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-31
Литвиненко Сергій Андрійович
номер у списку групи: 12

Перевірила:
Молчанова А. А.

Київ 2024

Постановка задачі

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

- кількість вершин n;
- розміщення вершин;
- матриця суміжності A.

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

Варіант 12

n1 = 3, n2 = 1, n3 = 1, n4 = 2;

Кількість вершин - 10 + n3 = 11;

Розміщення вершин – квадрат (прямокутник);

Текст програми

Файл config.py:

```
n1, n2, n3, n4 = 3, 1, 1, 2
k = 1 - n3 * 0.02 - n4 * 0.005 - 0.25

VERTICES_COUNT = 10 + n3
WIDTH = 1000
HEIGHT = 800
LINE_WIDTH = 3
LINE_COLOR = 'white'
ARROWS_LENGTH = 15
VERTEX_RADIUS = 50
STEP = VERTEX_RADIUS * 3.5
TITLE = 'Lytvynenko Serhiy, IM-31'

canvas_options = {
  'bg': 'black',
  'borderwidth': 0,
  'highlightthickness': 0,
}

line_options = {
  'fill': LINE_COLOR,
  'width': LINE_WIDTH,
}

text_options = {
  'fill': LINE_COLOR,
  'font': 14,
}
oval_options = {
  'outline': LINE_COLOR,
  'width': LINE_WIDTH,
}
```

Файл main.py:

```
from config import WIDTH, HEIGHT, VERTICES_COUNT, k, TITLE
from matrix import adjacency_matrix, print_matrix, to_indirected
from vertex import vertex_draw, vertex_line, vertex_arc, vertex_loop
from utils import (
  create_window,
  create_vertices,
  is_neighbors,
  in_one_line,
  split_int,
  partial,
  minmax,
)

def draw_graph(canvas, vertices, matrix, ranges, directed=True):
  length = len(vertices)
  one_line = partial(in_one_line, length, ranges)
  neighbors = partial(is_neighbors, length)
  for i in range(length):
    vertex = vertices[i]
    row = matrix[i]
    vertex_draw(canvas, vertex)
    count = length if directed else i + 1
    for j in range(count):
      if row[j] != 1: continue
      other_vertex = vertices[j]
      (m, n) = minmax(i, j)
      if m == n:
        vertex_loop(canvas, vertex, arrows=directed)
      elif not neighbors(m, n) and one_line(m, n):
        vertex_arc(canvas, vertex, other_vertex, arrows=directed)
      else:
        vertex_line(canvas, vertex, other_vertex, arrows=directed)
```

```
def main():
  (root1, canvas1) = create_window(TITLE, WIDTH, HEIGHT)
  (root2, canvas2) = create_window(TITLE, WIDTH, HEIGHT)

  ranges = split_int(VERTICES_COUNT, 4)
  vertices = create_vertices(ranges)

  adjacency = adjacency_matrix(VERTICES_COUNT, k)
  print('Directed graph: ')
  print_matrix(adjacency)
  draw_graph(canvas1, vertices, adjacency, ranges)

  print()

  matrix = to_indirected(adjacency)
  print('Indirected graph: ')
  print_matrix(matrix)
  draw_graph(canvas2, vertices, matrix, ranges, directed=False)

  root1.mainloop()
  root2.mainloop()

if __name__ == '__main__':
  main()
```

Файл matrix.py:

```
import random
import math

float_random = lambda min, max: random.random() * (max - min) + min

def print_matrix(matrix):
  width = 5
  intend = 3
  print(width * ' ', end='')
  length = len(matrix)
  for i in range(length):
    print(f'{i:> {width}}', end=' ')
  print('\n', width * ' ', end='')
  for _ in range(length):
    print('-' * (width + 1), end='')
  for i in range(length):
    print()
    print(f'{i:> {intend}}', end=' |')
    for j in range(length):
        print(f'{matrix[i][j]:> {width}}', end=' ')
  print()


def adjacency_matrix(size, k):
  random.seed(3112)
  matrix = []
  for _ in range(size):
    row = []
    for _ in range(size):
      val = math.floor(float_random(0, 2) * k)
      row.append(val)
    matrix.append(row)
  return matrix
```

```
def to_indirected(matrix):
  length = len(matrix)
  result = []
  for i in range(length):
    result.append([0] * length)
    for j in range(i + 1):
      value = matrix[i][j] | matrix[j][i]
      result[i][j] = value
      result[j][i] = value
  return result
```

