НТУУ «КПІ»

ФІОТ

Кафедра ОТ

# Лабораторна робота №4

Виконав:

студент групи ІО-34

Власов Максим

Номер залікової книжки:

3405

Київ 2014р.

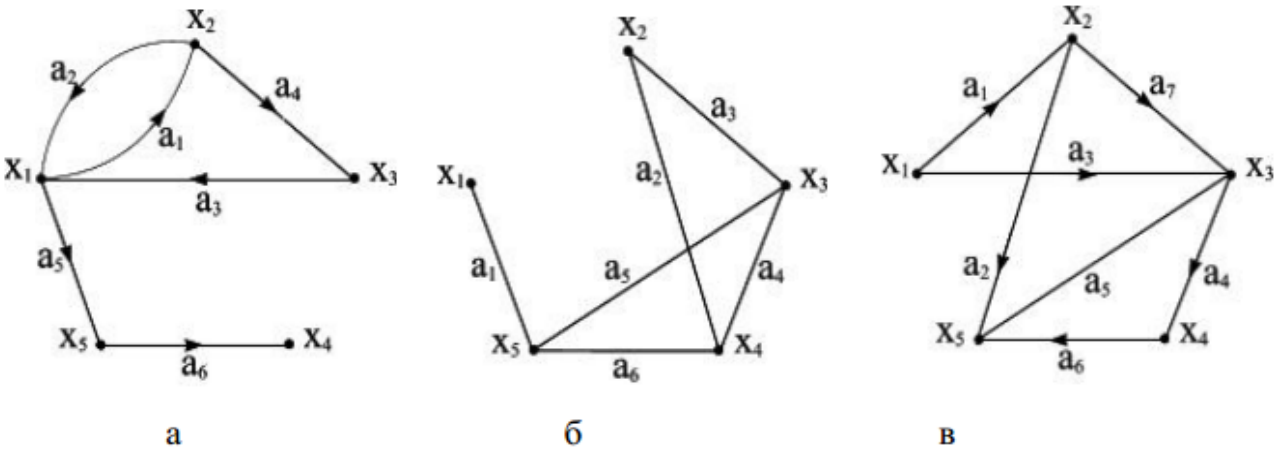
**Тема:** «Графи. Способи представлення графів. Остовні дерева. Пошук найкоротших шляхів»
**Мета роботи:** Вивчення властивостей графів, способів їх представлення та основних алгоритмів на графах.
**Завдання:** створити програму, яка реалізує один з алгоритмів на графах.

**I = 3405 mod 8 + 1 = 6**

**Теорія:**

**Граф** $G$ задається множиною точок або вершин $x_1, x_2, ..., x_n$ (яке позначається через $X$) і множиною ліній або ребер $a_1, a_2, ...,$ (яке позначається символом $A$), що з'єднують між собою всі або частину цих точок. Таким чином, граф $G$ повністю задається (і позначається) парою $(X, A)$.

Якщо ребра з множини $A$ орієнтовані, що зазвичай показується стрілкою, то вони називаються дугами, і граф з такими ребрами називається орієнтованим графом (рис. 4.1 (а)). Якщо ребра не мають орієнтації, то граф називається неорієнтованим (рис. 4.1 (б)). У разі, коли $G = (X, A)$ є орієнтованим графом і ми хочемо знехтувати спрямованістю дуг з множини $A$, то неорієнтований граф, відповідний $G$, будемо позначати як $G = (X, A)$.




а б в

Рис. 4.1 (а) – орієнтований граф; (б) – неорієнтований граф; (в) – змішаний граф

Якщо дуга позначається впорядкованою парою, що складається з початкової та кінцевої вершин (тобто двома кінцевими вершинами дуги), її напрямок передбачається заданим від першої вершини до другої. Так,

наприклад, на рис.4.1 (а) позначення $(x_1, x_2)$ відноситься до дуги $a_1$, а $(x_2, x_1)$ – до дуги $a_2$.

Інший, вживаний частіше, опис орієнтованого графа $G$ полягає у задаванні множини вершин $X$ і *відповідності* Г, яка показує, як між собою пов'язані вершини. Відповідність Г називається відображенням множини $X$ в $X$, а граф в цьому випадку позначається парою $G = (X, Г)$.

Для графа на рис.4.1 (а) маємо $Г(x_1) = \{x_2, x_5\}$, тобто вершини $x_2$ і $x_5$ є кінцевими вершинами дуг, у яких початковою вершиною є $x_1$.

