Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №3

з дисципліни
«Алгоритми і структури даних»

Виконав

Студент групи ІМ-22
Тимофеєв Даниіл Костянтинович
номер у списку групи: 20

Перевірила:

Молчанова А. А.

Київ 2023

Постановка задачі :

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:

— число вершин n;

— розміщення вершин;

— матриця суміжності A.

Параметри задаються на основі номера групи, представленого десятковими цифрами n1, n2 та номера студента у списку групи — десяткового числа n3, n4.

Число вершин n дорівнює 10 + n3.

Розміщення вершин:

— колом при n4 = 0,1;

— прямокутником (квадратом) при n4 = 2,3;

— трикутником при n4 = 4,5;

— колом з вершиною в центрі при n4 = 6,7;

— прямокутником (квадратом) з вершиною в центрі при n4 = 8,9.

Наприклад, при n4 = 10 розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа рис.4.

Матриця A напрямленого графа за варіантом формується за функціями:

```
srand(n1 n2 n3 n4);
T = randm(n,n);
A = mulmr((1.0 - n3*0.02 - n4*0.005 - 0.25),T);
```

де randm(n,n) – розроблена функція, яка формує матрицю розміром n*n, що складається з випадкових чисел у діапазоні (0, 2.0);

mulmr() — розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1 (0, якщо результат менший за 1.0 і 1 — якщо більший за 1.0).

2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

Завдання для конкретного варіанту :

Варіант : 2220.

Число вершин n = 10 + 2 = 12.

Розміщення вершин : колом, бо n4 = 0.

Srand (2220);

Текст програми :

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <math.h>
#define vertices 12
#define IDC_BUTTON 1
#define IDC_BUTTON2 2

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

char ProgName[] = "Lab 3";

struct coordinates {
    double nx[vertices];
    double ny[vertices];
    double loopX[vertices];
    double loopY[vertices];
};


double **randm(int n) {
  srand(2220);
  double **matrix = (double **) malloc(sizeof(double *) * n);
  for (int i = 0; i < n; i++) {
    matrix[i] = (double *) malloc(sizeof(double) * n);
  }
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      matrix[i][j] = (double) (rand() * 2.0) / (double) RAND_MAX;
    }
  }
  return matrix;

}

double **mulmr(double coef, double **matrix, int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      matrix[i][j] *= coef;
      matrix[i][j] = matrix[i][j] < 1 ? 0 : 1;
    }
  }
  return matrix;
}


double **symmetricMatrix(double **matrix, int n) {
```

```
  double **symmetrical = (double **) malloc(n * sizeof(double *));
  for (int i = 0; i < n; ++i) {
    symmetrical[i] = (double *) malloc(n * sizeof(double));
  }
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      symmetrical[i][j] = matrix[i][j];
    }
  }
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (symmetrical[i][j] != symmetrical[j][i]) {
        symmetrical[i][j] = 1;
        symmetrical[j][i] = 1;
      }
    }
  }
  return symmetrical;
}

void freeMatrix(double **matrix, int n) {
  for (int i = 0; i < n; ++i) {
    free(matrix[i]);
  }
  free(matrix);
}

void printMatrix(double **matrix, int n, int initialX, int initialY, HDC hdc)
{
  for (int i = 0, y = initialY + 30; i < n; i++, y += 15) {
    for (int j = 0, x = initialX; j < n; j++, x += 13) {
      wchar_t buffer[2];
      swprintf(buffer, 2, L"%lf", matrix[i][j]);
      TextOut(hdc, x, y, (LPCSTR) buffer, 1);
    }
    MoveToEx(hdc, initialX, y, NULL);
  }
}

void arrow(double fi, double px, double py, HDC hdc) {
  double lx, ly, rx, ry;
  lx = px + 15 * cos(fi + 0.3);
  rx = px + 15 * cos(fi - 0.3);
  ly = py + 15 * sin(fi + 0.3);
  ry = py + 15 * sin(fi - 0.3);
  MoveToEx(hdc, lx, ly, NULL);
  LineTo(hdc, px, py);
  LineTo(hdc, rx, ry);
}

void depictArch(int startX, int startY, int finalX, int finalY, int
archInterval, HDC hdc) {
  XFORM transformMatrix;
  XFORM initialMatrix;
  GetWorldTransform(hdc, &initialMatrix);

  double angle = atan2(finalY - startY, finalX - startX) - M_PI / 2.0;
  transformMatrix.eM11 = (FLOAT) cos(angle);
  transformMatrix.eM12 = (FLOAT) sin(angle);
  transformMatrix.eM21 = (FLOAT) (-sin(angle));
  transformMatrix.eM22 = (FLOAT) cos(angle);
  transformMatrix.eDx = (FLOAT) startX;
  transformMatrix.eDy = (FLOAT) startY;
  SetWorldTransform(hdc, &transformMatrix);
```