Файл utils.py:

```
from config import STEP, VERTEX_RADIUS
from config import canvas_options
from vertex import Vertex
import tkinter as tk
import math

cases = (
  lambda x, y, s: (x + s, y),  # right
  lambda x, y, s: (x, y + s),  # down
  lambda x, y, s: (x - s, y),  # left
  lambda x, y, s: (x, y - s),  # up
)

def split_int(x, ranges):
  high = math.ceil(x / ranges)
  sum = high
  result = []
  while sum <= x:
    result.append(high)
    sum += high
  if sum - high < x:
    result.append(x - sum + high)
  return result

def create_vertices(ranges):
  posx, posy = STEP + VERTEX_RADIUS, STEP
  vertex_index = 0
  vertices = []
  for i in range(len(ranges)):
    cnt = ranges[i]
    next_position = cases[i % len(cases)]
    for _ in range(cnt):
      vertex = Vertex(posx, posy, vertex_index)
```

```
      vertices.append(vertex)
      (posx, posy) = next_position(posx, posy, STEP)
      vertex_index += 1
  return vertices

minmax = lambda x, y: (min(x, y), max(x, y))

partial = lambda fn, *args: lambda *pars: fn(*args, *pars)

is_neighbors = lambda length, x, y: x == y - 1 or y == length - 1 and x ==
0

def in_one_line(length, ranges, x, y):
  last = length - ranges[-1]
  if x == 0 and y >= last and y < length:
    return True
  start = x - x % ranges[0]
  end = start + ranges[0]
  return y >= start and y <= end

def create_window(title, width, height):
  root = tk.Tk()
  root.geometry('%dx%d' % (width, height))
  root.title(title)
  canvas = tk.Canvas(root, canvas_options)
  canvas.pack(fill=tk.BOTH, expand=1)
  return (root, canvas)
```

Файл vertex.py:

```
from math import cos, sin, atan2, sqrt, pi
from dataclasses import dataclass
from config import (
  line_options,
  text_options,
  oval_options,
  ARROWS_LENGTH,
  VERTEX_RADIUS,
)

@dataclass
class Vertex:
  x: int
  y: int
  text: str | int

rotate = lambda x, y, l, f: (
  x + l * cos(f),
  y + l * sin(f),
)

def vertex_arrows(canvas, x, y, fi, delta):
  (lx, ly) = rotate(x, y, ARROWS_LENGTH, fi + delta)
  (rx, ry) = rotate(x, y, ARROWS_LENGTH, fi - delta)
  canvas.create_line(lx, ly, x, y, line_options, width=2)
  canvas.create_line(x, y, rx, ry, line_options, width=2)

def vertex_draw(canvas, vertex):
  x, y = vertex.x, vertex.y
  x1, y1 = x + VERTEX_RADIUS, y + VERTEX_RADIUS
  x2, y2 = x - VERTEX_RADIUS, y - VERTEX_RADIUS
  canvas.create_oval(x1, y1, x2, y2, oval_options)
  canvas.create_text(x, y, text=vertex.text, **text_options)
```

```
def vertex_line(canvas, v1, v2, arrows=False):
  x1, y1 = v1.x, v1.y
  x2, y2 = v2.x, v2.y
  fi = atan2(y2 - y1, x2 - x1)
  (x3, y3) = rotate(x1, y1, VERTEX_RADIUS, fi)
  (x4, y4) = rotate(x2, y2, VERTEX_RADIUS, fi + pi)
  canvas.create_line(x3, y3, x4, y4, line_options)
  if arrows:
    vertex_arrows(canvas, x4, y4, fi + pi, pi / 8)

def vertex_arc(canvas, v1, v2, arrows=False):
  fi = atan2(v2.y - v1.y, v2.x - v1.x)
  (x3, y3) = rotate(v1.x, v1.y, VERTEX_RADIUS, fi) # - pi / 2
  (x4, y4) = rotate(v2.x, v2.y, VERTEX_RADIUS, fi + pi) # + pi / 2
  meadle = sqrt((x4 - x3)**2 + (y4 - y3)**2) / 2
  length = sqrt((3 * VERTEX_RADIUS)**2 + meadle**2)
  f = -atan2(3 * VERTEX_RADIUS, meadle)
  (x5, y5) = rotate(x3, y3, length, f + fi)
  canvas.create_line(x3, y3, x5, y5, x4, y4, line_options, smooth=1)
  if arrows:
    vertex_arrows(canvas, x4, y4, fi + pi - f, pi / 8)

def vertex_loop(canvas, vertex, arrows=False):
  x = vertex.x
  y = vertex.y - VERTEX_RADIUS
  (x1, y1) = rotate(x, y, VERTEX_RADIUS, -pi / 4)
  (x2, y2) = rotate(x, y, VERTEX_RADIUS, -3 * pi / 4)
  canvas.create_line(x, y, x1, y1, x2, y2, x, y, line_options)
  if arrows:
    vertex_arrows(canvas, x, y, pi / 4 + pi, pi / 8)
```

