

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);
            }
        }
    }

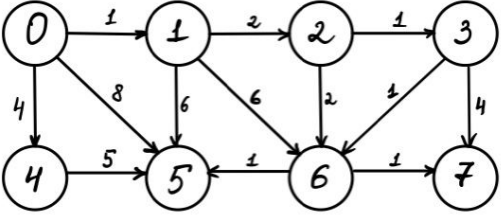
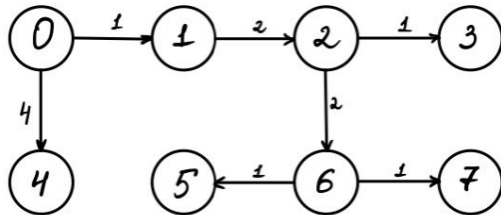
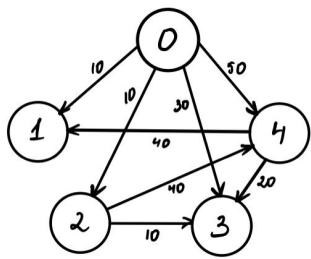
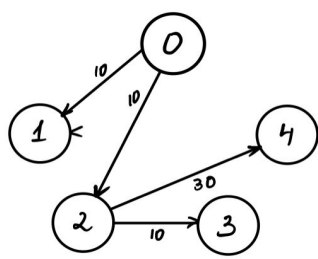
    std::cout << '\n';

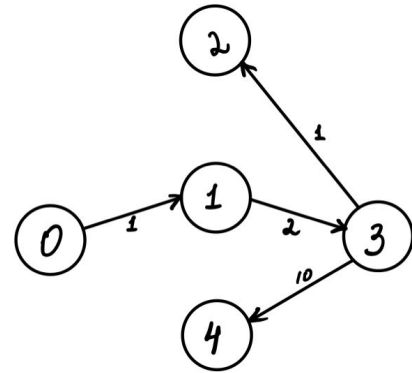
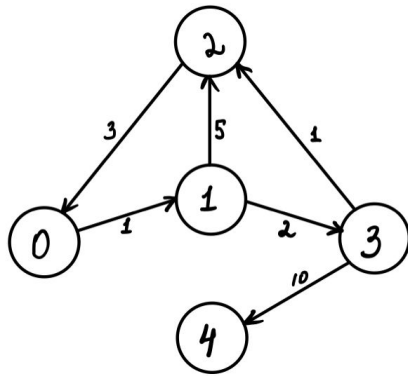
    graph.bellmanFord(0);

    return 0;
}

```

Тесты с визуализацией

| Входные данные | Выходные данные |
|---|--|
|  |  |
| <pre> 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 </pre> | <p>The weight of a shortest path from vertex 0 to vertex</p> <pre> 0 is 0 1 is 1 2 is 3 3 is 4 4 is 4 5 is 6 6 is 5 7 is 6 </pre> <p>The shortest-path tree:</p> <pre> 0 1 2 3 6 5 7 4 </pre> |
|  |  |
| <pre> 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 </pre> | <p>The weight of a shortest path from vertex 0 to vertex</p> <pre> 0 is 0 1 is 10 2 is 10 3 is 20 4 is 40 </pre> <p>The shortest-path tree:</p> <pre> 0 1 2 3 4 </pre> |



```

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

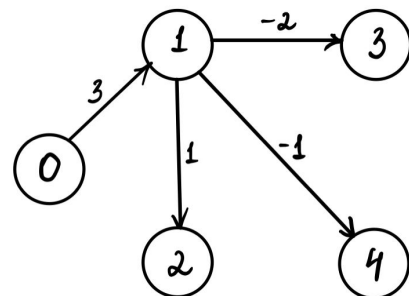
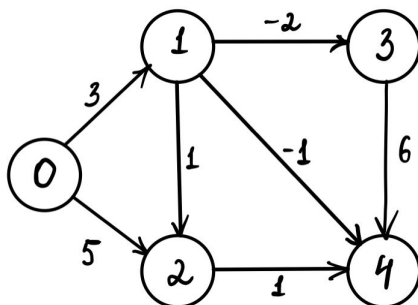
0 is 0
 1 is 1
 2 is 4
 3 is 3
 4 is 13