```
  const double archWidthRatio = 0.75;
  double archLength = sqrt((finalX - startX) * (finalX - startX) + (finalY -
startY) * (finalY - startY));
  double radiusOfVertex = 15.0;
  double semiMinorAxis = archWidthRatio * archLength;
  double semiMajorAxis = archLength / 2;

  double ellipseStartY = semiMajorAxis;

  double vertexAreaSquared = semiMajorAxis * semiMajorAxis * radiusOfVertex *
radiusOfVertex;
  double semiAxesSquared = semiMinorAxis * semiMinorAxis * semiMajorAxis *
semiMajorAxis;
  double distanceFromCenter = semiMinorAxis * semiMinorAxis * ellipseStartY *
ellipseStartY;
  double distanceFromVertex = semiMinorAxis * semiMinorAxis * radiusOfVertex
* radiusOfVertex;
  double semiMinorAxisPow = pow(semiMinorAxis, 4);

  double intersection = semiMajorAxis *
                        sqrt(vertexAreaSquared - semiAxesSquared +
distanceFromCenter - distanceFromVertex +
                             semiMinorAxisPow);
  double semiMinorAxisSquaredEllipseStartY = semiMinorAxis * semiMinorAxis *
ellipseStartY;
  double denominator = -semiMajorAxis * semiMajorAxis + semiMinorAxis *
semiMinorAxis;


  double contactYRightTop = (semiMinorAxisSquaredEllipseStartY -
intersection) / denominator;
  double contactXRightTop = sqrt(radiusOfVertex * radiusOfVertex -
contactYRightTop * contactYRightTop);
  double contactYBottom = archLength - contactYRightTop;
  double contactXLeftBottom = -contactXRightTop;

  if (archInterval <= vertices / 2) {
    Arc(hdc, -archWidthRatio * archLength, archLength, archWidthRatio *
archLength, 0, 0, 0, 0, archLength);
    double angleOfArrow = -atan2(archLength - contactYBottom,
contactXLeftBottom) + 0.3 / 3;
    arrow(angleOfArrow, contactXLeftBottom, contactYBottom, hdc);
  } else {
    Arc(hdc, -archWidthRatio * archLength, archLength, archWidthRatio *
archLength, 0, 0, archLength, 0, 0);
    double angleOfArrow = -atan2(archLength - contactYBottom, -
contactXLeftBottom) - 0.3 / 3;
    arrow(angleOfArrow, -contactXLeftBottom, contactYBottom, hdc);
  }

  SetWorldTransform(hdc, &initialMatrix);

}

void depictDirectedGraph(int centerX, int centerY, int radiusOfGraph, int
radiusOfVertex, int radiusOfLoop, double angle,
                    struct coordinates coordinates, double **matrix,
                    HPEN KPen, HPEN GPen, HDC hdc) {
  for (int i = 0; i < vertices; i++) {
    for (int j = 0; j < vertices; j++) {
      MoveToEx(hdc, coordinates.nx[i], coordinates.ny[i], NULL);
      if ((j >= i && matrix[i][j] == 1) || (j <= i && matrix[i][j] == 1 &&
matrix[j][i] == 0)) {
```

```
        if (i == j) {
          SelectObject(hdc, GPen);

