БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра программного обеспечения информационных технологий

Факультет ФНиДО

Специальность ПОИТ

Индивидуальная практическая работа № 2.3

«Ориентированные графы»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнила: Карпеко Н. Г.

Договор № 941 от 20.02.2019 г.

Минск 2020

**Цель работы**: научиться обрабатывать элементы ориентированных графов, находить в них заданные маршруты.

**Задание.** Представить ориентированный граф, состоящий из 7-10 вершин, с помощью матрицы смежности. Указать вершину-источник, а затем решить следующие задачи.

1. Кратчайшие пути от вершины-источника до всех вершин орграфа на основе алгоритма Дейкстры.
2. Кратчайшие расстояния между каждой парой вершин орграфа на основе алгоритма Флойда.

Листинг программы:

**program** Project2;

**uses**

crt;

**const**

N = 5; inf = 100000;

**type**

TGraph = **array**[1..N, 1..N] **of** integer;

TVisited = **array**[1..N] **of** boolean;

**const** Graph: TGraph = (

(0, 4, 2, 0, 0),

(0, 0, 0, 1, 2),

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

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

(0, 0, 0, 0, 0)

);

**procedure** Dijkstra(**const** Graph: TGraph; **const** start, last: integer);

**var**

i, j, index, min: integer;

distance: **array**[1..N] **of** integer;

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

**begin**

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

**begin**

distance[i] := inf;

visited[i] := false;

**end**;

distance[start] := 0;

**for** j := 1 **to** N - 1 **do**

**begin**

min := inf;

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

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

**begin**

min := distance[i];

index := i;

**end**;

visited[index] := true;

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

**if** (**not** visited[i]) **and** (Graph[index, i] <> 0) **and** (distance[index] <> inf) **and**

(distance[index] + Graph[index, i] < distance[i]) **then**

distance[i] := distance[index] + Graph[index, i];

**end**;

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

for i := 1 to V do

if distance[i] <> inf then

writeln(start,' - ', i,' = ', distance[i])

else

writeln(start,' - ', i,' = ', 'маршрут недоступен');}

**if** (last <= N) **and** (distance[last] <> inf) **then**

writeln(start, ' - ', last, ' = ', distance[last])

**else**

writeln('Такого пути нет');

**end**;

**procedure** FindMinWay(**const** Graph: TGraph);

**var**

start, last: integer;

**begin**

//while True do

**begin**

write(' Начальная вершина ');

readln(start);

write(' Конечная вершина ');

readln(last);

Dijkstra(Graph, start, last);

**end**;

**end**;

//---

**procedure** CentreGraph(Graph: TGraph; **const** N: integer);

**type**

TTemp = **record**

str, stl, value: integer;

**end**;

**var**

i, j: integer;

ArrTemp: **array of** TTemp;

Temp: TTemp;

**begin**

Setlength(ArrTemp, N);

**for** j := 1 **to** N **do**

**begin**

Temp.value := Graph[1, j];

Temp.stl := j;

Temp.str := 1;

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

**begin**

**if** Graph[i, j] > Temp.value **then**

**begin**

Temp.value := Graph[i, j];

Temp.stl := j;

Temp.str := i;

**end**;

**end**;

ArrTemp[j - 1] := Temp;

**end**;

Temp := ArrTemp[0];

**for** i := 0 **to** N - 1 **do**

**if** ArrTemp[i].value < Temp.value **then**

**begin**

Temp.value := ArrTemp[i].value;

Temp.stl := ArrTemp[i].stl;

Temp.str := ArrTemp[i].str;

**end**;

Writeln(' Центр графа: Значение=', Temp.value, ', Строка=', Temp.str, ', Столбик=', Temp.stl);

**end**;

**procedure** Floid(Graph: TGraph; **const** N: integer);

**var**

i, j, k : integer;

**begin**

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

**for** j := 1 **to** N **do**

**begin**

**if** Graph[i, j] = 0 **then**

Graph[i, j] := inf;

**if** i = j **then**

Graph[i, j] := 0;

**end**;

**for** k := 1 **to** N **do**

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

**for** j := 1 **to** N **do**

**if** Graph[i, k] + Graph[k, j] < Graph[i, j] **then**

Graph[i, j] := Graph[i, k] + Graph[k, j];

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

**begin**

**for** j := 1 **to** N **do**

write(Graph[i, j]: 8, ' ');

writeln;

**end**;

CentreGraph(Graph, N);

**end**;

//------------------

**procedure** RoundVertex(**const** Graph: TGraph; **var** visited: TVisited; v: integer);

**var**

w: integer;

**begin**

visited[v] := true;

write(v, ' ');

**for** w := 1 **to** N **do**

**if** (visited[w] = false) **and** (Graph[v, w] <> 0) **then**

RoundVertex(Graph, visited, w);

**end**;

**procedure** RoundGraph(**const** Graph: TGraph);

**var**

v: integer;

visited: TVisited;

**begin**

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

visited[v] := false;

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

**if** visited[v] = false **then**

RoundVertex(Graph, visited, v);

writeln;

**end**;

//------------------------

**procedure** AllWayVertex(**const** Graph: TGraph; v: integer; **var** s: string);

**var**

w: integer;

