МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Частное учреждение образования

«Гродненский колледж бизнеса и права»

**Лабораторная работа № 12**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

**Тема:** Решение задач на нахождение кратчайших расстояний

для учащихся 2 курса специальности

2-40 01 01 «Программное обеспечение информационных технологий»

**ЛАБОРАТОРНАЯ РАБОТА № 12**

Тема: Решение задач на нахождение кратчайших расстояний.

Цель:

Образовательная**:**

* Обучить основным алгоритмам обхода графа и научиться решать задачи обхода графа на основе поиска в ширину и поиска в глубину,

Развивающая:

* научить анализировать алгоритмы обхода графа и научить решать задачи обхода графа на основе поиска в ширину и поиска в глубину,
* создать условия для развития способности четко формулировать свои мысли.

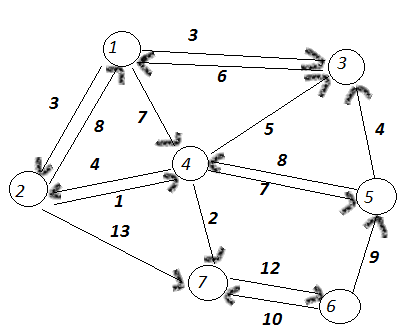
Воспитательная:

* воспитывать в обучающихся средствами урока уверенность в своих силах;

воспитывать сознательное и серьёзного отношения обучающихся к учебной дисциплине, убеждая их в том, что полученные знания пригодятся им в будущей деятельности.

Задачи: Освоение теоретического материала и выполнение индивидуального задания.

**ЗАДАЧИ**

Условие: Найти кратчайший путь из 1-вой вершины ко всем остальным, используя алгоритм Дейкстри

Алгоритм: Предоставлен преподавателю в письменном виде.

Решение:

**uses** crt;

**const**

v = 7;

inf = 100000;

**type**

vector = **array**[1..V] **of** integer;

**var**

start: integer;

**const**

GR: **array**[1..V, 1..V] **of** integer = (

(0, 3, 3, 7, 0, 0, 0),

(8, 0, 0, 1, 0, 0, 13),

(6, 0, 0, 0, 0, 0, 0),

(0, 4, 5, 0, 7, 0, 2),

(0, 0, 4, 8, 0, 8, 0),

(0, 0, 0, 0, 9, 10, 0),

(0, 0, 0, 0, 0, 12, 0));

**procedure** Dijkstra(st: integer);

**var**

count, index, i, u, m, min: integer;

distance: vector;

visited: **array**[1..V] **of** boolean;

**begin**

m := st;

**for** i := 1 **to** v **do**

**begin**

distance[i] := inf;

visited[i] := false;

**end**;

distance[st] := 0;

**for** count := 1 **to** v - 1 **do**

**begin**

min := inf;

**for** i := 1 **to** V **do**

**if** (**not** visited[i]) **and** (distance[i] < +min) **then**

**begin**

min := distance[i];

index := i;

**end**;

u := index;

visited[u] := true;

**for** i := 1 **to** v **do**

**if** (**not** visited[i]) **and** (GR[u, i] <> 0) **and** (distance[u] <> inf) **and** (distance[u] + GR[u, i] < distance[i]) **then**

distance[i] := distance[u] + GR[u, i];

**end**;

writeln('СТоимость пути из начальной вершины до остальных: ');

**for** i := 1 **to** v **do**

**if** distance[i] <> inf **then**

writeln(m, ' > ', i, ' = ', distance[i])

**else**

writeln(m, ' > ', i, ' = ', 'Маршрут недоступен.');

**end**;

**begin**

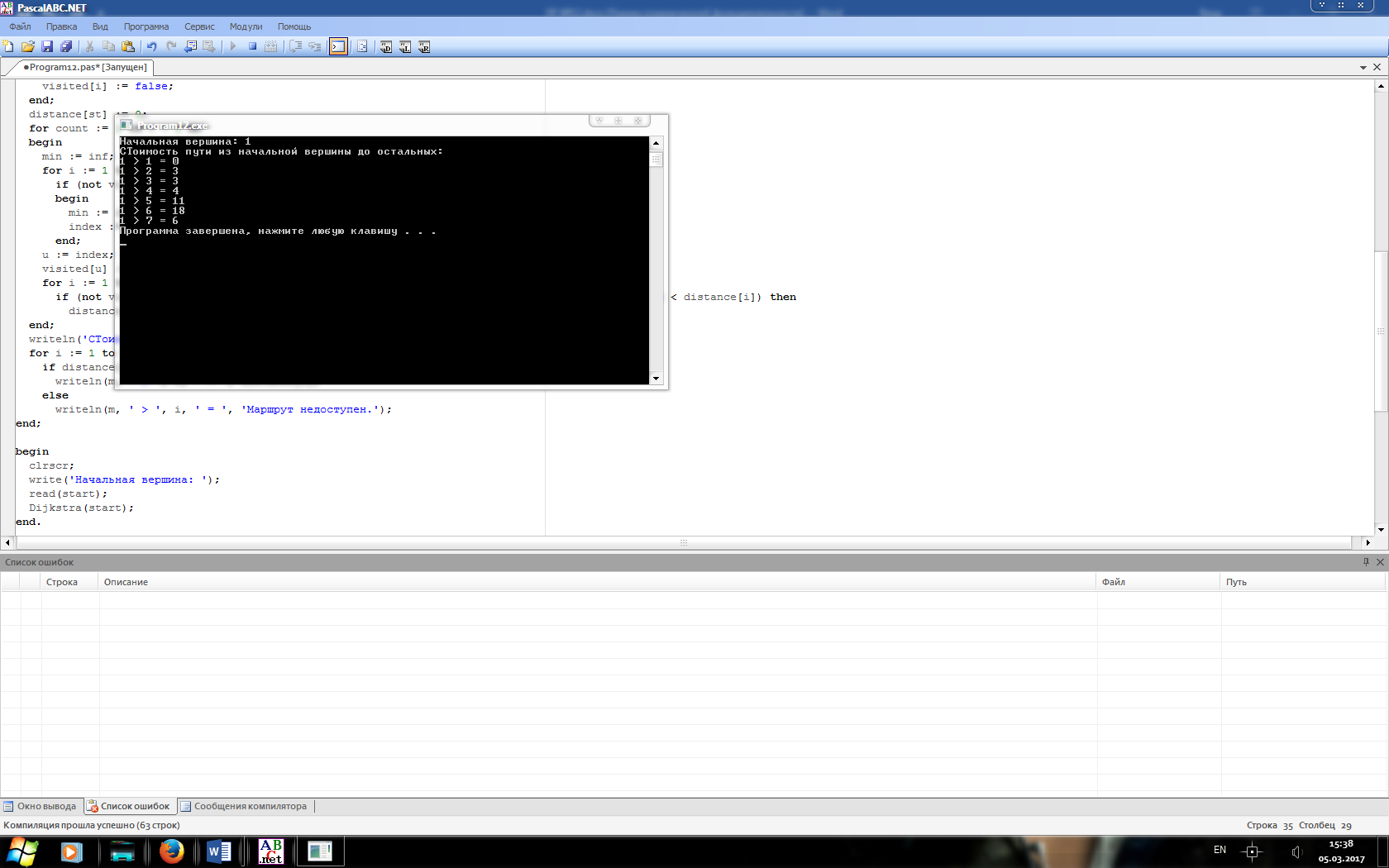
clrscr;

write('Начальная вершина: ');

read(start);

Dijkstra(start);

**end**.

**ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Как связаны между собой различные способы представления графов?

* [**связным**](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84), если для любых вершин u {\displaystyle u} , v {\displaystyle v} есть путьv {\displaystyle v} .
* **сильно связным** или **ориентированно связным**, если он ориентированный, и из любой вершины в любую другую имеется ориентированный путь.
* [**деревом**](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%29), если он связный и не содержит нетривиальных циклов.
* [**полным**](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84), если любые его две (различные, если не допускаются петли) вершины соединены ребром.
* [**двудольным**](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B3%D1%80%D0%B0%D1%84), если его вершины можно разбить на два непересекающихся подмножества V 1 {\displaystyle V\_{1}} и V 2 {\displaystyle V\_{2}} так, что всякое ребро соединяет вершину из V 1 {\displaystyle V\_{1}} с вершиной из V 2 {\displaystyle V\_{2}} .

1. Приведите алгоритм Дейкстри.

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим [шаг алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B#.D0.A8.D0.B0.D0.B3).

1. Приведите алгоритм Флойда.

Имеется кратчайший путь p1k=(v1,v2,… ,vk) от вершины v1 до вершины vk, а также его подпуть p'(vi,vi+1,… ,vj), при этом действует 1 <= i <= j <= k.

Если p — кратчайший путь от v1 до vk, то p' также является кратчайшим путем от вершины vi до vj  
Это можно легко доказать, так как стоимость пути p складывается из стоимости пути p' и стоимости остальных его частей. Так вот представив что есть более короткий путь p', мы уменьшим эту сумму, что приведет к противоречию с утверждением, что эта сумма и так уже была минимальной.  
Второе свойство является основой алгоритма. Мы рассматриваем граф G с пронумерованными от 1 до n вершинами {v1,v2,… ,vn} и путь pij от vi до vj, проходящий через определенное множество разрешенных вершин, ограниченное индексом k.   
То есть если k=0, то мы рассматриваем прямые соединения вершин друг с другом, так как множество разрешенных промежуточных вершин рано нулю. Если k=1 — мы рассматриваем пути, проходящие через вершину v1, при k=2 — через вершины {v1, v2}, при k=3 — {v1, v3, v3} и так далее.