Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнил:

студент группы 20ВВ2

Барсуков Н.И.

Проверили:

Митрохин М.А.

Юрова О.В.

Пенза 2021

**Задание 1:**

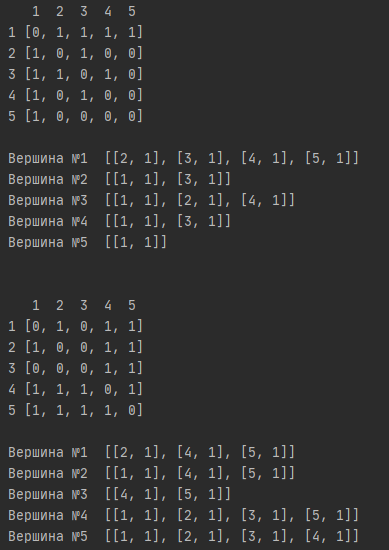
1. Сгенерируйте (используя генератор случайных чисел) две матрицы M 1 ,

М 2 смежности неориентированных помеченных графов G 1 , G 2 . Выведите

сгенерированные матрицы на экран.

2. \* Для указанных графов преобразуйте представление матриц

смежности в списки смежности. Выведите полученные списки на экран.



**Листинг**

import Graph  
  
n = 5  
graphA = []  
line\_data\_A = []  
graphB = []  
line\_data\_B = []  
  
Graph.generator\_matrix(graphA, n)  
Graph.print\_matrix(graphA, 0, n)  
Graph.generator\_line(line\_data\_A, graphA, n)  
print()  
  
Graph.generator\_matrix(graphB, n)  
Graph.print\_matrix(graphB, 0, n)  
Graph.generator\_line(line\_data\_B, graphB, n)

**Листинг функций**

def generator\_matrix(graph, n):  
 for i in range(n):  
 a = []  
 for j in range(n):  
 a.append(0)  
 graph.append(a)  
  
 for i in range(n):  
 for j in range(n):  
 if (graph[i][j] == 0) and (i != j):  
 graph[i][j] = random.randint(0, 1)  
 graph[j][i] = graph[i][j]

def print\_matrix(graph, min, n):  
 a = []  
 for i in range(min, n):  
 a.append(i+1)  
 print(" ", end=' ')  
 for i in range(n - min):  
 print(a[i], end=' ')  
 print("")  
 for i in range(n - min):  
 print("%d" % (a[i]), graph[i])  
 print()

def generator\_line(line\_data, graph, n):  
 for i in range(n):  
 line = []  
 for j in range(n):  
 if (graph[i][j] != 0):  
 a = []  
 a.append(j+1)  
 a.append(graph[i][j])  
 line.append(a)  
 print("Вершина №%d " % (i + 1), line)  
 line\_data.append(line)  
 print()

**Задание 2:**

1. Для матричной формы представления графов выполните операцию:

а) отождествления вершин

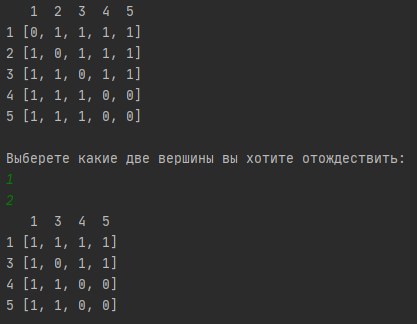
б) стягивания ребра

в) расщепления вершины

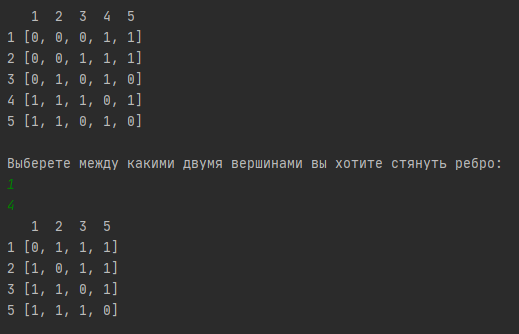
Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