Згенеровані за варіантом матриці суміжності напрямленого і ненапрямленого графів

Матриця суміжності напрямленого графу:

0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1

0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0

0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0

0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0

0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0

1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0

1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0

1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0

1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0

Матриця суміжності ненапрямленого графу:

0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1

0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1

0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1

1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0

0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0

0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0

1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0

1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0

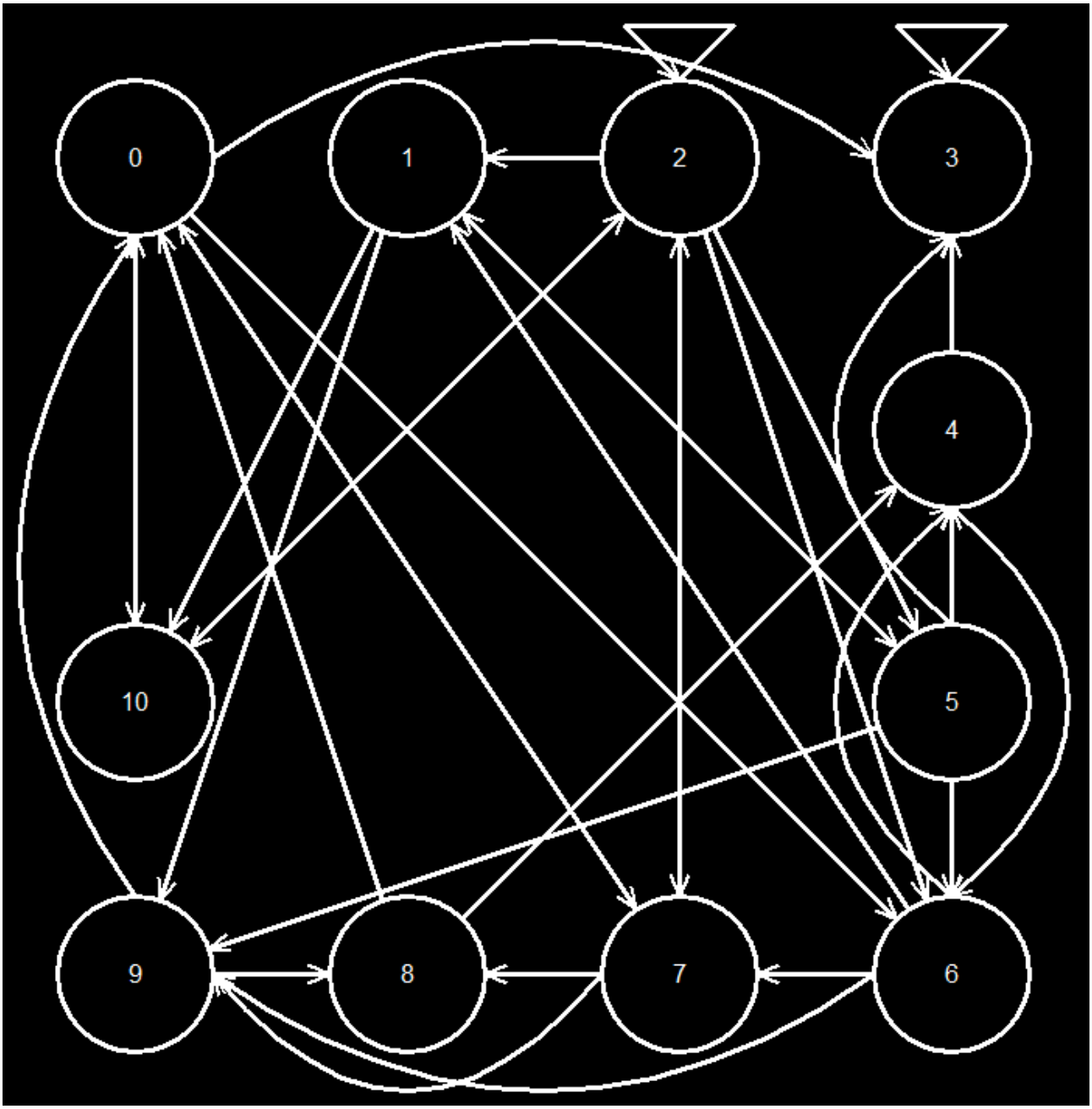
1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0