          Ellipse(hdc, coordinates.loopX[i] - radiusOfLoop,
coordinates.loopY[i] - radiusOfLoop,
                  coordinates.loopX[i] + radiusOfLoop, coordinates.loopY[i] +
radiusOfLoop);


          double radiusOfContact = radiusOfGraph + radiusOfLoop / 2.;
          double triangleHeight = sqrt(3) * radiusOfVertex / 2.;
          double loopAngle = atan2(triangleHeight, radiusOfContact);
          double contactDistance = sqrt(radiusOfContact * radiusOfContact +
triangleHeight * triangleHeight);
          double angleToContactVertex = atan2(coordinates.ny[i] - centerY,
coordinates.nx[i] - centerX);

          double contactPointX = centerX + contactDistance *
cos(angleToContactVertex + loopAngle);
          double contactPointY = centerY + contactDistance *
sin(angleToContactVertex + loopAngle);

          double curvatureAngle = angleToContactVertex + 0.3 / 2.;
          arrow(curvatureAngle, contactPointX, contactPointY, hdc);
          SelectObject(hdc, KPen);
        } else {
          LineTo(hdc, coordinates.nx[j], coordinates.ny[j]);
          double line_angle = atan2(coordinates.ny[i] - coordinates.ny[j],
coordinates.nx[i] - coordinates.nx[j]);
          arrow(line_angle, coordinates.nx[j] + radiusOfVertex *
cos(line_angle),
                coordinates.ny[j] + radiusOfVertex * sin(line_angle), hdc);
        }
      } else if (j < i && matrix[i][j] == 1 && matrix[j][i] == 1) {
        depictArch(coordinates.nx[i], coordinates.ny[i], coordinates.nx[j],
coordinates.ny[j], fabs(i - j), hdc);
      }

    }
  }
}

void depictUndirectedGraph(int centerX, int centerY, int radiusOfGraph, int
radiusOfVertex, int radiusOfLoop, double angle,
                      struct coordinates coordinates, double **matrix,
                      HPEN KPen, HPEN GPen, HDC hdc) {
  for (int i = 0; i < vertices; ++i) {

    for (int j = 0; j < vertices; ++j) {
      MoveToEx(hdc, coordinates.nx[i], coordinates.ny[i], NULL);

      if (matrix[i][j] == 1) {

        if (i == j) {
          SelectObject(hdc, GPen);
          Ellipse(hdc, coordinates.loopX[i] - radiusOfLoop,
coordinates.loopY[i] - radiusOfLoop,
                  coordinates.loopX[i] + radiusOfLoop, coordinates.loopY[i] +
radiusOfLoop);
          SelectObject(hdc, KPen);
        } else {
          LineTo(hdc, coordinates.nx[j], coordinates.ny[j]);
        }
```

```
      }
    }
  }

}


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpszCmdLine, int nCmdShow) {
  WNDCLASS w;
  w.lpszClassName = ProgName;
  w.hInstance = hInstance;
  w.lpfnWndProc = WndProc;
  w.hCursor = LoadCursor(NULL, IDC_ARROW);
  w.hIcon = 0;
  w.lpszMenuName = 0;
  w.hbrBackground = WHITE_BRUSH;
  w.style = CS_HREDRAW | CS_VREDRAW;
  w.cbClsExtra = 0;
  w.cbWndExtra = 0;
  if (!RegisterClass(&w)) {
    return 0;
  }
  HWND hWnd;
  MSG lpMsg;
  hWnd = CreateWindow(ProgName,
                      (LPCSTR) "Lab 3.  by Daniil Timofeev IM-22",
                      WS_OVERLAPPEDWINDOW,
                      100,
                      100,
                      1000,
                      700,
                      (HWND) NULL,
                      (HMENU) NULL,
                      (HINSTANCE) hInstance,
                      (HINSTANCE) NULL);
  ShowWindow(hWnd, nCmdShow);
  while (GetMessage(&lpMsg, hWnd, 0, 0)) {
    TranslateMessage(&lpMsg);
    DispatchMessage(&lpMsg);
  }
  return (lpMsg.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT messg, WPARAM wParam, LPARAM lParam)
{
  HDC hdc;
  PAINTSTRUCT ps;
  HWND Button_directed;
  HWND Button_undirected;
  int state = 0;
  switch (messg) {
    case WM_CREATE: {
      Button_directed = CreateWindow(
              (LPCSTR) "BUTTON",
              (LPCSTR) "Switch to Directed",
              WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
              700,
              30,
              160,
              50,
              hWnd,
              (HMENU) IDC_BUTTON,
              (HINSTANCE) GetWindowLongPtr(hWnd, GWLP_HINSTANCE),
```

