|  |
| --- |
| 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» |
|  |
|  |
| Выполнила студентка группы 19ВВ3:  Ланцов А.С.  Принял:  Митрохин М. А. |
| Пенза  2020 |

### **Задание 1**

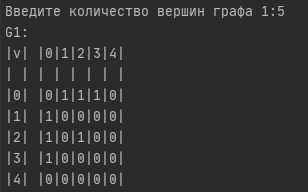
1. Сгенерируйте (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Выведите сгенерированные матрицы на экран.

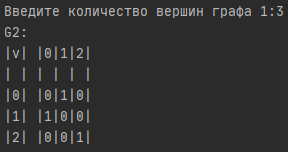
Листинг:

def randomMatrixGraph(n):  
 rng=list(range(n))  
  
 matrix = [[0 for i in range(n)] for i in range(n)]  
  
 for i in rng:  
 for j in rng:  
 matrix[i][j] = random.choice([0, 1])  
 matrix[j][i] = matrix[i][j]  
 del rng[0]  
 return matrix  
  
def printGraph(graph):  
 rng=range(len(graph))  
  
 print('|v| |', end='')  
 for i in rng:  
 print("%d|" %i, end='')  
 print('', end="\n| |")  
  
 for i in rng:  
 print(" |", end='')  
 print(' |', end='\n')  
  
 for i in rng:  
 print("|%d| "% (i), end='')  
 for j in rng:  
 print("|%d"%(graph[i][j]),end='')  
 print('|',end='\n')  
 print('', end='\n')

def task1():  
 size1 = int(input("Введите количество вершин графа 1:"))  
  
 graph1 = randomMatrixGraph(size1)  
  
 print("G1:")  
 printGraph(graph1)  
  
 size2 = int(input("Введите количество вершин графа 1:"))  
  
 graph2 = randomMatrixGraph(size2)  
  
 print("G2:")  
 printGraph(graph2)

Результат работы:





### **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.

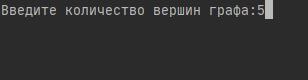
Листинг:

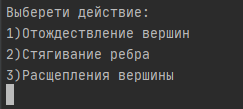
def otozhdestvVer(graph,v,u):  
 gsize=len(graph)  
  
 rng = list(range(gsize))  
  
 #делаем объединения ребер вершин u и v  
 for i in range(gsize):  
 if i!=v:  
 graph[v][i]=graph[v][i] or graph[u][i]  
 graph[i][v] = graph[v][i]  
  
 #удаляем вершину u  
 for i in range(gsize):  
 del graph[i][u]  
  
 #удаляем вершину u  
 del graph[u]  
  
def styanRebro(graph,v,u):  
 #проверяем на смежность  
 if (graph[v][u]==1):  
 otozhdestvVer(graph, v, u)  
 return True  
 else:  
 return False

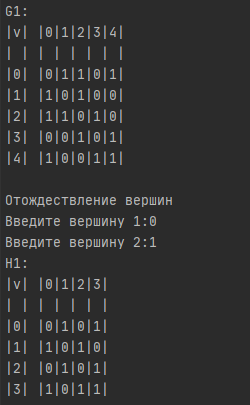
def rasshepVer(graph,v):  
 gsize=len(graph)  
  
 new\_vertx=[graph[v][i] for i in range(gsize)]  
  
 new\_vertx.append(graph[v][v])  
  
 graph.append(new\_vertx)  
  
 for i in range(gsize):  
 graph[i].append(graph[v][i])

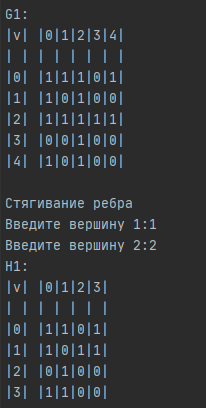
def task2():  
 os.system('cls')  
 size = int(input("Введите количество вершин графа:"))  
 os.system('cls')  
  
 print("Выберети действие:\n"  
 "1)Отождествление вершин\n"  
 "2)Стягивание ребра\n"  
 "3)Расщепления вершины")  
 switch=int(input())  
 os.system('cls')  
  
 graph = randomMatrixGraph(size)  
 print("G1:")  
 printGraph(graph)  
  
 if switch==1:  
 ver1 = int(input("Введите вершину 1:"))  
 ver2 = int(input("Введите вершину 2:"))  
  
 otozhdestvVer(graph, ver1, ver2)  
  
 print("H1:")  
 printGraph(graph)  
  
 elif switch==2:  
 ver1 = int(input("Введите вершину 1:"))  
 ver2 = int(input("Введите вершину 2:"))  
  
 if (styanRebro(graph, ver1, ver2)):  
 print("H1:")  
 printGraph(graph)  
 else:  
 print("Вершины не смежные")  
  
 elif switch==3:  
 ver = int(input("Введите вершину:"))  
  
 rasshepVer(graph,ver)  
  
 print("H1:")  
 printGraph(graph)

