

# Báo cáo kết quả lab1

## Bài 1: Biểu diễn đồ thị

### 1. Biểu diễn bằng danh sách/ ma trận kề

Kết quả:

```
{'A': ['C', 'D', 'E', 'F'],  
'B': [],  
'C': ['H'],  
'D': ['H', 'E'],  
'E': ['K', 'I'],  
'F': ['I', 'G'],  
'G': [],  
'H': ['K'],  
'I': ['B', 'K'],  
'K': ['B']}
```

Code:

```
ADJ = {  
    'A': ['C', 'D', 'E', 'F'],  
    'B': [],  
    'C': ['H'],  
    'D': ['H', 'E'],  
    'E': ['K', 'I'],  
    'F': ['I', 'G'],  
    'G': [],  
    'H': ['K'],  
    'I': ['B', 'K'],  
    'K': ['B']  
}
```

```
import pprint  
pprint.pprint(ADJ)
```

### 2. Thao tác duyệt đồ thị

#### a. Liệt kê các đỉnh trong đồ thị

(a) Liệt kê các đỉnh trong đồ thị  
Các đỉnh của đồ thị: A B C D E F G H I K

```
print("(a) Liệt kê các đỉnh trong đồ thị")  
print("Các đỉnh của đồ thị: ", end=" ")  
# Liệt kê các đỉnh trong đồ thị  
for v in ADJ.keys():  
    print(v, end=" ")
```

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


```
↔ (b) Liệt kê tất cả các cạnh đồ thị hiển thị dạng danh sách kề  
Danh sách kề của đồ thị:  
Đỉnh A : C, D, E, F  
Đỉnh B :  
Đỉnh C : H  
Đỉnh D : H, E  
Đỉnh E : K, I  
Đỉnh F : I, G  
Đỉnh G :  
Đỉnh H : K  
Đỉnh I : B, K  
Đỉnh K : B
```

```
▶ print("(b) Liệt kê tất cả các cạnh đồ thị hiển thị dạng danh sách kề")  
print("Danh sách kề của đồ thị: ")  
  
def HienThiDoThi(G):  
    for v in G:  
        print("Đỉnh", v, ":", ", ".join(G[v]) if G[v] else " ")  
  
HienThiDoThi(ADJ)
```


c. Cho hai đỉnh A, B có kề nhau không ?

```
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: đỉnh a hoặc b không tồn tại  
    + 0: đỉnh a, b tồn tại nhưng không kề nhau  
    + 1: đỉnh a, b kề nhau  
    """  
  
    # kiểm tra tồn tại  
    if a not in G or b not in G:  
        return -1  
  
    # kiểm tra kề  
    if b in G[a]:  
        return 1  
    else:  
        return 0
```

 (c) Cho hai đỉnh A, B. Hỏi hai đỉnh A, B có kề nhau không?

```
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)}")
```

 L ke R: -1  
A ke R: -1  
E ke D: 0  
D ke E: 1

d. Cho một đỉnh A. Hỏi danh sách các đỉnh kề với A

```

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

def LayKe(G, a):
    """
    input: G, a
    return:
    + None: neu a không tồn tại
    + [] : a không kề với bất kỳ đỉnh nào
    + [x, y, ...]: mảng các đỉnh kề với a
    """
    if a not in G:      # không tồn tại đỉnh a
        return None

    return G[a]        # trả về list kề (có thể rỗng [])

# Test
a = "P"; print(f"Danh sách kề với đỉnh {a}: {LayKe(ADJ, a)}")
a = "A"; print(f"Danh sách kề với đỉnh {a}: {LayKe(ADJ, a)}")
a = "B"; print(f"Danh sách kề với đỉnh {a}: {LayKe(ADJ, a)}")
a = "D"; print(f"Danh sách kề với đỉnh {a}: {LayKe(ADJ, a)}")

```

```

➡ (d) Cho một đỉnh A. Hỏi danh sách các đỉnh kề với A
Danh sách kề với đỉnh P: None
Danh sách kề với đỉnh A: ['C', 'D', 'E', 'F']
Danh sách kề với đỉnh B: []
Danh sách kề với đỉnh D: ['H', 'E']

```

### 3. Đọc và lưu đồ thị

#### 3.1 Lưu đồ thị xuống tập tin

```

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:
        # ghi số đỉnh
        file.write(str(len(G)) + "\n")
        # ghi danh sách kề
        for v in G:
            line = v
            if len(G[v]) > 0:
                line += " " + " ".join(G[v])
            file.write(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. Lưu đồ thị xuống tập tin
Luu do thi thanh cong xuong tap tin: dske1.txt
10
A C D E F
B
C H
D H E
E K I
F I G
G
H K
I B K
K B

```

### 3.2 Đọc đồ thị từ tập tin

```

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())
            for line in file:
                parts = line.strip().split()
                if len(parts) == 0:
                    continue
                v = parts[0]          # đỉnh
                neighbors = parts[1:] # các đỉnh kề
                G[v] = neighbors

            pass
        result = G
    return result

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

```

#

```

➡ 2. Đọc đồ thị từ tập tin
{'A': ['C', 'D', 'E', 'F'],
 'B': [],
 'C': ['H'],
 'D': ['H', 'E'],
 'E': ['K', 'I'],
 'F': ['I', 'G'],
 'G': [],
 'H': ['K'],
 'I': ['B', 'K'],
 'K': ['B']}

```

## Bài 2: Tìm kiếm đường đi trên đồ thị

### 1. Tìm kiếm theo chiều rộng (BFS)

```

import pprint

def BFS(G, start, goal):
    """
    return:
    + path: dict cha của từng đỉnh (path[v] = u nghĩa là đi tới v từ u)
    + None: nếu start hoặc goal không hợp lệ
    """
    if G.get(start) is None or G.get(goal) is None:
        return None

    path = {start: None}  # lưu cha của mỗi node
    s_open = [start]      # hàng đợi (queue)
    s_closed = []          # đã duyệt

    while len(s_open) > 0:
        u = s_open.pop(0)  # lấy node đầu queue
        s_closed.append(u)

        for v in G[u]:
            if v not in path:  # chưa đi qua
                path[v] = u
                s_open.append(v)

            # Nếu đã gặp goal thì có thể return luôn (tối ưu BFS)
            if v == goal:
                return path

    return path

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

```

```

{'A': None,
 'B': 'K',
 'C': 'A',
 'D': 'A',
 'E': 'A',
 'F': 'A',
 'G': 'F',
 'H': 'C',
 'I': 'E',
 'K': 'E'}

```

```
def find_path(path, start, goal):
    """
    Cho mảng truy hồi đường (path),
    tìm danh sách đỉnh đi từ start --> goal
    """
    result = []

    # Nếu không có thông tin hoặc goal không tồn tại trong path
    if path is None or goal not in path:
        return result

    # Lần ngược từ goal về start
    current = goal
    while current is not None:
        result.append(current)
        current = path.get(current, None)

    # Đang đi ngược, cần đảo lại để ra đúng thứ tự start -> goal
    result.reverse()

    # Nếu node đầu tiên không phải start thì không có đường đi
    if len(result) == 0 or result[0] != start:
        return []

    return result

find_path(path, 'A', 'B')

['A', 'E', 'K', 'B']
```

## 2. Tìm Kiếm theo chiều sâu (DFS)



```

import pprint

def DFS(G, start, goal):
    """
    return:
    + dict path[a] = b nghĩa là muốn đi tới a thì phải qua b
    + None: đỉnh start hoặc goal không hợp lệ
    """
    if G.get(start) is None or G.get(goal) is None:
        return None

    path = {}          # path[v] = u (cha của v)
    s_open = []        # stack
    s_closed = set()    # tập đã duyệt

    # đưa start vào open
    s_open.append(start)
    path[start] = None

    while len(s_open) > 0:
        # lấy đỉnh cuối stack
        current = s_open.pop()
        if current in s_closed:
            continue
        s_closed.add(current)

        # nếu đã tới goal thì dừng
        if current == goal:
            continue

        # duyệt các đỉnh kề theo thứ tự thuận (không đảo ngược!)
        for neighbor in G[current]:
            if neighbor not in s_open and neighbor not in s_closed:
                s_open.append(neighbor)
                path[neighbor] = current

    return path

```

```

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

```

```

→ {'A': None,
   'B': 'I',
   'C': 'A',
   'D': 'A',
   'E': 'A',
   'F': 'A',
   'G': 'F',
   'H': 'D',
   'I': 'F',
   'K': 'I'}

```