$Г(x_2) = \{x_1, x_3\}$, $Г(x_3) = \{x_1\}$, $Г(x_4) = \varnothing$ - порожня множина, $Г(x_5) = \{x_4\}$

У разі неорієнтованого графа або графа, що містить і дуги, і неорієнтовані ребра (див., наприклад, графи, зображені на рис.4.1 (б) і рис.4.1 (в)), передбачається, що відповідність Г задає такий еквівалентний орієнтований граф, який отримуємо з вихідного графа заміною кожного неорієнтованого ребра двома протилежно спрямованими дугами, що з'єднують ті ж самі вершини. Так, наприклад, для графа, наведеного на рис.4.1 (б), маємо $Г(x_5) = \{x_1, x_3, x_4\}$, $Г(x_1) = \{x_5\}$ і ін.

**Метод топологічного сортування:**

У деяких випадках початковий граф є ациклічним, але має неправильну нумерацію – містить дуги $(x_j, x_i)$, орієнтовані від вершини $x_j$ до вершини $x_i$, яка має менший номер ($j>i$). Для успішного знаходження найкоротшого шляху за допомогою методу динамічного програмування до такого графу спочатку застосовується алгоритм топологічного сортування вершин.

Алгоритм топологічного сортування вершин дуже простий. Він дозволяє не тільки правильно перенумерувати вершини графа, але й визначити його ациклічність.

*Крок* 1. Нехай $i=n$, де $n$ – число вершин графа $G$.

*Крок* 2. У графі визначається вершина $x_k$, для якої виконується умова $|Г(x_k)|=\varnothing$ (тобто, вершина, з якої не виходить жодна дуга). Вершина $x_k$ отримує порядковий номер $i$ (перенумеровується) і виключається з подальшого розгляду разом з усіма вхідними в неї інцидентними дугами. $i=i-1$.

*Крок* 3. Повторювати п.2. доти, поки не буде виконано одну з умов: 1) $i=1$ – досягнута початкова вершина. Вершини графа отримали правильну нумерацію.

2) Неможливо визначити вершину, для якої виконувалася б умова $|Г(x_k)|=\varnothing$.. У графі є цикл.

В останньому випадку алгоритм динамічного програмування непридатний. Для пошуку найкоротших шляхів на такому графі необхідно використовувати більш ефективні методи, наприклад, алгоритм Дейкстри.