```
                NULL);
      Button_undirected = CreateWindow(
              (LPCSTR) "BUTTON",
              (LPCSTR) "Switch to Undirected",
              WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
              700,
              600,
              160,
              50,
              hWnd,
              (HMENU) IDC_BUTTON2,
              (HINSTANCE) GetWindowLongPtr(hWnd, GWLP_HINSTANCE),
              NULL);
      return 0;
    }
    case WM_COMMAND: {
      switch (LOWORD(wParam)) {

        case IDC_BUTTON:
          state = 0;
          InvalidateRect(hWnd, NULL, FALSE);
          break;

        case IDC_BUTTON2:
          state = 1;
          InvalidateRect(hWnd, NULL, FALSE);
          break;
      }
    }
    case WM_PAINT :
      hdc = BeginPaint(hWnd, &ps);
      SetGraphicsMode(hdc, GM_ADVANCED);
      HPEN BPen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
      HPEN KPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));
      HPEN GPen = CreatePen(PS_SOLID, 2, RGB(0, 255, 0));
      HPEN NoPen = CreatePen(PS_NULL, 0, RGB(0, 0, 0));
      SelectObject(hdc, NoPen);
      Rectangle(hdc, 0, 0, 670, 700);


      char *nn[vertices] = {"1", "2", "3", "4", "5", "6", "7", "8", "9",
"10\0", "11\0", "12\0"};

      struct coordinates coordinates;

      double circleRadius = 200;
      double vertexRadius = circleRadius / 12;

      double loopRadius = vertexRadius;
      double dtx = vertexRadius / 2.5;

      double circleCenterX = 370;
      double circleCenterY = 360;


      double angleAlpha = 2.0 * M PI / (double) vertices;
      for (int i = 0; i < vertices; i++) {

        double sinAlpha = sin(angleAlpha * (double) i);
        double cosAlpha = cos(angleAlpha * (double) i);
        coordinates.nx[i] = circleCenterX + circleRadius * sinAlpha;
        coordinates.ny[i] = circleCenterY - circleRadius * cosAlpha;
        coordinates.loopX[i] = circleCenterX + (circleRadius + loopRadius) *
sinAlpha;
```

```
        coordinates.loopY[i] = circleCenterY - (circleRadius + loopRadius) *
cosAlpha;

      }


      double **T = randm(vertices);
      double coefficient = 1.0 - 0.02 - 0.005 - 0.25;
      double **A = mulmr(coefficient, T, vertices);

      int initialXOofRandMatrix = 750;
      int initialYofRandMatrix = 150;
      TextOut(hdc, initialXOofRandMatrix, initialYofRandMatrix, (LPCSTR)
L"Initial Matrix", 28);
      printMatrix(A, vertices, initialXOofRandMatrix, initialYofRandMatrix,
hdc);

      double **R = randm(vertices);
      double **C = symmetricMatrix(mulmr(coefficient, R, vertices),
vertices);
      int initialXOofSymmMatrix = initialXOofRandMatrix;
      int initialYofSymmMatrix = initialYofRandMatrix + 210;
      TextOut(hdc, initialXOofSymmMatrix, initialYofSymmMatrix, (LPCSTR)
L"Symmetric Matrix", 31);
      printMatrix(C, vertices, initialXOofSymmMatrix, initialYofSymmMatrix,
hdc);


      SelectObject(hdc, GetStockObject(HOLLOW_BRUSH));
      SelectObject(hdc, KPen);

      if (state == 0) {
        depictDirectedGraph(circleCenterX, circleCenterY, circleRadius,
vertexRadius, loopRadius, angleAlpha,
                            coordinates, A, KPen, GPen, hdc);
      } else {
        depictUndirectedGraph(circleCenterX, circleCenterY, circleRadius,
vertexRadius, loopRadius, angleAlpha,
                              coordinates, C, KPen, GPen, hdc);
      }

      SelectObject(hdc, BPen);
      SelectObject(hdc, GetStockObject(DC_BRUSH));
      SetDCBrushColor(hdc, RGB(204, 204, 255));
      SetBkMode(hdc, TRANSPARENT);

      for (int i = 0; i < vertices; ++i) {
        Ellipse(hdc, coordinates.nx[i] - vertexRadius, coordinates.ny[i] -
vertexRadius,
                coordinates.nx[i] + vertexRadius, coordinates.ny[i] +
vertexRadius);
        TextOut(hdc, coordinates.nx[i] - dtx, coordinates.ny[i] -
vertexRadius / 2, nn[i], 2);
      }


