**Cho đồ thị hàm số G = (V,E) như bên dưới:**

**A diagram of a network

AI-generated content may be incorrect.**

1. **Biểu diễn ma trận kề**

edges = [

    ("A", "C"), ("A", "D"), ("A", "E"), ("A", "F"),

    ("C", "H"),

    ("D", "E"), ("D", "H"),

    ("E", "K"), ("E", "I"),

    ("F", "G"), ("F", "I"),

    ("H", "K"),

    ("K", "B"),

    ("I", "B"), ("I","K")

]

nodes = ["A","B","C","D","E","F","G","H","I","K"]

graph = {node: {} for node in nodes}

for u , v in edges:

  if graph[u] == {}:

    graph[u] = {v}

  else:

    graph[u].add(v)

# print(graph)

import pprint

pprint.pprint(graph)

1. **Thao tác duyệt đồ thị**
2. **Liệt kê các đỉnh trong đồ thị**

for u  in graph:

  print(u,end=" ")

1. **Liệt kê tất cả các cạnh đồ thị hiển thị dạng danh sách kề**

def HienThiDoThi(G):

    for u in G:

      print(f"+ Đỉnh {u}",end=": ")

      for v in G[u]:

        print(v,end=", ")

      print()

    pass

HienThiDoThi(graph)

1. **Cho hai đỉnh A,B. Hỏi hai đỉnh A, B có kề nhau không?**

def LaKe(G, a, b):

    """

    input: G, a, b

    return:

    + -1: dinh a, hay b khong ton tai

    +  0: dinh a, b ton tai va a khong ke b

    +  1: dinh a, b ton tai va a, b ke nhau

    """

    result = None

    if a not in G or b not in G:

      result = -1

    elif b in G[a]:

      result = 1

    else:

      result = 0

    return result

ADJ = graph

a = "L"; b = "R"; print(f"{a} ke {b}: {LaKe(ADJ, a, b)}")

a = "A"; b = "R"; print(f"{a} ke {b}: {LaKe(ADJ, a, b)}")

a = "E"; b = "D"; print(f"{a} ke {b}: {LaKe(ADJ, a, b)}")

a = "D"; b = "E"; print(f"{a} ke {b}: {LaKe(ADJ, a, b)}")

1. **Cho một đỉnh A. Hỏi danh sách các đỉnh kề với A**

def LayKe(G, a):

    """

    input: G, a

    return:

    + None: neu a khong ton tai

    + []  : a khong ke voi bat ky dinh nao

    + [x, y, ...]: mang cac dinh ke voi a

    """

    result = None

    """ CÁC BẠN LÀM BÀI Ở ĐÂY """

    # ...

    if a not in G:

      result = None

    else:

      result = list(G[a])

    return result

a = "P"; print(f"Danh sach ke voi dinh {a}: {LayKe(ADJ, a)}")

a = "A"; print(f"Danh sach ke voi dinh {a}: {LayKe(ADJ, a)}")

a = "B"; print(f"Danh sach ke voi dinh {a}: {LayKe(ADJ, a)}")

a = "D"; print(f"Danh sach ke voi dinh {a}: {LayKe(ADJ, a)}")

1. **Đọc và lưu đồ thị**
2. **Lưu đồ thị**

def LuuDoThi(G, file\_path, verbose = True):

    import os

    # Tạo thư mục chứa file\_path

    file\_dir = os.path.dirname(file\_path)

    if file\_dir != "" and os.path.exists(file\_path) == False:

        os.makedirs(file\_dir)

        if verbose: print(f"+ Tao thu muc: {file\_dir}")

    # Lưu đồ thị

    with open(file\_path, "wt") as file:

        # ...

        file.write(f"{len(G)}\n")

        for node, neighbors in G.items():

            line = f"{node}"

            for neighbor in neighbors:

                line += f" {neighbor}"

            file.write(f"{line}\n")

        if verbose: print(f"Luu do thi thanh cong xuong tap tin: {file\_path}")

    pass

LuuDoThi(ADJ, "dske1.txt", verbose = True)

with open("dske1.txt", "rt") as file:

    lines = file.readlines()

    for line in lines: print(line, end="")

1. **Đọc đồ thị từ tập tin**

def DocDoThi(file\_path, verbose = True):

    """

    return:

    + None: doc that bai

    + <>None: tra ve do thi

    """

    import os

    result = None

    if os.path.exists(file\_path) == False:

        result = None

    else:

        G = {}

        with open(file\_path, "rt") as file:

            n = int(file.readline())

            """ CÁC BẠN LÀM BÀI Ở ĐÂY """

            # ...

            for i in range(n):

                line = file.readline().strip().split()

                node = line[0]

                neighbors = set(line[1:])

                G[node] = neighbors

            pass

        result = G

    return result

G1 = DocDoThi("dske1.txt", verbose = True)

pprint.pprint(G1)

1. **Tìm kiếm đường đi trên đồ thị**
2. **Tìm kiếm theo chiều rộng (BFS)**

import pprint

def BFS(G, start, goal):

    """

    return:

    + mang chua duong di tu a --> b, neu rong la ko co duong di

    + None: dinh a, b khong hop le

    """

    result = None

    if G.get(start) is None or G.get(goal) is None:

        result = None

    else:

        path = {}     # path[a]=b nghia la muon di toi dinh b phai qua a

        s\_open   = [] # hàng đợi (Queue)

        s\_closed = [] # đã duyệt

        # đưa start vào open

        s\_open.append(start)

        path[start] = None

        while len(s\_open)>0:

            """ CÁC BẠN LÀM BÀI Ở ĐÂY """

            current = s\_open.pop(0)  # lấy từ đầu queue

            if current == goal:

                break

            s\_closed.append(current)

            for neighbor in G[current]:

                if neighbor not in path:   # chưa có đường đến

                    s\_open.append(neighbor)

                    path[neighbor] = current

        return path

        pass

    return path

path = BFS(ADJ, "A", "B")

pprint.pprint(path)

def find\_path(path, start, goal):

    """

    Cho mang truy hoi duong, tim ds dinh tu start --> goal

    """

    result = []

    """ CÁC BẠN LÀM BÀI Ở ĐÂY """

    # ...

    if start not in path or goal not in path:

        return result  # không có đường đi

    current = goal

    while current is not None:

        result.append(current)

        current = path.get(current)

    result.reverse()  # đảo ngược lại để từ start -> goal

    return result

    pass

find\_path(path, 'A', 'B')

1. **Tìm kiếm theo chiều sâu (DFS)**

import pprint

def DFS(G, start, goal):

    """

    return:

    + mang chua duong di tu a --> b, neu rong la ko co duong di

    + None: dinh a, b khong hop le

    """

    result = None

    if G.get(start) is None or G.get(goal) is None:

        result = None

    else:

        path = {}     # path[a]=b nghia la muon di toi dinh b phai qua a

        s\_open   = []

        s\_closed = []

        # đưa start vào open

        s\_open.append(start)

        path[start] = None

        while len(s\_open)>0:

            """ CÁC BẠN LÀM BÀI Ở ĐÂY """

            # ...

            current = s\_open.pop()  # lấy phần tử cuối (stack)

            if current == goal:

                break

            s\_closed.append(current)

            for neighbor in G[current]:

                if neighbor not in path:  # chưa thăm

                    s\_open.append(neighbor)

                    path[neighbor] = current

            pass

    return path

path = DFS(ADJ, "A", "B")

pprint.pprint(path)