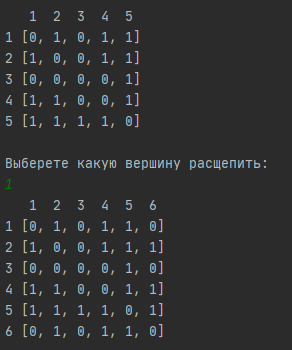
а)



б)



в)



**Листинг**

import Graph  
import Operations  
  
n = 5  
graphA = []  
  
Graph.generator\_matrix(graphA, n)  
Graph.print\_matrix(graphA, 0, n)  
  
  
print("Выберете какие две вершины вы хотите отождествить: ")  
peak1 = int(input())  
peak2 = int(input())  
  
Operations.identification\_peak(graphA, peak1, peak2, n)  
Operations.print\_newGraph(graphA, peak2, n)  
  
  
graphB = []  
  
Graph.generator\_matrix(graphB, n)  
Graph.print\_matrix(graphB, 0, n)  
  
print("Выберете между какими двумя вершинами вы хотите стянуть ребро: ")  
peak1 = int(input())  
peak2 = int(input())  
  
Operations.contraction\_edge(graphB, peak1, peak2, n)  
Operations.print\_newGraph(graphB, peak2, n)  
  
  
graphC = []  
  
Graph.generator\_matrix(graphC, n)  
Graph.print\_matrix(graphC, 0, n)  
  
print("Выберете какую вершину расщепить: ")  
peak = int(input())  
  
Operations.copy\_peak(graphC, peak, n)  
Graph.print\_matrix(graphC, 0, n+1)

**Листинг функций**

def identification\_peak(graph, peak1, peak2, n):  
 for j in range(n):  
 if (graph[peak1-1][j] != graph[peak2-1][j]):  
 if (graph[peak2-1][j] == 0):  
 graph[peak1 - 1][j] += graph[peak2 - 1][j]  
 continue  
 graph[peak1-1][j] = graph[peak2-1][j]  
 for i in range(n):  
 for j in range(n):  
 if (graph[peak1-1][j] != graph[j][peak1-1]):  
 graph[j][peak1 - 1] = graph[peak1-1][j]  
 disconnect\_peak(graph, peak2-1)

def disconnect\_peak(graph, peak):  
 for i in range(len(graph)):  
 del graph[i][peak]  
 del graph[peak]

def print\_newGraph(graph, peak2, n):  
 a = []  
 for i in range(n):  
 if (i != peak2 - 1):  
 a.append(i + 1)  
 print(" ", end=' ')  
 for i in range(n-1):  
 print(a[i], end=' ')  
 print("")  
 j = 0  
 for i in range(n-1):  
 print("%d" % (a[i]), graph[i])  
 print()

def contraction\_edge(graph, peak1, peak2, n):  
 running = True  
 while(running == True):  
 if (graph[peak1-1][peak2-1] == 0):  
 print("Между данными вершинами нет ребра, выберите другие вершины")  
 peak1 = int(input())  
 peak2 = int(input())  
 elif (graph[peak1-1][peak2-1] != 0):  
 for j in range(n):  
 if (graph[peak1 - 1][j] != graph[peak2 - 1][j]):  
 if (graph[peak2 - 1][j] == 0):  
 graph[peak1 - 1][j] += graph[peak2 - 1][j]  
 continue  
 graph[peak1 - 1][j] = graph[peak2 - 1][j]  
  
 for i in range(n):  
 for j in range(n):  
 if (graph[peak1 - 1][j] != graph[j][peak1 - 1]):  
 graph[j][peak1 - 1] = graph[peak1 - 1][j]  
 graph[peak1-1][peak1-1] = 0  
 disconnect\_peak(graph, peak2-1)  
 running = False

def copy\_peak(graph, peak, n):  
 a = []  
 graph.append(a)  
 for i in range(n):  
 a.append(0)  
 a[i] = graph[peak-1][i]  
  
 for i in range(n+1):  
 graph[i].append(graph[peak - 1][i])  
 graph[peak-1][peak-1] = 0  
 graph[n][n] = 0

2. \* Для представления графов в виде списков смежности выполните

операцию:

а) отождествления вершин

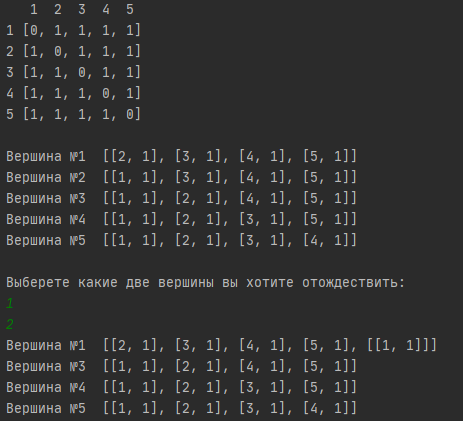
б) стягивания ребра

в) расщепления вершины

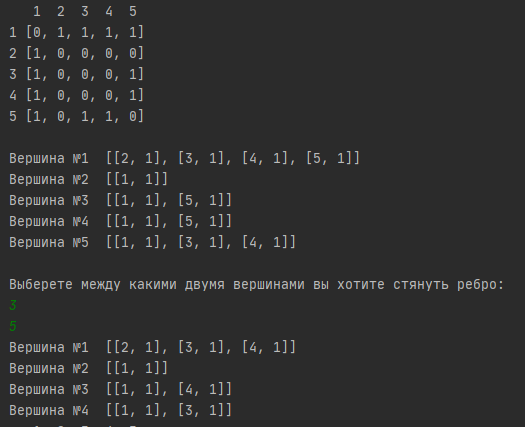
Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

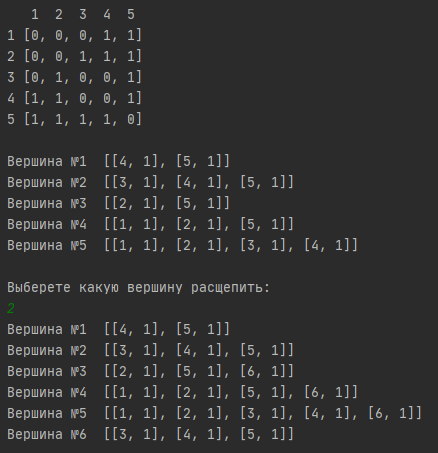
а)



б)



в)



**Листинг**

import Graph  
import Operations  
  
n = 5  
graphA = []  
line\_data = []  
  
Graph.generator\_matrix(graphA, n)  
Graph.print\_matrix(graphA, 0, n)  
Graph.generator\_line(line\_data, graphA, n)  
  
  
print("Выберете какие две вершины вы хотите отождествить: ")  
peak1 = int(input())  
peak2 = int(input())  
  
Operations.identification\_peak\_for\_line(line\_data, peak1, peak2)  
  
for i in range(n):  
 if (i == peak2-1):  
 continue  
 print("Вершина №%d " % (i + 1), (line\_data[i]))  
  
  
graphB = []  
line\_data\_B = []  
  
Graph.generator\_matrix(graphB, n)  
Graph.print\_matrix(graphB, 0, n)  
Graph.generator\_line(line\_data\_B, graphB, n)  
  
print("Выберете между какими двумя вершинами вы хотите стянуть ребро: ")  
peak1 = int(input())  
peak2 = int(input())  
  
Operations.contraction\_edge\_for\_line(line\_data\_B, graphB, peak1, peak2, n)  
  
for i in range(n):  
 if (i == peak2-1):  
 continue  
 print("Вершина №%d " % (i + 1), (line\_data\_B[i]))  
  
  
graphC = []  
line\_data\_C = []  
  
Graph.generator\_matrix(graphC, n)  
Graph.print\_matrix(graphC, 0, n)  
Graph.generator\_line(line\_data\_C, graphC, n)  
  
print("Выберете какую вершину расщепить: ")  
peak = int(input())  
  
Operations.copy\_peak\_for\_line(line\_data\_C, peak, n)  
for i in range(n+1):  
 print("Вершина №%d " % (i + 1), (line\_data\_C[i]))

