



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»

КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

«Представление графов и операции над ними»

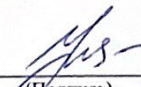
ДИСЦИПЛИНА: «Дискретная математика»

Выполнил: студент гр. ИУК4-32Б


(Подпись)

(Зудин Д.В.)
(Ф.И.О.)

Проверил:


(Подпись)

(Никитенко У.В.)
(Ф.И.О.)

Дата сдачи (защиты): 05.12.2022

Результаты сдачи (защиты):

- Балльная оценка:

75

- Оценка:

зачтено

Калуга, 2022 г.

Цель: изучение основ теории графов, базовых понятий и определений; освоение компьютерных способов представления графов и операций над ними.

Задачи:

Составить программу, которая:

- позволяет задавать пользователю граф в любом из трех видов (список смежности, матрица смежности или матрица инцидентности) и получать на выходе любое другое представление;
- по запросу пользователя выводит характеристики графа (число вершин, число ребер, степенную последовательность, степень выбранной вершины);
- позволяет пользователю выполнять операции (объединение, пересечение, кольцевой суммы, соединение графов, ...);
- визуализация графа – по желанию.

Вариант №4

Формулировка общего задания

Граф G_1 задан матрицей смежности, граф G_2 – матрицей инцидентности, орграф G_3 – матрицей инцидентности. В матрицах инцидентности по строкам располагаются номера вершин. Вывод результатов понимается как вывод их в файл и/или на экран (в отчете также представить результат). Если выполнить операцию невозможно – в отчете необходимо обоснование.

Задание 1. Для заданных графов и орграфа вывести (определить):

- а) число вершин и число ребер (дуг);
- б) списки смежности;
- в) степенные последовательности (для орграфа – полустепени захода и исхода каждой вершины);
- г) матрицу инцидентности G_1 , матрицы смежности G_2 и G_3 .

Задание 2.

- а) Добавить в граф G_1 новые вершины v_1, v_2 и удалить из него вершину v_3 . Вывести результат;
- б) Добавить в полученный на предыдущем шаге граф ребра e_1, e_2, e_3, e_4, e_5 и удалить ребра e_6, e_7 . Вывести результат в виде матрицы инцидентности;
- в) Построить дополнение полученного на шаге б) графа. Пусть это граф G_4 (вывести его в виде матрицы смежности);

- d) Добавить в оргграф G_3 вершины v_4, v_5, v_6 и удалить v_7 . Добавить дуги f_1, f_2, f_3 , удалить дугу f_4 . Вывести результат;
- e) Выполнить операции предыдущего пункта в обратном порядке (сначала дуги, потом вершины). Вывести результат и сравнить с предыдущим.

Формулировка индивидуального задания

Вариант 4.

$$A(G_1) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, I(G_2) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$I(G_3) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Списки вершин, ребер и дуг для заданий:

v_i : {11, 12, 8, 8, 9, 10, 5}; e_i : {(10,11), (2,11), (1,12), (7, 12), (10,12), (2,5), (4,10)}; f_i : {(1,4), (3,4), (3,5), (1,7)}.

Результат выполнения программы

```
for G1
vertices count
10
edge count
10
adjacency lists
[3]
[4]
[6, 7]
[0, 9]
[1, 5]
[4, 7, 8]
[2, 7, 8]
```

```

[2, 5, 6]
[5, 6]
[3]
power sequences
(0, 1)
(0, 1)
(0, 2)
(1, 1)
(1, 1)
(1, 2)
(1, 2)
(3, 0)
(2, 0)
(1, 0)
for G2
vertices count
6
edge count
6
adjacency lists
[4, 5]
[2]
[1, 3, 5]
[2]
[0, 4]
[0, 2]
power sequences
(0, 2)
(0, 1)
(1, 2)
(1, 0)
(2, 0)
(2, 0)
for G3
vertices count
7
edge count
7
adjacency lists
[5]
[2, 3, 4, 6]
[1, 3]

```

```

[1, 2]
[1]
[0, 6]
[1, 5]
power sequences
(0, 1)
(0, 4)
(1, 1)
(2, 0)
(1, 0)
(1, 1)
(2, 0)
incidence matrix for G1
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 0, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
adjacency matrix for G2
[0, 0, 0, 0, 1, 1]
[0, 0, 1, 0, 0, 0]
[0, 1, 0, 1, 0, 1]
[0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 2, 0]
[1, 0, 1, 0, 0, 0]
adjacency matrix for G3
[0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 1]
[0, 1, 0, 1, 0, 0, 0]
[0, 1, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 1, 0]
task_2
new G1
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