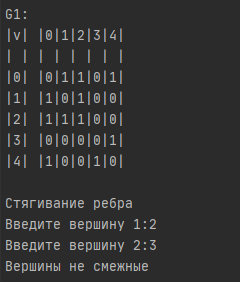
Результат работы:

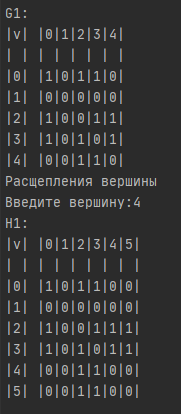












**Задание 3**

1. Для матричной формы представления графов выполните операцию:

а) объединения *G* = *G*1  *G*2

б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

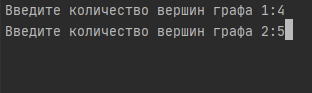
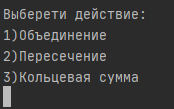
Результат выполнения операции выведите на экран.

Листинг:

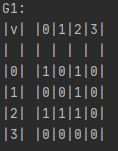
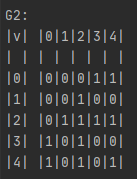
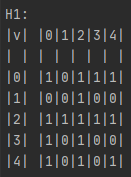
def obedin(graph1,graph2):  
 gsize1 = len(graph1)  
 gsize2 = len(graph2)  
  
 gsize= min(gsize1,gsize2)  
  
 if gsize==gsize1:  
 oldgraph = graph1  
 newgraph = graph2  
 else:  
 oldgraph = graph2  
 newgraph = graph1  
  
 rng=list(range(gsize))  
 for i in rng:  
 for j in rng:  
 newgraph[i][j] = newgraph[i][j] or oldgraph[i][j]  
 newgraph[j][i] = newgraph[i][j]  
 del rng[0]  
  
 return newgraph  
  
def peresechenie(graph1,graph2):  
 gsize1 = len(graph1)  
 gsize2 = len(graph2)  
  
 gsize= min(gsize1,gsize2)  
  
 if gsize==gsize1:  
 oldgraph = graph1  
 newgraph = graph2  
 else:  
 oldgraph = graph2  
 newgraph = graph1  
  
 rng=list(range(gsize))  
 for i in rng:  
 for j in rng:  
 newgraph[i][j] = newgraph[i][j] and oldgraph[i][j]  
 newgraph[j][i] = newgraph[i][j]  
 del rng[0]  
  
 return newgraph  
  
def xorGraphs(graph1,graph2):  
 gsize1 = len(graph1)  
 gsize2 = len(graph2)  
  
 gsize= min(gsize1,gsize2)  
  
 if gsize==gsize1:  
 oldgraph = graph1  
 newgraph = graph2  
 else:  
 oldgraph = graph2  
 newgraph = graph1  
  
 rng=list(range(gsize))  
 for i in rng:  
 for j in rng:  
 newgraph[i][j] = xor(newgraph[i][j],oldgraph[i][j])  
 newgraph[j][i] = newgraph[i][j]  
 del rng[0]  
  
 return newgraph

def task3():  
 os.system('cls')  
  
 size1 = int(input("Введите количество вершин графа 1:"))  
 size2 = int(input("Введите количество вершин графа 2:"))  
 os.system('cls')  
  
 print("Выберети действие:\n"  
 "1)Объединение\n"  
 "2)Пересечение\n"  
 "3)Кольцевая сумма")  
 switch = int(input())  
 os.system('cls')  
  
 graph1 = randomMatrixGraph(size1)  
 graph2 = randomMatrixGraph(size2)  
  
 print("G1:")  
 printGraph(graph1)  
  
 print("G2:")  
 printGraph(graph2)  
  
 if switch == 1:  
 newgraph=obedin(graph1,graph2)  
 print("H1:")  
 printGraph(newgraph)  
  
 elif switch == 2:  
 newgraph = peresechenie(graph1, graph2)  
 print("H1:")  
 printGraph(newgraph)  
  
 elif switch == 3:  
 newgraph = xorGraphs(graph1, graph2)  
 print("H1:")  
 printGraph(newgraph)

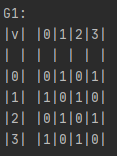
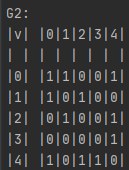
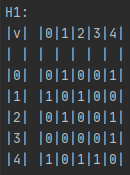
Результат работы:

Объединение:

Пересечение:

Кольцевая сумма:

