* Biểu diễn danh sách kề của đồ thị:

ADJ = {

    'A': ['C', 'E', 'F', 'D'],

    'B': [],

    'C': ['H'],

    'D': ['E', 'H'],

    'E': ['K', 'I'],

    'F': ['I', 'G'],

    'G': [],

    'H': ['K'],

    'I': ['K', 'B'],

    'K': ['B']

}

* **Thao tác duyệt đồ thị** duyệt các keys trong đồ thị ADJ và in ra

print("(a) Liệt kê các đỉnh trong đồ thị")

print("Các đỉnh của đồ thị: ", end=" ")

for i in ADJ.keys():

    print(i, end=" ")

* Duyệt qua các đỉnh trong i và duyệt các đỉnh kề với đỉnh i đó

def HienThiDoThi(G):

    """ CÁC BẠN LÀM BÀI Ở ĐÂY """

    for i in G:

        print(f"Đỉnh {i} : ", end="")

        for j in G[i]:

            print(f"{j} ", end="")

        print()

    pass

* Kiểm tra 2 đỉnh kề nhau

HienThiDoThi(ADJ)

print("(c) 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

    """ CÁC BẠN LÀM BÀI Ở ĐÂY """

    # ...

    if a not in G or b not in G:

        result = -1

    elif a in G and b in G and b in G[a]:

        result = 1

    else:

        result = 0

    return result

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)}")

print("(d) Cho một đỉnh A. Hỏi danh sách các đỉnh kề với A")

* Ta có đầu vào dthi G và đỉnh a. nếu a có trg G thì gán None, nếu a trong G và số lượng phần tử kề tại đỉnh a == 0 thì gán ds rỗng. Ngược 2 th đó thì ta gán gtri de return là list kề

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

    elif a in G and len(G[a]) == 0:

        result = []

    else:

        result = 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)}")

* Đọc lưu đồ thị
* Ghi số lượng đỉnh lên dòng đầu tiên của file sau đó duyệt các đỉnh và các đỉnh kề với đỉnh đó. Nối các đỉnh kề với line sau ghi line xuống file và xuống dòng
* print("1. Lưu đồ thị xuống tập tin")
* def LuuDoThi(G, file\_path, verbose = True):
* import os
* """ CÁC BẠN LÀM BÀI Ở ĐÂY """
* # 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:
* """ CÁC BẠN LÀM BÀI Ở ĐÂY """
* # ...
* file.write(f"{len(G)}\n")
* for u in G:
* line = u
* for v in G[u]:
* line += f" {v}"
* 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="")
* Mỗi lần lặp sẽ đọc 1 dòng thông tin của một đỉnh và các đỉnh kề. Sau đó từng line 1 loại bỏ khoảng trắng và tách thành từng chuỗi danh sách các từ. Phần tử đầu tiên là tên đỉnh sau đó tạo 1 list rỗng G[u] rỗng duyệt qua từng line 1 và bỏ phần tử đầu tiên của từng line và đưa vào list đó
* print("2. Đọc đồ thị từ tập tin")
* import pprint
* 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 \_ in range(n):
* line = file.readline().strip().split()
* u = line[0]
* G[u] = []
* for v in line[1:]:
* G[u].append(v)
* pass
* result = G
* return result
* G1 = DocDoThi("dske1.txt", verbose = True)
* pprint.pprint(G1)
* lấy node từ danh sách mở, đưa sang danh sách đóng, kiểm tra goal, rồi thêm các đỉnh kề chưa duyệt vào s\_open và lưu trong path

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 = []

        s\_closed = []

        # đưa start vào open

        s\_open.append(start)

        path[start] = None

        while len(s\_open) > 0:

            #...

            node = s\_open.pop(0)

            s\_closed.append(node)

            if node == goal:

                return path

            for neighbor in G[node]:

                if neighbor not in s\_closed and neighbor not in s\_open:

                    s\_open.append(neighbor)

                    path[neighbor] = node

    return path

Đoạn code này thực hiện bước **truy vết đường đi** từ goal về start:

* Tạo danh sách result rỗng để chứa đường đi.
* Nếu không có path hoặc goal không nằm trong path → trả về rỗng (không có đường đi).
* Bắt đầu từ goal, lần lượt truy ngược cha trong path.
* Mỗi bước, chèn đỉnh hiện tại (current) vào đầu result để giữ thứ tự từ start → goal.
* Tiếp tục cho đến khi gặp None (nghĩa là đến đỉnh xuất phát).
* Cuối cùng trả về result là danh sách đường đi từ start → goal.
* def find\_path(path, start, goal):
* """
* Cho mang truy hoi duong, tim ds dinh tu start --> goal
* """
* result = []
* # Check if path exists
* if path is None or goal not in path:
* return result
* # Reconstruct path from goal to start
* current = goal
* while current is not None:
* result.insert(0, current)  # Insert at beginning to maintain order
* current = path[current]
* return result
* khởi tạo danh sách open và closed, thêm đỉnh bắt đầu vào open lưu vết: start không có cha lặp khi open còn phần tử và lấy đỉnh cuối trong open để duyệt (DFS) chuyển node từ open sang closed kiểm tra nếu gặp đỉnh đích trả về đường đi đã tìm được duyệt các đỉnh kề của node theo thứ tự đảo ngược bỏ qua đỉnh đã duyệt hoặc đang chờ thêm đỉnh kề hợp lệ vào open lưu cha của neighbor để truy ngược đường đi.
* 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:
* node = s\_open.pop()  # DFS uses pop() instead of pop(0) like BFS
* s\_closed.append(node)
* if node == goal:
* return path
* for neighbor in reversed(G[node]):  # reversed to maintain original order
* if neighbor not in s\_closed and neighbor not in s\_open:
* s\_open.append(neighbor)
* path[neighbor] = node
* return path