```

```

[0, 0, 0, 0, 0, 0, 1, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 1, 0]
[0, 0, 1, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
new G1
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
G4
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[1, 1, 0, 0, 0, 0, 1, 0, 0, 1]
new G3
[1, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 1, 1, 0]
[-1, 0, 0, 1, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, -1, 1]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]

```

```

new G3
[1, 0, 0]
[0, 1, 1]
[0, 1, 0]
[0, 0, 1]
[0, 0, 0]
[1, 0, 0]
[0, 0, 0]

```

Листинг программы

```

class Graph:
    def __init__(self):
        self.gr = []

    def from_sm(self, i):
        self.gr = i

    def from_in(self, i):

        n = len(i)
        self.gr = [[0 for _ in range(n)] for _ in range(n)]
        for j in range(len(i[0])):
            ver = []
            ver_id = []
            for k in range(len(i)):
                if i[k][j] != 0:
                    ver.append(i[k][j])
                    ver_id.append(k)

            if len(ver) == 1:
                if ver[0] != 2:
                    raise ValueError("wrong edge nums")
                self.gr[ver_id[0]][ver_id[0]] = 2

            elif len(ver) == 2:
                if ver[0] < ver[1]:
                    ver[0], ver[1] = ver[1], ver[0]
                    ver_id[0], ver_id[1] = ver_id[1], ver_id[0]

            if ver[0] == 1 and ver[1] == -1:
                # print( ver_id[0], ver_id[1], n )
                self.gr[ver_id[0]][ver_id[1]] = 1

```

```

        elif ver[0] == 1 and ver[1] == 1:
            self.gr[ver_id[0]][ver_id[1]] = 1
            self.gr[ver_id[1]][ver_id[0]] = 1
        else:
            raise ValueError("wrong edge nums")

    else:
        raise ValueError("wrong len of ver")

def to_sm(self):
    return self.gr

def to_in(self):

    n = len(self.gr)
    inc = []
    for i in range(n):
        for j in range(n - i - 1):
            if self.gr[i][1 + i + j] != 0 or self.gr[1 + i + j][i] != 0:
                inc.append([0 for _ in range(n)])
                if self.gr[i][1 + i + j] == 0 and self.gr[1 + i + j][i]
== 1:

                    inc[-1][i] = -1
                    inc[-1][1 + i + j] = 1
                    elif self.gr[i][1 + i + j] == 1 and self.gr[1 + i + j][i]
== 0:

                        inc[-1][i] = 1
                        inc[-1][1 + i + j] = -1
                        elif self.gr[i][1 + i + j] == 1 and self.gr[1 + i + j][i]
== 1:

                            inc[-1][i] = 1
                            inc[-1][1 + i + j] = 1

        for i in range(n):
            if self.gr[i][i] == 2:
                inc.append([0 for _ in range(n)])
                inc[-1][i] = 2

    inc_t = [[inc[i][j] for i in range(len(inc))] for j in
range(len(inc[0]))]

    return inc_t

```



```

def ver_count(self):
    return len(self.gr)

def edg_count(self):
    return len(self.to_in()[0])

def sm_list(self):
    lines = []
    for i in range(len(self.gr)):
        l = []
        for j in range(len(self.gr[0])):
            if self.gr[i][j] != 0:
                l.append(j)
        lines.append(l)
    return lines

def sm_list_r(self):
    lines_in = []
    lines_out = []
    for i in range(len(self.gr)):
        li = []
        lo = []
        for j in range(len(self.gr[0])):
            if self.gr[i][j] != 0:
                if i < j:
                    lo.append(j)
                else:
                    li.append(j)
        lines_in.append(li)
        lines_out.append(lo)
    return lines_in, lines_out

def is_ograph(self):
    n = len(self.gr)
    for i in range(n):
        for j in range(n - i - 1):
            if self.gr[i][1 + i + j] != 0 and self.gr[1 + i + j][i] != 0:
                if self.gr[i][1 + i + j] != self.gr[1 + i + j][i]:
                    return False
    return True