      EndPaint(hWnd, &ps);

      freeMatrix(A, vertices);
      freeMatrix(C, vertices);
    case WM_DESTROY:
      PostQuitMessage(0);
      break;
    default:
```

```
        return (DefWindowProc(hWnd, messg, wParam, lParam));
    }
    return 0;
}
```

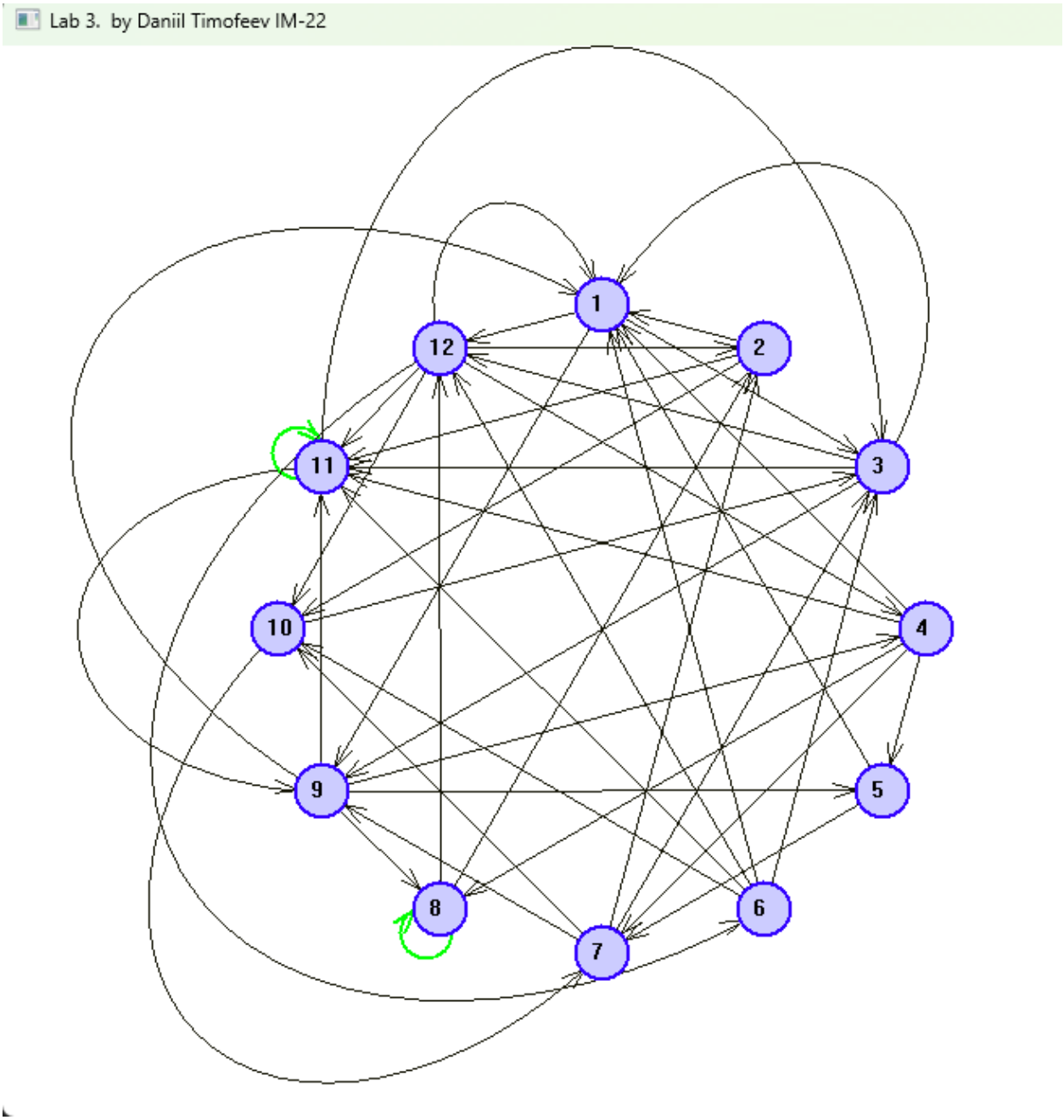
Матриця суміжності напрямленого графа :