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

• Псевдокод

```
Ф-я isCorrectShortPaths ( $G, s, a$ ) {  
  // Вход: граф  $G=(V,E)$  с целочисленными весами на реб-  
  // рах, вершина  $s \in V$ , массив 'a' из  $|V|$  целых чисел, где  
  // для всех 'v' выполняется  $a[v] \geq \delta(s, v)$ .  
  // Выход: true - если массив 'a' содержит все элементы  
  // равные  $\delta(s, v)$ ; false - если массив 'a' содержит хотя бы  
  // одно значение, не равное  $\delta(s, v)$ .  
  // Примечание:  $w(u, v)$  - вес ребра проведенного из  $u \in V$  в  $v \in E$ ;  
  // если  $a[s] = 0$ :  
  //   для каждой  $v \in V$ :  
  //     для каждого  $u \in E$ :  
  //       если  $a[u] > a[v] + w(v, u)$ :  
  //         вернуть false;  
  //       вернуть true;  
  // иначе:  
  //   вернуть false;  
}
```

• Обоснование корректности

Если первый элемент в массиве 'a' не равен нулю, то наш алгоритм возвращает 'false', т.к. путь из исходной вершины в исходную равен нулю и дальнейшая проверка остальных вершин не имеет смысла. Если же первый элемент в массиве 'a' не равен нулю, то алгоритм осуществляет обход всех ребер графа при этом проверяя является ли значение в $a[v]$ кратчайшим путем из 's' в 'v'. Это происходит следующим образом:

По условию задачи для всех $v \in V$ в массиве 'a' выполняется условие $a[v] \geq \delta(s, v)$ и необходимо проверить, что для всех $v \in V$ выполняется $a[v] = \delta(s, v)$. Поэтому, если окажется, что $a[v] > \delta(s, v)$, то функция вернет false. Если же это условие ни разу за весь обход не выполнилось, то функция вернет true. Исходя из представленного обоснования можно сделать вывод, что мой алгоритм корректен.

- Оценка времени работы

$T(V, E)$ - время работы ф-и `isCorrectShortPaths`

♦ Время просмотра всех ребер орграфа $G = (V, E)$, представленного списками смежности с весами, есть $O(|V| + |E|)$;

Следовательно, получим $T(V, E) = O(|V| + |E|)$ - время работы алгоритма `isCorrectShortPaths`.

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 |
|---|-----------------|
| | |
| <pre>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}</pre> | True |
| | |

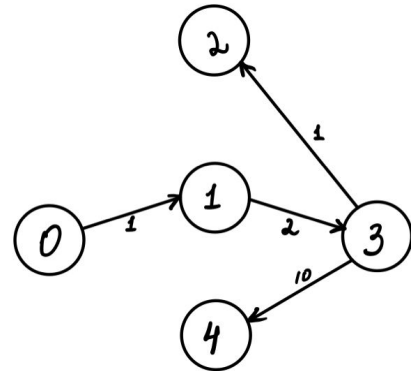
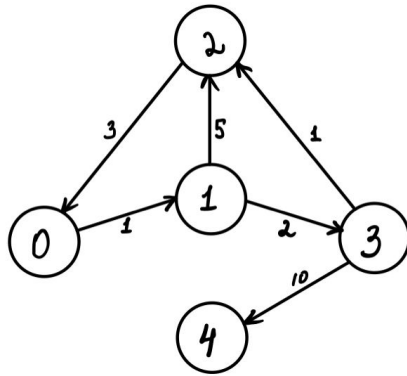

```

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



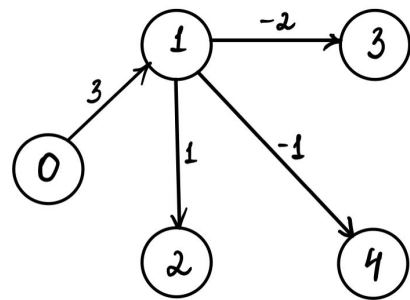
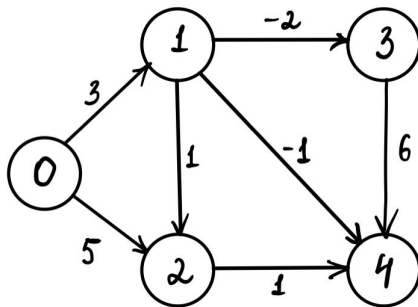
```

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