## Код програми:

```
unit Unit2;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, ExtCtrls,
  StdCtrls, Grids;
Const    pp=50;

type

    Graph=Array[1..PP,1..PP]  of integer;
    TNodeCoord=record
    X:Integer;
    Y:Integer;
    end;
  { TOperForm }

  TOperForm = class(TForm)
    ResultGrid:  TStringGrid;
    ResultButton:  TButton;
    ClearButton:  TButton;
    LoadButton:  TButton;
    ExitButton:  TButton;
    GraphImage:  TImage;
    InfoPanel:  TPanel;
    IMatrixGrid:TStringGrid;
    procedure ClearButtonClick(Sender:  TObject);
    procedure LoadButtonClick(Sender:  TObject);
    procedure ExitButtonClick(Sender:  TObject);
    procedure FormActivate(Sender:  TObject);
    procedure FormCreate(Sender:  TObject);
    procedure NodesPaint;
    procedure EdgesPaint;
    procedure ClearGrid;
    procedure LoadFromFile(M:Integer);
    procedure ResultButtonClick(Sender:  TObject);
    procedure Floyd;
  private
    { private declarations  }
  public
    { public declarations  }
  end;

var
  OperForm: TOperForm;
  CoordArr: array [1..20] of TNodeCoord;
  i,j,p:integer;
  t:Graph;
```

```
implementation
Uses Unit5;
{$R *.lfm}

{ TOperForm }

procedure TOperForm.ExitButtonClick(Sender: TObject);
begin
  Close;
end;

procedure TOperForm.LoadButtonClick(Sender: TObject);
begin
 LoadFromFile(NumVariant); //Загрузили данные в таблицк IMatrixGrid
 EdgesPaint; // Нарисовали ребра
 NodesPaint; // Нарисовали вершины
end;

procedure TOperForm.ClearButtonClick(Sender: TObject);
begin
 ClearGrid;
end;

procedure TOperForm.FormActivate(Sender: TObject);
var i,j:Integer;
begin
 NumNodes:=10+NZK mod 11; //Задаем количество вершин графа
 IMatrixGrid.ColCount:=NumNodes+1;  //Фрмируем колич. колонок матрицы смежности
 IMatrixGrid.RowCount:=NumNodes+1; //Фрмируем колич. строк матрицы смежности
 For i:=0 to IMatrixGrid.ColCount-1 do
  For j:=0 to IMatrixGrid.RowCount-1 do
 begin
  If j=0 then IMatrixGrid.Cells[i,j]:=IntToStr(i) else //Нумерация строк и столбцов
  If i=0 then IMatrixGrid.Cells[i,j]:=IntToStr(j) else //Нумерация строк и столбцов
  IMatrixGrid.Cells[i,j]:='0'; //Заполняем нулями
 end;
 GraphImage.Canvas.Pen.Width:= 1;  // Толщина окаймляющей линии
 GraphImage.Canvas.Pen.Color:=clBlack; // Цвет окаймляющей линии
 GraphImage.Canvas.Brush.Color := clwhite; // Цвет фона
 GraphImage.Canvas.Rectangle(0, 0, 600, 600);// Размер картинки
 GraphImage.Canvas.Font.Size:=12; // Размер шрифиа на картинке
 NodesPaint;
end;

procedure TOperForm.FormCreate(Sender: TObject);
begin

end;

procedure TOperForm.NodesPaint; //Отрисовка вершин
  var i:Integer;
      x,y: integer;     // координаты номеров вершин
      x0,y0: integer;   // центр графа
      a: Extended;      // угол между OX и прямой (x0,yo) (x,y)
```

```
    h: integer;       // номера вершины
begin
 X0:=300;  //Координаты центра графа
 Y0:=300;
 a:=0; // вершины ставим начиная с 1, против часовой стрелки
 h:=1;  // номер вершины
 GraphImage.Canvas.Pen.Width:=  2; //Установка толщины пера
 GraphImage.Canvas.Pen.Color:=clBlack;  //Установка цвета пера
  GraphImage.Canvas.Font.Size:=12;   // Размер шрифта для номеров вершин
 While  a<360 do
 begin
  CoordArr[h].X:=x0+Round(  220 * cos(a*2*pi/360));  //массив координат
  CoordArr[h].Y:=x0-Round(  220 * sin(a*2*pi/360));   //Формируем массив координат вершин
  GraphImage.Canvas.MoveTo(CoordArr[h].X,CoordArr[h].Y);  //Стали в центр вершины
  {Рисуем кружок вершины}
  GraphImage.Canvas.Ellipse(CoordArr[h].X-4,CoordArr[h].Y-
4,CoordArr[h].X+4,CoordArr[h].Y+4);
  // цифры по большему радиусу
  x:=x0+Round(  (220+25) * cos(a*2*pi/360));
  y:=y0-Round(  (220+25) * sin(a*2*pi/360));
  GraphImage.Canvas.TextOut(x-2,y-2,IntToStr(h));  // Рисуем номера вершин
  Inc(h);
  a:=a+360/NumNodes;  // Угловой шаг
 end;
end;

procedure TOperForm.EdgesPaint;  // отрисовка ребер
var i,j:Integer;
    x1,y1:Integer;       // Служебные координаты