```
Initial Matrix

0 0 1 0 0 0 0 0 1 0 0 1
1 0 0 0 0 0 0 0 0 1 1 0
1 0 0 0 0 0 0 0 1 0 1 1
1 0 0 0 1 0 1 1 0 0 1 0
1 0 0 0 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 0 1 1 1
0 1 1 0 0 0 0 0 1 1 0 0
0 1 0 0 0 0 0 1 0 0 0 1
1 0 0 1 1 0 0 1 0 0 1 0
0 0 1 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0 1 0
1 1 0 1 0 1 0 0 0 1 1 0
```

Матриця суміжності ненапрямленого графа :

```
Symmetric Matrix

0 1 1 1 1 1 0 0 1 0 0 1
1 0 0 0 0 0 1 1 0 1 1 1
1 0 0 0 0 1 1 0 1 1 1 1
1 0 0 0 1 0 1 1 1 0 1 1
1 0 0 1 0 0 1 0 1 0 0 0
1 0 1 0 0 0 0 0 0 1 1 1
0 1 1 1 1 0 0 0 1 1 0 0
0 1 0 1 0 0 0 1 1 0 0 1
1 0 1 1 1 0 1 1 0 0 1 0
0 1 1 0 0 1 1 0 0 0 0 1
0 1 1 1 0 1 0 0 1 0 1 1
1 1 1 1 0 1 0 1 0 1 1 0
```

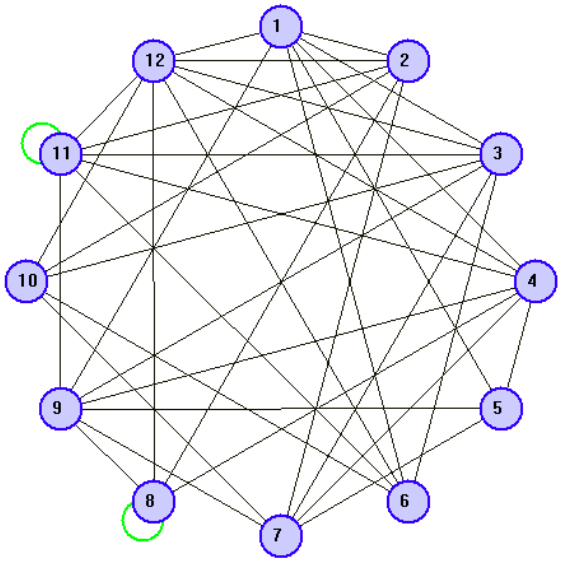
Скриншоти напрямленого графа, який побудований за варіантом :



Скриншоти ненапрямленого графа, який побудований за варіантом :

Lab 3. by Daniil Timofeev IM-22

Switch to Directed



Initial Matrix

0 0 1 0 0 0 0 0 1 0 0 1
1 0 0 0 0 0 0 0 0 1 1 0
1 0 0 0 0 0 0 0 1 0 1 1
1 0 0 0 1 0 1 1 0 0 1 0
1 0 0 0 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 0 1 1 1
0 1 1 0 0 0 0 0 1 1 0 0
0 1 0 0 0 0 0 1 0 0 0 1
1 0 0 1 1 0 0 1 0 0 1 0
0 0 1 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0 1 0
1 1 0 1 0 1 0 0 0 1 1 0

Symmetric Matrix

0 1 1 1 1 1 0 0 1 0 0 1
1 0 0 0 0 0 1 1 0 1 1 1
1 0 0 0 0 1 1 0 1 1 1 1
1 0 0 0 1 0 1 1 1 0 1 1
1 0 0 1 0 0 1 0 1 0 0 0
1 0 1 0 0 0 0 0 0 1 1 1
0 1 1 1 1 0 0 0 1 1 0 0
0 1 0 1 0 0 0 1 1 0 0 1
1 0 1 1 1 0 1 1 0 0 1 0
0 1 1 0 0 1 1 0 0 0 0 1
0 1 1 1 0 1 0 0 1 0 1 1
1 1 1 1 0 1 0 1 0 1 1 0

Switch to Undirected