**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

дисциплина: Дискретная математика

Лабораторная работа №4.4

тема: «Кратчайшие пути во взвешенном орграфе»

|  |  |
| --- | --- |
|  | Выполнил: ст. группы ВТ-202  Проверил: Рязанов Ю.Д. |

Белгород 2021

Изображение выглядит как текст

Автоматически созданное описание

**Задание 2:**

2. Используя алгоритм Дейкстры, разработать и реализовать алгоритм решения задачи (см. варианты заданий).

**Вариант 3**

Изображение выглядит как текст, стол

Автоматически созданное описание

**Текст программы:**

#include <iostream>

#include <vector>

using namespace std;

const size\_t Max = 10000;

//ввод взвешеного орграфа в матрицу смежности

void inputWeightDigraph(vector<vector<size\_t>>& a){

size\_t i, j;

size\_t weight;

while (cin >> i >> j >> weight) {

a[i - 1][j - 1] = weight;

}

cin.clear();

}

//алгоритм Дийгстры по по поиску кратчайшего пути

void digstrsAlgoritm(const vector<vector<size\_t>>& graph, const size\_t v1, const size\_t v\_last, vector<size\_t>& travel, size\_t& sizeOftravel) {

vector<bool> markedVertice(graph.size(), 0);

vector<size\_t> minDistans(graph.size(), Max);

vector<int> treeOfShorRath(graph.size(), -1);

size\_t lastMinVetrx = v1;

markedVertice[v1] = 1;

minDistans[v1] = 0;

treeOfShorRath[v1] = -1;

bool f = true;

while (lastMinVetrx != v\_last && f) {

f = false;

for (size\_t i = 0; i < graph.size(); i++) {

if (graph[lastMinVetrx][i] != 0 && minDistans[lastMinVetrx] + graph[lastMinVetrx][i] < minDistans[i]) {

minDistans[i] = minDistans[lastMinVetrx] + graph[lastMinVetrx][i];

treeOfShorRath[i] = lastMinVetrx;

}

}

size\_t min = Max;

size\_t i;

for (i = 0; i < graph.size(); i++) {

if (!markedVertice[i] && minDistans[i] < min) {

min = minDistans[i];

lastMinVetrx = i;

f = true;

}

}

markedVertice[lastMinVetrx] = 1;

}

if (f) {

sizeOftravel = minDistans[lastMinVetrx];

while (treeOfShorRath[lastMinVetrx] != -1) {

travel.push\_back(lastMinVetrx);

lastMinVetrx = treeOfShorRath[lastMinVetrx];

};

travel.push\_back(v1);

for (size\_t i = 0; i < travel.size() / 2; i++) {

swap(travel[i], travel[travel.size() - i - 1]);

}

}

}

int main() {

vector<vector<size\_t>> a;

size\_t n;

cout << "Enter count of vetrix: ";

cin >> n;

a.resize(n);

for (size\_t i = 0; i < n; i++) {

a[i].resize(n, 0);

}

size\_t v1, minLen;

cout << "Enter first vetrix: ";

cin >> v1;

cout << "Enter min len: ";

cin >> minLen;

bool f = false;

inputWeightDigraph(a);

for (size\_t i = 0; i < a.size(); i++) {

size\_t sizeOftravel;

vector<size\_t> travel;

digstrsAlgoritm(a, v1 - 1, i, travel, sizeOftravel);

if (travel.size() && sizeOftravel > minLen) {

cout << "the shortest road more then " << minLen <<" from " << v1 << " to " << i + 1 << ":\n";

for (int i = 0; i < travel.size(); i++) {

cout << travel[i] + 1 << " ";

}

cout << endl << "len: " << sizeOftravel << endl;

f = true;

}

}

if (!f) {

cout << "not found conections.";

}

return 0;

}

**Задание 3:**

3. Подобрать тестовые данные. Результат представить в виде диаграммы графа.

**Тестовые данные**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | | | **Результат** | |
| Граф | Первая вершина | Мин длинна | Множество вершин | Длинна |
| 1 | 4  2 1 7  3 1 8  4 2 9  4 3 10 | 1 | 1 | not found conections. | - |
| 2 | 6  1 2 1  1 5 1  2 4 1  2 3 1  5 6 1 | 1 | 6 | not found conections. | - |
| 3 | 6  1 2 3  2 3 13  2 4 4  3 5 12  3 1 8  4 6 5  5 2 11  5 4 7  6 3 6 | 3 | 15 | 3 1 2 4 6 | 20 |
| 4 | 6  1 2 1  1 3 9  2 3 7  2 4 2  3 5 1  4 3 4  4 5 8  4 6 2  5 6 5  6 5 4 | 1 | 6 | 1 2 4 3 | 7 |
| 1 2 4 3 5 | 8 |

**Результаты работы**

**1.**

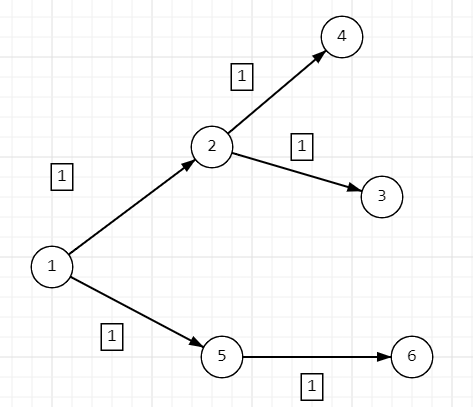
Изображение выглядит как текст, белый, провод

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**2.**



Изображение выглядит как текст

Автоматически созданное описание

**3.**

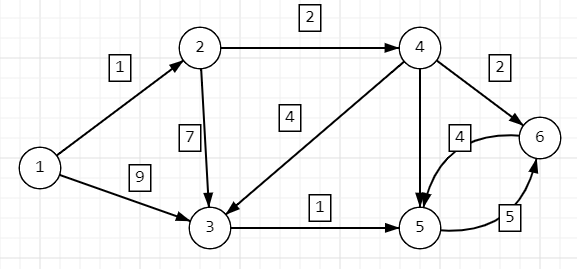
Изображение выглядит как текст, стена, внутренний, белый

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**4.**



Изображение выглядит как текст

Автоматически созданное описание