    Angle:Extended;      // Угол наклона вершины относительно центра графа
    Weight:Integer;     //Вес ребра
    WL1,WL2:TNodeCoord;
    Len:Extended;     //Длина ребра
    QX,QY:Integer;
begin
 GraphImage.Canvas.Font.Size:=8;  //Устанавливаем размер шрифта для веса ребра
 GraphImage.Canvas.Pen.Color:=clBlack;  //Устанавливаем цвет пера
 For i:=1  to NumNodes do
  For j:=1 to NumNodes do
  begin
   Weight:=StrToInt(IMatrixGrid.Cells[i,j]);
   If Weight>0  then
   begin
    If i<>j then
    begin   // Отрисовка ребер, соединяющих различные вершины
     GraphImage.Canvas.MoveTo(CoordArr[i].X,CoordArr[i].Y);  //Стали в начало ребра
     GraphImage.Canvas.LineTo(CoordArr[j].X,CoordArr[j].Y);  //Нарисовали ребро
     GraphImage.Canvas.Pen.Width:=  4;   // Толщина линии стрелки
     QX:=(CoordArr[j].X-CoordArr[i].X)*(CoordArr[j].X-CoordArr[i].X);
     QY:=(CoordArr[j].Y-CoordArr[i].Y)*(CoordArr[j].Y-CoordArr[i].Y);
     Len:=sqrt(QX+QY);   // Вычислили длину ребра по теореме Пифагора
     x1:=Round((CoordArr[j].X-CoordArr[i].X)*20/Len)+CoordArr[i].X;  //Координаты начла
стрелки
```

```
    y1:=Round((CoordArr[j].Y-CoordArr[i].Y)*20/Len)+CoordArr[i].Y;//Координаты начла стре-
лки
    GraphImage.Canvas.MoveTo(x1,y1); //Стали в начало стрелки
    GraphImage.Canvas.LineTo(CoordArr[i].X,CoordArr[i].Y); // Нарисовали стрелку
    GraphImage.Canvas.Pen.Width:= 2; // Восстановили базовую толщину линии
    WL1.X:=(CoordArr[j].X+CoordArr[i].X) div 2;
    WL1.Y:=(CoordArr[j].Y+CoordArr[i].Y) div 2;
    WL2.X:=(CoordArr[j].X+WL1.X) div 2; // Вычислли координаты для веса ребра
    WL2.Y:=(CoordArr[j].Y+WL1.Y) div 2;
    GraphImage.Canvas.TextOut(WL2.X,WL2.Y,IMatrixGrid.Cells[i,j]); //Нарисовали вес
   end else
   begin // Отрисовка петель
    Angle:=(j-1)*360/NumNodes; //Ввычислили текущий угол наклона вершины
    x1:=CoordArr[i].X+Round( 30 * cos(Angle*2*pi/360)); // Координаты центра петли
    y1:=CoordArr[i].Y-Round( 30 * sin(Angle*2*pi/360));
    GraphImage.Canvas.Ellipse(x1-30,y1-30,x1+30,y1+30); //Нарисовали петлю
    WL2.X:=300+Round( 280 * cos(Angle*2*pi/360)); //Координаты для веса петли
    WL2.Y:=300-Round( 280 * sin(Angle*2*pi/360));
    GraphImage.Canvas.TextOut(WL2.X,WL2.Y,IMatrixGrid.Cells[i,j]); //Нарисовали вес петли
   end;
  end;
 end;
end;
procedure TOperForm.LoadFromFile(M:Integer); //Процедура чтения из файла
var F : Text;
   LogFileName,Str:String;
   i,j:Integer;
begin
 ClearGrid;
 LogFileName:=DataPath+'DATA\P'+IntToStr(M)+'.TXT';
 AssignFile(F,LogFileName);
 {$I-} Reset(F); {$I+}
 if IOResult <> 0 then
 begin
  InfoPanel.Caption:='Невозможно прочитать из '+LogFileName;
  Exit;
 end;
 IMatrixGrid.RowCount:=17;
 IMatrixGrid.ColCount:=17;
 For j:=1 to IMatrixGrid.RowCount-1 do
 For i:=1 to IMatrixGrid.ColCount-1 do
 begin
  Readln(F,str);
 IMatrixGrid.Cells[i,j]:=str;
 T[j,i]:=Strtoint(str);
 end;
 CloseFile(F);
end;

procedure TOperForm.ClearGrid; // Очистка всего
var i,j:Integer;
   x1,y1:Integer;
   Angle:Extended;
begin
```

```
 For j:=1 to IMatrixGrid.RowCount-1 do
 For i:=1 to IMatrixGrid.ColCount-1 do IMatrixGrid.Cells[i,j]:='0';
 GraphImage.Canvas.FillRect(Rect(1,1,599,599));
 NodesPaint;
end;

procedure TOperForm.ResultButtonClick(Sender: TObject);
begin
 Floyd;
end;

procedure TOperForm.Floyd;
var i, j, k, max : integer;
     f:text;
     LogFileName:string;
begin
max:=16;
   for k := 1 to max do
   for i := 1 to max do
   for j := 1 to max do
      begin
       if (t[i,j]<>0)and(t[i,k]<>0)and(t[k,j]<>0)then
       begin
           if t[i,j]>t[i,k]+t[k,j] then
           begin
            t[i,j]:=0;
           end
           else
           begin
           t[i,k]:=0;
           t[k,j]:=0;
           end;
       end;
      end;
for i:=1 to max do
for j:=1 to max do
ResultGrid.Cells[j,i]:=inttostr(t[i,j]);
end;

end.
```

## **Висновок:**

В даній лабораторній роботі я вивчав графи. За варіантом мого завдання я робив пошук короткого шляху методом топологічного сортування.