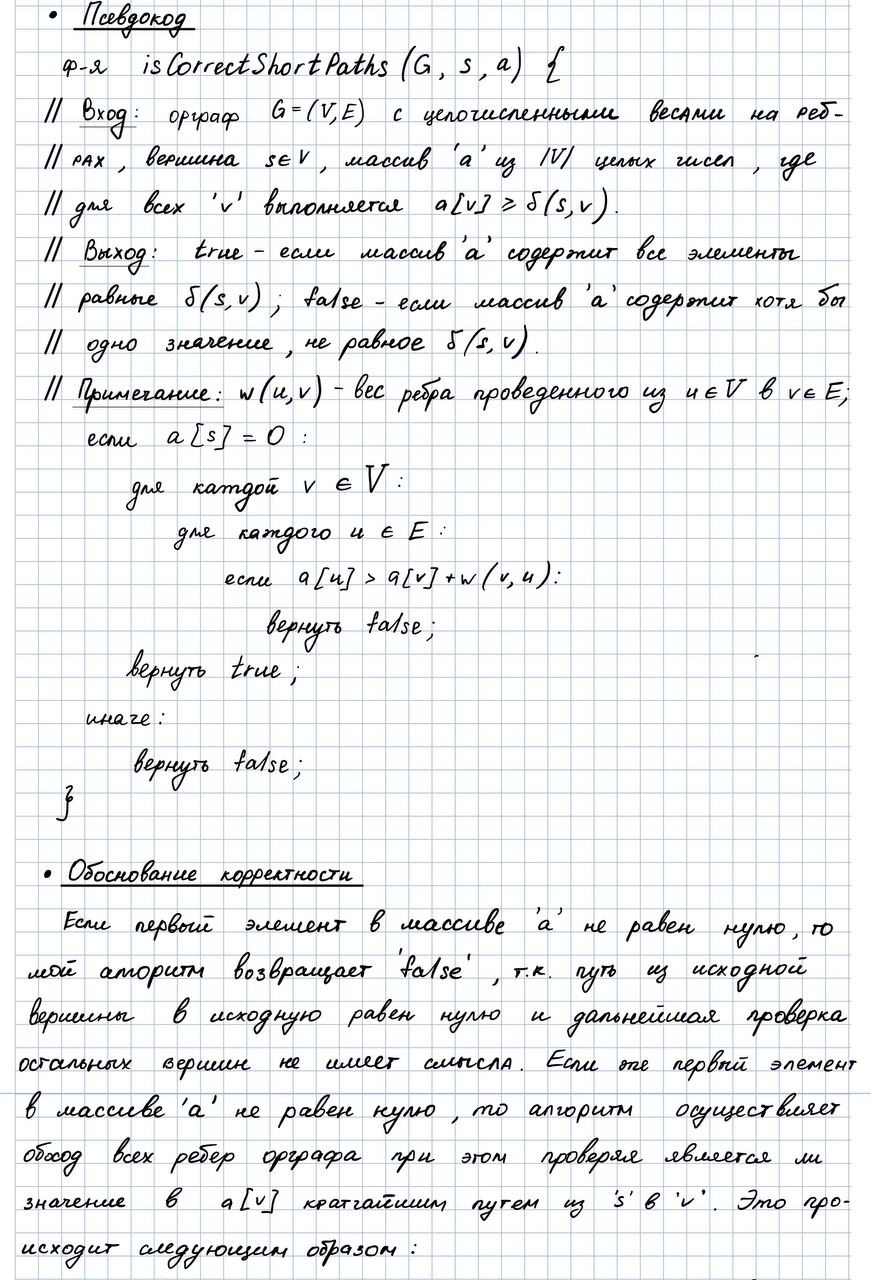
**return** 0;

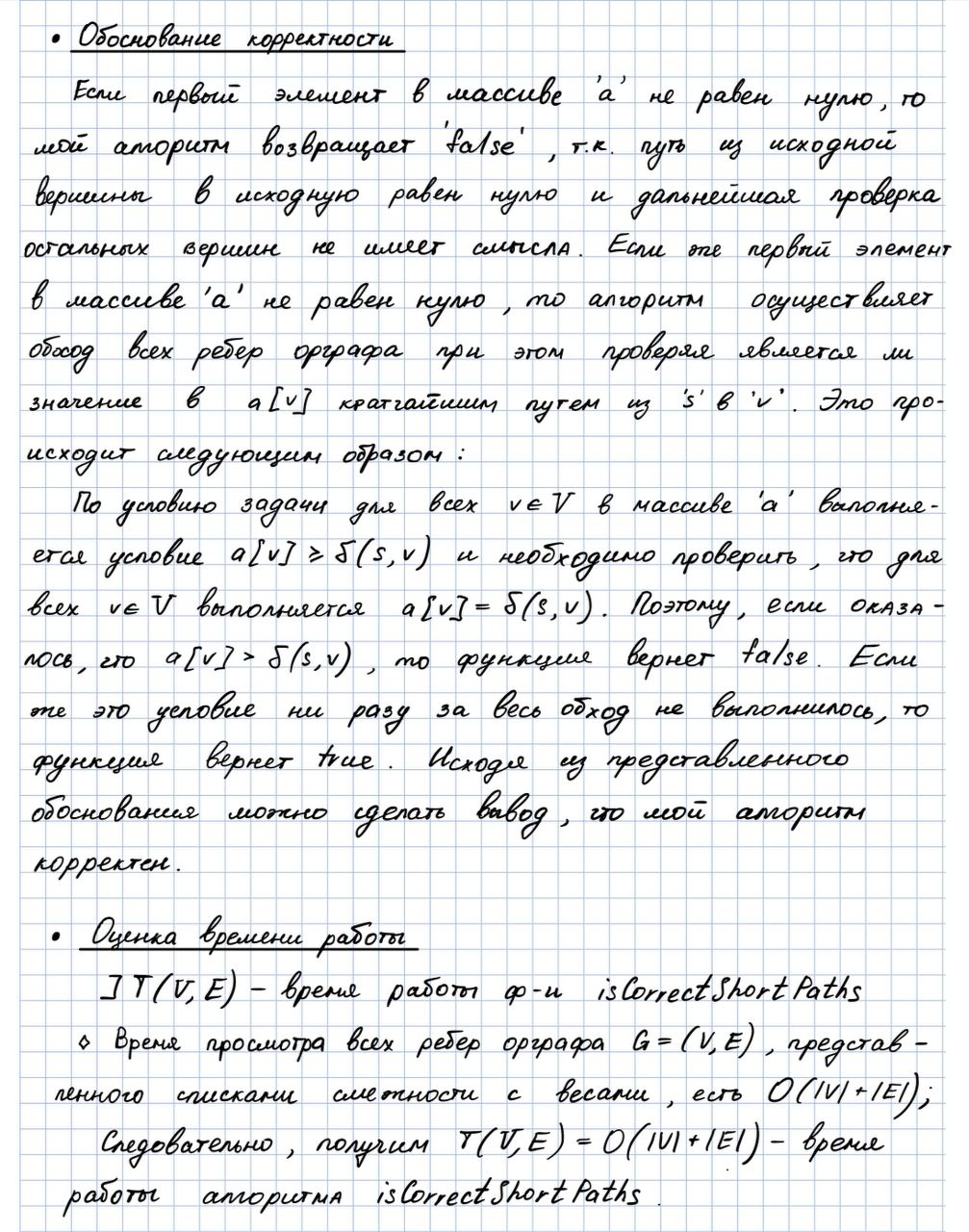
}

Тесты с визуализацией

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  |  |
| 8  0 1 1 4 4 5 8 -1  1 2 2 6 6 5 6 -1  2 3 1 6 2 -1  3 6 1 7 4 -1  4 5 5 -1  5 -1  6 5 1 7 1 -1  7 -1 | The weight of a shortest path from vertex 0 to vertex  0 is 0  1 is 1  2 is 3  3 is 4  4 is 4  5 is 6  6 is 5  7 is 6  The shortest-path tree:  0  1  2  3  6  5  7  4 |
|  |  |
| 5  0 1 10 2 10 3 30 4 50 -1  1 -1  2 3 10 4 30 -1  3 -1  4 1 40 3 20 -1 | The weight of a shortest path from vertex 0 to vertex  0 is 0  1 is 10  2 is 10  3 is 20  4 is 40  The shortest-path tree:  0  1  2  3  4 |
|  |  |
| 5  0 1 1 -1  1 2 5 3 2 -1  2 0 3 -1  3 2 1 4 10 -1  4 -1 | The weight of a shortest path from vertex 0 to vertex  0 is 0  1 is 1  2 is 4  3 is 3  4 is 13  The shortest-path tree:  0  1  3  2  4 |
|  |  |
| 5  0 1 3 2 5 -1  1 2 1 3 -2 4 -1 -1  2 4 1 -1  3 4 6 -1  4 -1 | The weight of a shortest path from vertex 0 to vertex  0 is 0  1 is 3  2 is 4  3 is 1  4 is 2  The shortest-path tree:  0  1  2  3  4 |

Задача 6





Факультативная задача 6А

**bool** Graph::isCorrectShortPaths(std::vector<**double**> &a, **int** source)

{

**if** (a[source] == 0)

{

**for** (**int** i = 0; i < v\_; i++)

{

**for** (**auto** j: adj[i])

{

**int** v = j.adjVertex;

**double** w = j.weight;

**if** (a[v] > a[i] + w)

{

**return** **false**;

}

}

}

**return** **true**;

}

**else**

{

**return** **false**;

}

}

Тесты с визуализацией

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
|  |  |
| 8  0 1 1 4 4 5 8 -1  1 2 2 6 6 5 6 -1  2 3 1 6 2 -1  3 6 1 7 4 -1  4 5 5 -1  5 -1  6 5 1 7 1 -1  7 -1  a {0, 1, 3, 4, 4, 6, 5, 6} | True |
|  |  |
| 5  0 1 10 2 10 3 30 4 50 -1  1 -1  2 3 10 4 30 -1  3 -1  4 1 40 3 20 -1  a{0, 10, 10, 30, 50} | False |
|  |  |
| 5  0 1 1 -1  1 2 5 3 2 -1  2 0 3 -1  3 2 1 4 10 -1  4 -1  a{0, 1, 6, 3, 13} | False |
|  |  |
| 5  0 1 3 2 5 -1  1 2 1 3 -2 4 -1 -1  2 4 1 -1  3 4 6 -1  4 -1  a{0, 3, 4, 1, 2} | True |