```

```

def step_posl(self):
    if self.is_ograph():
        return self.polu_step_posl()
    l = self.sm_list()
    c = [len(i) for i in l]
    return c

def polu_step_posl(self):
    li, lo = self.sm_list_r()
    c1 = [len(i) for i in li]
    c2 = [len(i) for i in lo]
    c = [(c1[i], c2[i]) for i in range(len(c1))]
    return c

def del_ver(self, i):
    i -= 1
    for line in self.gr:
        line.pop(i)
    self.gr.pop(i)

def add_ver(self, i):
    n = len(self.gr)
    i -= 1
    new_l = [0 for _ in range(n + 1)]
    for line in self.gr:
        line.insert(i, 0)
    self.gr.insert(i, new_l)

def del_edg(self, e):
    a, b = e
    a -= 1
    b -= 1
    if self.gr[a][b] == self.gr[b][a] == 1:
        self.gr[a][b] = 0
        self.gr[b][a] = 0
    if self.gr[a][b] == 1:
        self.gr[a][b] = 0

def add_edg(self, e, pos=False):
    a, b = e
    a -= 1
    b -= 1

```

```

        self.gr[a][b] = 1
    if not pos:
        self.gr[b][a] = 1

def dop(self):
    n = len(self.gr)
    g = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n):
        for j in range(n):
            if self.gr[i][j] != 0:
                g[i][j] = 0
            else:
                g[i][j] = 1
                if i == j:
                    g[i][j] = 2

    return g

def out(self):
    print_matr(self.gr)

def print_matr(m):
    for l in m:
        print(l)

def task_1(g):
    print("vertices count")
    print(g.ver_count())
    print("edge count")
    print(g.edg_count())
    print("adjacency lists")
    print_matr(g.sm_list())
    print("power sequences")
    print_matr(g.step_posl())

if __name__ == "__main__":
    G_1 = [[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 1, 1, 0, 0],

```

```

[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0],
[0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], ]
G_2 = [[1, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0],
[0, 0, 1, 1, 0, 1],
[0, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 2, 0],
[1, 0, 0, 0, 0, 1], ]
G_3 = [[1, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 1, 1, 0, 1, 0],
[0, 0, 0, 1, 1, 0, 0],
[0, 0, 0, 0, 1, 1, 0],
[0, 0, 1, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 1],
[0, 1, 0, 0, 0, 0, 1], ]

v_i = [11, 12, 8, 8, 9, 7, 5]
e_i = [(10, 11), (2, 11), (1, 11), (7, 11), (10, 11), (2, 5), (4, 10)]
f_i = [(1, 4), (3, 4), (3, 5), (1, 7)]

G1 = Graph()
G1.from_sm(G_1)
G2 = Graph()
G2.from_in(G_2)
G3 = Graph()
G3.from_in(G_3)

# G2.out()

print("for G1")
task_1(G1)

print("for G2")
task_1(G2)

print("for G3")
task_1(G3)

```

```

print("incidence matrix for G1")
print_matr(G1.to_in())
print("adjacency matrix for G2")
print_matr(G2.to_sm())
print("adjacency matrix for G3")
print_matr(G3.to_sm())

print("task_2")

print("new G1")
print_matr(G1.to_sm())

G1.add_ver(v_i[0])
G1.add_ver(v_i[1])
G1.del_ver(v_i[2])

print("new G1")
print_matr(G1.to_sm())

G1.add_edg(e_i[0])
G1.add_edg(e_i[1])
G1.add_edg(e_i[2])
G1.add_edg(e_i[3])
G1.add_edg(e_i[4])
G1.del_edg(e_i[5])
G1.del_edg(e_i[6])

print("G4")
print_matr(G1.to_sm())

G3.add_ver(v_i[3])
G3.add_ver(v_i[4])
G3.add_ver(v_i[5])
G3.del_ver(v_i[6])

G3.add_edg(f_i[0], True)
G3.add_edg(f_i[1], True)
G3.add_edg(f_i[2], True)
G3.del_edg(f_i[3])

print("new G3")

```

```
print_matr(G3.to_in())

G3.add_edg(f_i[3], True)
G3.del_edg(f_i[2])
G3.del_edg(f_i[1])
G3.del_edg(f_i[0])

G3.add_ver(v_i[6])
G3.del_ver(v_i[3])
G3.del_ver(v_i[4])
G3.del_ver(v_i[5])

print("new G3")
print_matr(G3.to_in())
```

Выводы:

В ходе работы были изучены основы теории графов, базовые понятия и определения; освоены компьютерные способы представления графов и операции над ними.