Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Определение характеристик графа»

Выполнил:

студент группы 20ВВ2

Барсуков Н.И.

Проверили:

Митрохин М.А.

Юрова О.В.

Пенза 2021

**Задание 1:**

1. Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного взвешенного графа G. Выведите матрицу

на экран.

2. Определите радиус и диаметр графа G, используя матрицу смежности

графа.

3. Определите подмножества периферийных и центральных вершин

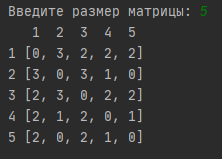
графа G, используя матрицу смежности.

4. Найдите изолированные, концевые и доминирующие вершины.

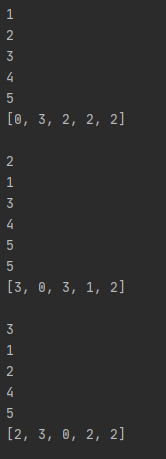
**Задание 2\*:**

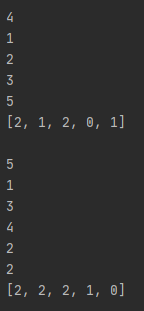
1. Постройте для графа G матрицу инцидентности.
2. Определите радиус и диаметр графа G, используя матрицу инцидентности графа.
3. Определите подмножества периферийных и центральных вершин графа G, используя матрицу инцидентности.

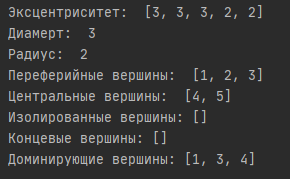
Взвешенный неориентированный граф.



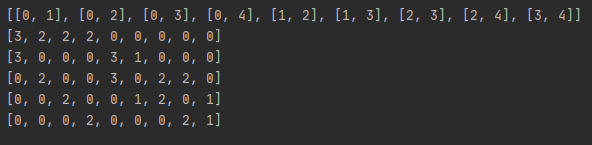
Поиск в ширину для всех вершин графа

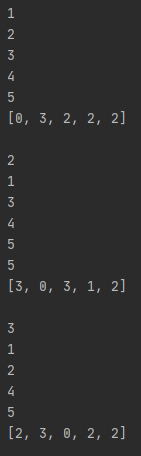


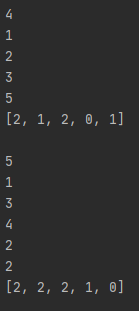


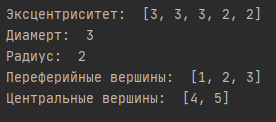


Взвешенный неориентированный граф, представленный матрицей инцидентности.









**Листинг**

import Graph  
import Operations  
  
n = int(input("Введите размер матрицы: "))  
  
graph = []  
Graph.generator\_w\_matrix(graph, n, 3)  
# graph = [[0, 2, 3, 0, 0, 0],  
# [2, 0, 0, 5, 0, 0],  
# [3, 0, 0, 0, 1, 0],  
# [0, 5, 0, 0, 0, 0],  
# [0, 0, 1, 0, 0, 0],  
# [0, 0, 0, 0, 0, 0]]  
Graph.print\_matrix(graph, 0, n)  
  
eccentricity = []  
for i in range(n):  
 Dist = [1000 for \_ in range(n)]  
 Operations.bfsd\_matrix\_real(graph, Dist, i)  
 print()  
 for j in range(n):  
 if Dist[j] == 1000:  
 Dist[j] = 0  
 eccentricity.append(max(Dist))  
  
print("Эксцентриситет: ", eccentricity)  
Diameter = max(eccentricity)  
Radius = min(eccentricity)  
print("Диамерт: ", Diameter)  
print("Радиус: ", Radius)  
  
peripheralNode = []  
centralNode = []  
Operations.search\_peripheral\_or\_central\_node(eccentricity, Diameter, peripheralNode)  
Operations.search\_peripheral\_or\_central\_node(eccentricity, Radius, centralNode)  
print("Переферийные вершины: ", peripheralNode)  
print("Центральные вершины: ", centralNode)  
  
Degrees = [0 for \_ in range(n)]  
isolationNode = []  
endNode = []  
dominationNode = []  
  
Operations.degree\_of\_node(graph, Degrees, n)  
for i in range(n):  
 if Degrees[i] == 0:  
 isolationNode.append(i + 1)  
 elif Degrees[i] == 1:  
 endNode.append(i + 1)  
 elif Degrees[i] == n - 1:  
 dominationNode.append(i + 1)  
  
print("Изолированные вершины:", isolationNode)  
print("Концевые вершины:", endNode)  
print("Доминирующие вершины:", dominationNode)  
print()  
  
incGraph = []  
edge = []  
Visits = [0 for \_ in range(n)]  
Graph.generator\_incident\_matrix(graph, edge, incGraph, Visits, n)  
  
eccentricityInc = []  
for i in range(n):  
 Dist = [1000 for \_ in range(n)]  
 Operations.bfsd\_incMatrix\_real(incGraph, edge, Dist, i)  
 print()  
 for j in range(n):  
 if Dist[j] == 1000:  
 Dist[j] = 0  
 eccentricityInc.append(max(Dist))  
  
print("Эксцентриситет: ", eccentricityInc)  
Diameter = max(eccentricityInc)  
Radius = min(eccentricityInc)  
print("Диамерт: ", Diameter)  
print("Радиус: ", Radius)  
  
peripheralNodeInc = []  
centralNodeInc = []  
Operations.search\_peripheral\_or\_central\_node(eccentricity, Diameter, peripheralNodeInc)  
Operations.search\_peripheral\_or\_central\_node(eccentricity, Radius, centralNodeInc)  
print("Переферийные вершины: ", peripheralNodeInc)  
print("Центральные вершины: ", centralNodeInc)

**Функция search\_peripheral\_or\_central\_node**

def search\_peripheral\_or\_central\_node(eccentricity, property, massOfNode):  
 for i in range(len(eccentricity)):  
 if eccentricity[i] == property:  
 massOfNode.append(i+1)

**Функция degree\_of\_node**

def degree\_of\_node(graph, degrees, n):  
 for i in range(n):  
 for j in range(n):  
 if graph[i][j] != 0:  
 degrees[i] += 1

**Функция bfsd\_incMatrix\_real**

def bfsd\_incMatrix\_real(graph, edge, dist, node):  
 queue = Queue()  
 queue.add(node)  
 while queue.len != 0:  
 item = queue.pop()  
 dist[node] = 0  
 print(item+1)  
 for i in range(len(edge)):  
 if item != edge[i][1]:  
 if graph[item][i] != 0 and dist[edge[i][1]] > dist[item] + graph[item][i]:  
 queue.add(edge[i][1])  
 dist[edge[i][1]] = dist[item] + graph[item][i]  
 elif item == edge[i][1]:  
 if graph[item][i] != 0 and dist[edge[i][0]] > dist[item] + graph[item][i]:  
 queue.add(edge[i][0])  
 dist[edge[i][0]] = dist[item] + graph[item][i]  
 print(dist)

**Функция generator\_incident\_matrix**

def generator\_incident\_matrix(graph, edge, incGraph, visits, n):  
 for i in range(n):  
 for j in range(n):  
 if graph[i][j] != 0 and i != j and visits[j] == 0:  
 b = []  
 b.append(i)  
 b.append(j)  
 edge.append(b)  
 visits[i] = 1  
 for i in range(n):  
 a = [0 for \_ in range(len(edge))]  
 incGraph.append(a)  
 print(edge)  
  
 for j in range(len(edge)):  
 incGraph[edge[j][0]][j] = graph[edge[j][0]][edge[j][1]]  
 incGraph[edge[j][1]][j] = graph[edge[j][0]][edge[j][1]]  
 for j in range(n):  
 print(incGraph[j])  
 print()