**Листинг функций**

def identification\_peak\_for\_line(line\_data, peak1, peak2, n):  
 m = len(line\_data[peak2 - 1])  
 for i in range(len(line\_data[peak1-1])):  
 for j in range(m):  
 if (line\_data[peak1-1][i][0] == line\_data[peak2-1][j][0]):  
 del line\_data[peak2-1][j]  
 m -= 1  
 break  
 for i in range(len(line\_data[peak2-1])):  
 line\_data[peak1-1].append(line\_data[peak2-1][i])  
 disconnect\_peak\_for\_line(line\_data, peak2, n)  
 connect\_peak\_for\_line(line\_data, peak1, peak2)

def contraction\_edge\_for\_line(line\_data, graph, peak1, peak2, n):  
 running = True  
 while(running == True):  
 if (graph[peak1-1][peak2-1] == 0):  
 print("Между данными вершинами нет ребра, выберите другие вершины")  
 peak1 = int(input())  
 peak2 = int(input())  
 elif (graph[peak1-1][peak2-1] != 0):  
 m = len(line\_data[peak2 - 1])  
 for i in range(len(line\_data[peak1 - 1])):  
 for j in range(m):  
 if (line\_data[peak1 - 1][i][0] == line\_data[peak2 - 1][j][0]):  
 del line\_data[peak2 - 1][j]  
 m -= 1  
 break  
 for i in range(len(line\_data[peak2-1])):  
 line\_data[peak1 - 1].append(line\_data[peak2 - 1][i])  
 disconnect\_loop\_for\_line(line\_data, peak1)  
 disconnect\_peak\_for\_line(line\_data, peak2, n)  
 connect\_peak\_for\_line(line\_data, peak1, peak2)  
 running = False

def copy\_peak\_for\_line(line\_data, peak, n):  
 a = []  
 line\_data.append(a)  
 for i in range(len(line\_data[peak-1])):  
 a.append(0)  
 a[i] = line\_data[peak - 1][i]  
  
 for i in range(len(line\_data[peak-1])):  
 length = line\_data[peak - 1][i][1]  
 newPeak = [6, length]  
 line\_data[line\_data[peak-1][i][0]-1].append(newPeak)  
 for i in range(len(line\_data[peak-1])):  
 if line\_data[peak-1][i][0] == peak:  
 del line\_data[peak-1][i]  
 if line\_data[n][i][0] == n+1:  
 del line\_data[n][n]

**Задание 3:**

1. Для матричной формы представления графов выполните операцию:

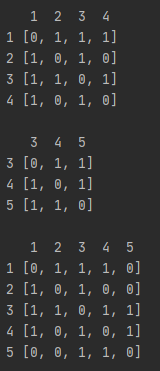
а) объединения

б) пересечения

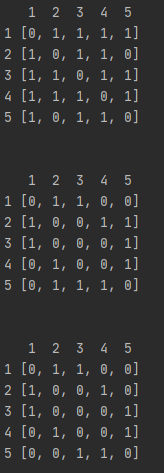
в) кольцевой суммы

Результат выполнения операции выведите на экран.

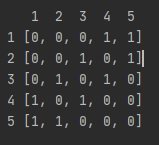
а)



б)



в)



**Листинг**

import Graph  
import Operations  
  
n = 5  
graphA = []  
graphB = []  
  
Graph.generator\_matrix(graphA, n)  
Graph.print\_matrix(graphA, 0, n)  
print()  
  
Graph.generator\_matrix(graphB, n)  
Graph.print\_matrix(graphB, 0, n)  
print()  
  
graphC = []  
for i in range(n):  
 a = []  
 for j in range(n):  
 a.append(0)  
 graphC.append(a)  
  
Operations.crossroads\_graphs(graphA, graphB, graphC, n)  
Graph.print\_matrix(graphC, 0, n)  
  
  
graphD = []  
for i in range(n):  
 a = []  
 for j in range(n):  
 a.append(0)  
 graphD.append(a)  
  
Operations.circle\_sum\_graphs(graphA, graphB, graphD, n)  
Graph.print\_matrix(graphD, 0, len(graphD))  
  
  
graphE = []  
graph1 = []  
graph2 = []  
  
Graph.generator\_matrix(graph1, 4)  
Graph.print\_matrix(graph1, 0, 4)  
  
Graph.generator\_matrix(graph2, 3)  
Graph.print\_matrix(graph2, 2, 5)  
  
