Министерство образования Республики Беларусь

Учреждения образования

Брестский государственный технический университет

Кафедра ИИТ

Лабораторная работа №3

По дисциплине ТМОИС

За 2 семестр

Тема:графы

Выполнил :

Студент 1 курса

Группы ИИ-21

Корпач Д.Р.

Проверил

Глущенко

Брест 2022

Вариант 7

import collections as coll

def DFS(graph, start, VISITED=None):

    if VISITED is None:              #

        VISITED = set()              #обход в ширину

    VISITED.add(start)               #

    for next in set(graph[start]) - VISITED:

        DFS(graph, next, VISITED)

    return VISITED

def BFS(graph, root):

    VISITED, queue = set(), coll.deque([root])

    VISITED.add(root)

    while queue:

        vertex = queue.popleft()

        for neighbour in graph[vertex]:

                                          #

            if neighbour not in VISITED:  #обход в глубину

                VISITED.add(neighbour)    #

                queue.append(neighbour)

    return VISITED

if \_\_name\_\_ == '\_\_main\_\_':

    graph = {0: [3],

             1: [3,5],

             2: [5],

             3: [0,1,4],

             4: [3,5],

             5: [1,2,4],

             6: [7],

             7: [6]}

    VISITED = BFS(graph, 0)

    count = 1

    for NODE in graph:

        if NODE not in VISITED:

            VISITED = BFS(graph, NODE)

            count += 1

    print(f'BFS: count = {count}')

    VISITED = DFS(graph, 0)

    count = 1

    for NODE in graph:

        if NODE not in VISITED:

            VISITED = DFS(graph, NODE)

            count += 1

    print(f'DFS: count = {count}')



**Алгоритм дейкстры**

class Graph():

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = [[0 for column in range(vertices)]

                      for row in range(vertices)]

    def printSolution(self, dist):

        print("Vertex \t Distance from Source")

        for node in range(self.V):

            print(node, "\t\t", dist[node])

    def minDistance(self, dist, sptSet):

        # Initialize minimum distance for next node

        min = 1e7

        for v in range(self.V):

            if dist[v] < min and sptSet[v] == False:

                min = dist[v]

                min\_index = v

        return min\_index

    def dijkstra(self, src):

        dist = [1e7] \* self.V

        dist[src] = 0

        sptSet = [False] \* self.V

        for cout in range(self.V):

            u = self.minDistance(dist, sptSet)

            sptSet[u] = True

            for v in range(self.V):

                if (self.graph[u][v] > 0 and

                   sptSet[v] == False and

                   dist[v] > dist[u] + self.graph[u][v]):

                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

g = Graph(7)

g.graph = [[0, 3, 7, 4, 0, 0, 0],

           [3, 0, 5, 2, 0, 0, 10],

           [7, 5, 0, 0, 0, 4, 0],

           [4, 2, 0, 0, 4, 0, 0],

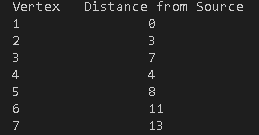
           [0, 0, 0, 4, 0, 5,6],

           [0, 0, 4, 0, 5, 0,7],

           [0, 10, 0, 0, 6, 7, 0]

           ]

g.dijkstra(0)



**Алгоритм флойда-уоршелла**

from math import inf

maxV,V=1000,7

def FU(D):

    for i in range(V):

        D[i][i]=0

    for k in range(V):

        for i in range(V):

            for j in range(V):

                D[i][j] = min(D[i][j], D[i][k] + D[k][j])

    for i in range(V):

        for j in range(V):

            print(D[i][j],end="\t")

        print()

GR=[[0,3,7,4,inf,inf,inf],

    [3,0,5,2,inf,inf,10],

    [7,5,0,inf,inf,4,inf],

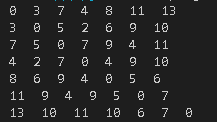
    [4,2,inf,0,4,inf,inf],

    [inf,inf,inf,4,0,5,6,],

    [inf,inf,4,inf,5,0,7],

    [inf,10,inf,inf,6,7,0]]

FU(GR)

****