**begin**

s := s + inttostr(v);

writeln(s);

**for** w := 1 **to** N **do**

**if** (Graph[v, w] <> 0) **then**

AllWayVertex(Graph, w, s);

delete(s, length(s), 1);

**end**;

**procedure** AllWays(**const** Graph: TGraph);

**var**

v: integer;

s: string;

**begin**

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

**begin**

s := '';

AllWayVertex(Graph, v, s);

writeln;

**end**;

**end**;

//------------

**procedure** WayBetweenVertices(**const** Graph: TGraph; v: integer; **var** s: string; **const** last: integer);

**var**

w: integer;

**begin**

s := s + inttostr(v);

**if** s[length(s)] = inttostr(last) **then**

writeln(s);

**for** w := 1 **to** N **do**

**if** (Graph[v, w] <> 0) **then**

WayBetweenVertices(Graph, w, s, last);

delete(s, length(s), 1);

**end**;

**procedure** WaysVertices(**const** Graph: TGraph; **const** start, last: integer);

**var**

s: string;

**begin**

**begin**

s := '';

WayBetweenVertices(Graph, start, s, last);

writeln;

**end**;

**end**;

//-------------------------

**function** WeightOfWay(**const** Graph: TGraph; **const** s: string): integer;

**var**

i: integer;

**begin**

Result := 0;

**for** i := 1 **to** length(s) - 1 **do**

Result := Result + Graph[strtoint(s[i]), strtoint(s[i + 1])];

**end**;

//---------------------

**procedure** AdjacentVertices(**const** Graph: TGraph; **const** Vertix: integer);

**var**

i: Integer;

**begin**

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

**begin**

**if** Graph[i, Vertix] <> 0 **then**

**begin**

writeln(i, ' смежна с ', Vertix, ' - ', WeightOfWay(Graph, inttostr(i) + inttostr(Vertix)));

**end**;

**end**;

**end**;

//--------------------------------

**begin**

writeln(WeightOfWay(Graph, inttostr(1)));

writeln(WeightOfWay(Graph, inttostr(13)));

WaysVertices(Graph, 1, 4);

AllWays(Graph);

RoundGraph(Graph);

Floid(Graph, N);

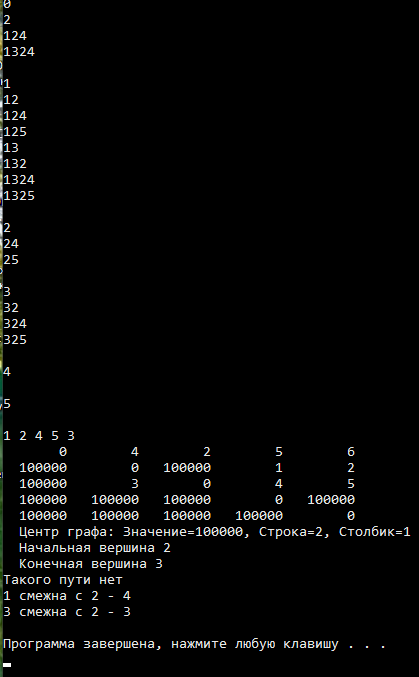
FindMinWay(Graph); // Дейкстра

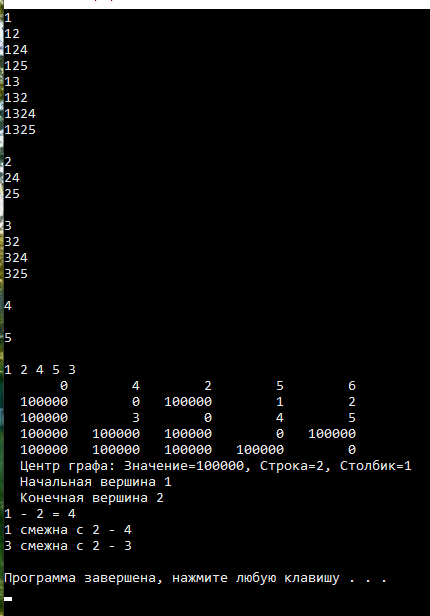
AdjacentVertices(Graph, 2);

readln;

**end**.

Результат работы программы:





**Контрольные вопросы:**

1. Для решения какой задачи на ориентированном графе удобно использовать алгоритм Дейкстры?

Алгоритм Дейкстры удобно использовать для задач поиска стоимости кратчайших путей от источника ко всем другим вершинам графа *G*. Длина пути здесь равна сумме стоимостей дуг, составляющих путь. Эта задача - нахождение кратчайшего пути с одним источником (длина пути м измеряться и в нелинейных единицах (временных)).

1. Для решения какой задачи на ориентированном графе удобно использовать алгоритм Флойда?

Алгоритм Флойда удобно использовать для решения задачи нахождения кратчайших путей - найти для каждой упорядоченной пары вершин (*v, w)* любого пути от вершины *v* в вершины *w*, длина кот-го минимальна среди всех возможных путей от *v* к *w*.

1. Какая вершина графа называется его центром?

Множество всех центральных вершин графа называется его центром. Центром может быть единственная вершина графа или несколько вершин.