Operations.unification\_graphs(graph1, graph2, 2)  
Graph.print\_matrix(graph1, 0, 5)

**Листинг функций**

def crossroads\_graphs(graph1, graph2, graph3, n):  
 for i in range(n):  
 for j in range(n):  
 if graph1[i][j] == 1 and graph2[i][j] == 1:  
 graph3[i][j] = 1  
 zero\_peak = []  
 for i in range(len(graph3)):  
 if sum(graph3[i]) == 0:  
 zero\_peak.append(i)  
  
 for i in range(len(zero\_peak)):  
 disconnect\_peak(graph3, zero\_peak[i])  
 if i != len(zero\_peak) - 1:  
 zero\_peak[i + 1] -= (1 + i)

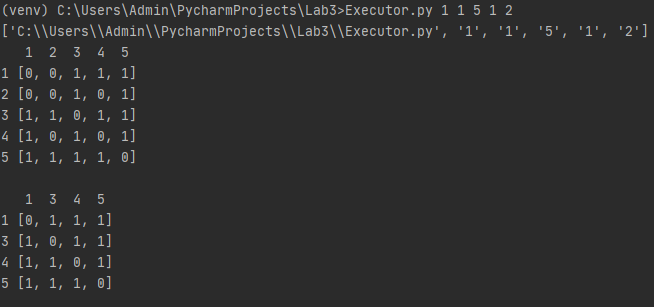
def circle\_sum\_graphs(graph1, graph2, graph3, n):  
 for i in range(len(graph3)):  
 if len(graph1) > len(graph2):  
 graph3[i][len(graph1)-1] = graph1[i][len(graph1)-1]  
 else:  
 graph3[i][len(graph2) - 1] = graph2[i][len(graph2) - 1]  
  
 for i in range(n):  
 for j in range(n):  
 if graph1[i][j] == 1 or graph2[i][j] == 1:  
 graph3[i][j] = 1  
 if graph1[i][j] == 1 and graph2[i][j] == 1:  
 graph3[i][j] = 0

def unification\_graphs(graph1, graph2, same\_peak):  
 n = len(graph1)  
 for i in range(len(graph2) - same\_peak):  
 a = []  
 for j in range(len(graph1)):  
 a.append(0)  
 graph1.append(a)  
 for j in range(len(graph1)):  
 graph1[j].append(0)  
  
 for i in range(len(graph2)):  
 k = 0  
 for j in range((n) - same\_peak, len(graph1)):  
 if graph1[(n) - same\_peak + i][j] == 0:  
 graph1[n - same\_peak + i][j] += graph2[i][k]  
 k += 1

**Файл Executor.py**

Данный файл собирает в себе все функции для того чтобы передавать в них параметры из командной строки.

Пример вызова операции отождествления вершины, на вершинах 1 и 2.



**Листинг файла Executor.py**

import sys  
import Operations  
import Graph  
#первое число номер задания, второе номер операции, последующие аргументы  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(sys.argv)  
 if int(sys.argv[1]) == 1:  
 if int(sys.argv[2]) == 1:  
 graph = []  
 Graph.generator\_matrix(graph, int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
 Operations.identification\_peak(graph, int(sys.argv[4]), int(sys.argv[5]), int(sys.argv[3]))  
 Operations.print\_newGraph(graph, int(sys.argv[5]), int(sys.argv[3]))  
 elif int(sys.argv[2]) == 2:  
 graph = []  
 Graph.generator\_matrix(graph, int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
 Operations.contraction\_edge(graph, int(sys.argv[4]), int(sys.argv[5]), int(sys.argv[3]))  
 Operations.print\_newGraph(graph, int(sys.argv[5]), int(sys.argv[3]))  
 elif int(sys.argv[2]) == 3:  
 graph = []  
 Graph.generator\_matrix(graph, int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
 Operations.copy\_peak(graph, int(sys.argv[4]), int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
  
 elif int(sys.argv[1]) == 2:  
 if int(sys.argv[2]) == 1:  
 graph = []  
 line\_data = []  
 Graph.generator\_matrix(graph, int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
 Graph.generator\_line(line\_data, graph, int(sys.argv[3]))  
 Operations.identification\_peak\_for\_line(line\_data, int(sys.argv[4]), int(sys.argv[5]), int(sys.argv[3]))  
 for i in range(int(sys.argv[3])):  
 if (i == int(sys.argv[5]) - 1):  
 continue  
 print("Вершина №%d " % (i + 1), (line\_data[i]))  
 elif int(sys.argv[2]) == 2:  
 graph = []  
 line\_data = []  
 Graph.generator\_matrix(graph, int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
 Graph.generator\_line(line\_data, graph, int(sys.argv[3]))  
 Operations.contraction\_edge\_for\_line(line\_data, graph, int(sys.argv[4]), int(sys.argv[5]), int(sys.argv[3]))  
 for i in range(int(sys.argv[3])):  
 if (i == int(sys.argv[5]) - 1):  
 continue  
 print("Вершина №%d " % (i + 1), (line\_data[i]))  
  
 elif int(sys.argv[2]) == 3:  
 graph = []  
 line\_data = []  
 Graph.generator\_matrix(graph, int(sys.argv[3]))  
 Graph.print\_matrix(graph, 0, int(sys.argv[3]))  
 Graph.generator\_line(line\_data, graph, int(sys.argv[3]))  
 Operations.copy\_peak\_for\_line(line\_data, int(sys.argv[4]), int(sys.argv[3]))  
 for i in range(int(sys.argv[3]) + 1):  
 print("Вершина №%d " % (i + 1), (line\_data[i]))  
  
 elif int(sys.argv[1]) == 3:  
 if int(sys.argv[2]) == 1:  
 graph = []  
 graph1 = []  
 graph2 = []  
 Graph.generator\_matrix(graph1, int(sys.argv[3]))  
 Graph.print\_matrix(graph1, 0, int(sys.argv[3]))  
  
 Graph.generator\_matrix(graph2, int(sys.argv[4]))  
 Graph.print\_matrix(graph2, int(sys.argv[5]), int(sys.argv[6])) # 5 - мин.; 6 - макс  
  
 Operations.unification\_graphs(graph1, graph2, int(sys.argv[7])) # 7 - колво одинак вершин  
 Graph.print\_matrix(graph1, 0, int(sys.argv[6]))  
 elif int(sys.argv[2]) == 2:  
 graphA = []  
 graphB = []  
  
 Graph.generator\_matrix(graphA, int(sys.argv[3]))  
 Graph.print\_matrix(graphA, 0, int(sys.argv[3]))  
 print()  
  
 Graph.generator\_matrix(graphB, int(sys.argv[4]))  
 Graph.print\_matrix(graphB, 0, int(sys.argv[4]))  
 print()  
  
 graphC = []  
 for i in range(int(sys.argv[3])):  
 a = []  
 for j in range(int(sys.argv[3])):  
 a.append(0)  
 graphC.append(a)  
  
 if int(sys.argv[3]) >= int(sys.argv[4]):  
 sys.argv[3] = int(sys.argv[4])  
 Operations.crossroads\_graphs(graphA, graphB, graphC, int(sys.argv[3]))  
 Graph.print\_matrix(graphC, 0, len(graphC))  
 elif int(sys.argv[2]) == 3:  
 graphA = []  
 graphB = []  
  
 Graph.generator\_matrix(graphA, int(sys.argv[3]))  
 Graph.print\_matrix(graphA, 0, int(sys.argv[3]))  
 print()  
  
 Graph.generator\_matrix(graphB, int(sys.argv[4]))  
 Graph.print\_matrix(graphB, 0, int(sys.argv[4]))  
 print()  
  
 graphD = []  
 for i in range(max(int(sys.argv[3]), int(sys.argv[4]))):  
 a = []  
 for j in range(max(int(sys.argv[3]), int(sys.argv[4]))):  
 a.append(0)  
 graphD.append(a)  
  
 if int(sys.argv[3]) >= int(sys.argv[4]):  
 n = int(sys.argv[4])  
 else:  
 n = int(sys.argv[3])  
  
 Operations.circle\_sum\_graphs(graphA, graphB, graphD, n)  
 Graph.print\_matrix(graphD, 0, len(graphD))