1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0

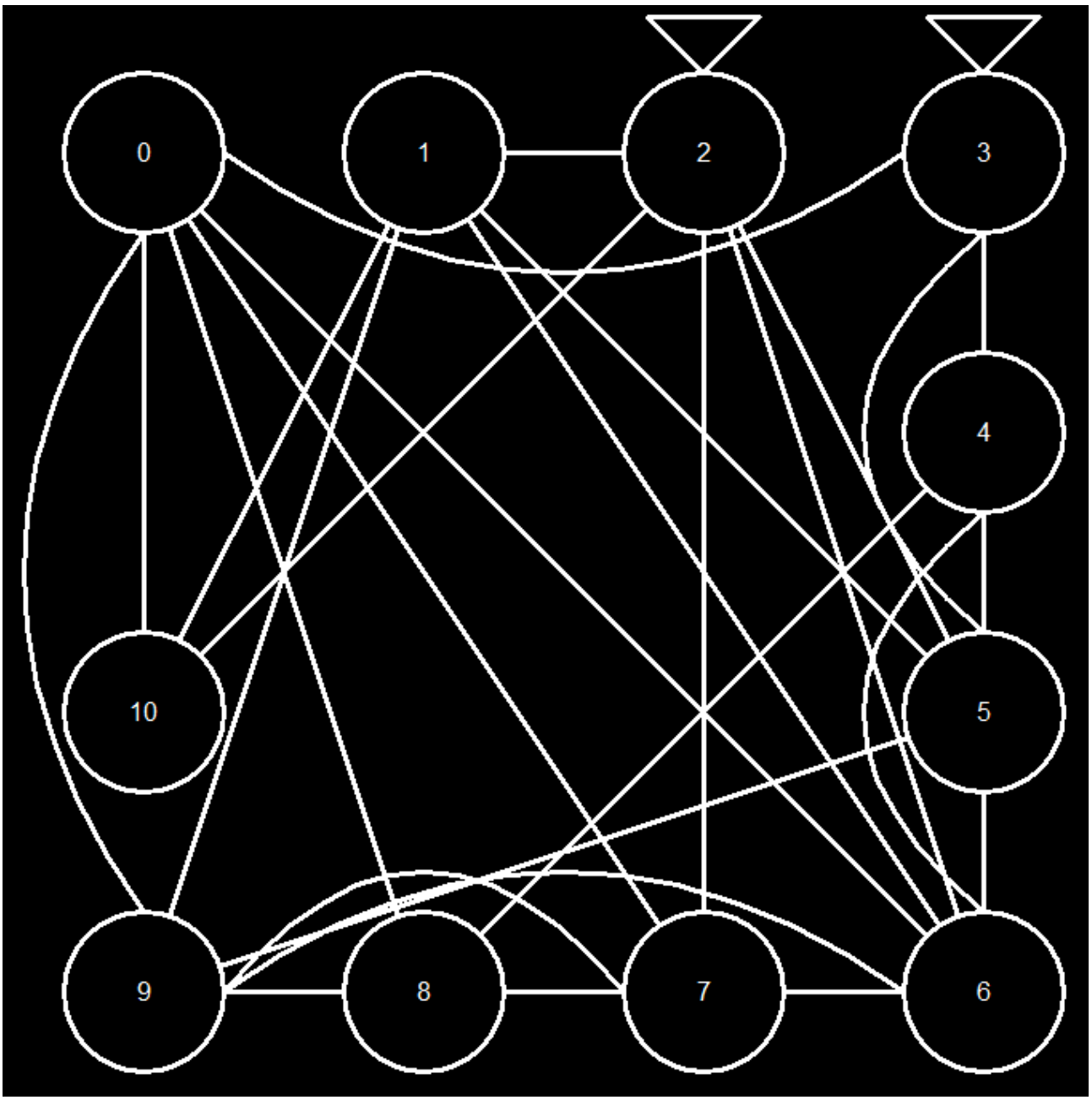
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0

Скришити напрямленого і ненапрямленого графів, які побудовані за варіантом

Напрямлений граф:



Ненапрямлений граф:



Висновки

Протягом даної лабораторної роботи я набув практичних навичок представлення графів у комп'ютері та ознайомився з принципами роботи операційної системи. Я розробив програму на мові програмування Python, яка графічно відображає граф, заданий матрицею суміжності. Ця програма є універсальною, вона видає очікуваний результат для будь-якої матриці суміжності та будь-якої кількості вершин, що поміститься на екрані. За допомогою графів можна наочно відобразити зв'язки між вершинами, що дозволяє розв'язати певний набір задач, наприклад, пошук найкоротшого шляху.

В результаті виконання лабораторної роботи я зрозумів структуру та особливості графів, покращив свої навички в написанні алгоритмів та набув практичних навичок в візуалізації даних.