Bài tập thực hành 1

Cài đặt thuật toán tìm kiếm DFS và BFS bằng Python

1. Hướng dẫn

1.1 Mã giả thuật toán

Đầu vào: trạng thái bắt đầu, hàm successor, hàm kiểm tra trạng thái đích

Đầu ra: kế hoạch tìm được (chuỗi các hành động để đi từ trạng thái bắt đầu đến trạng thái đích)

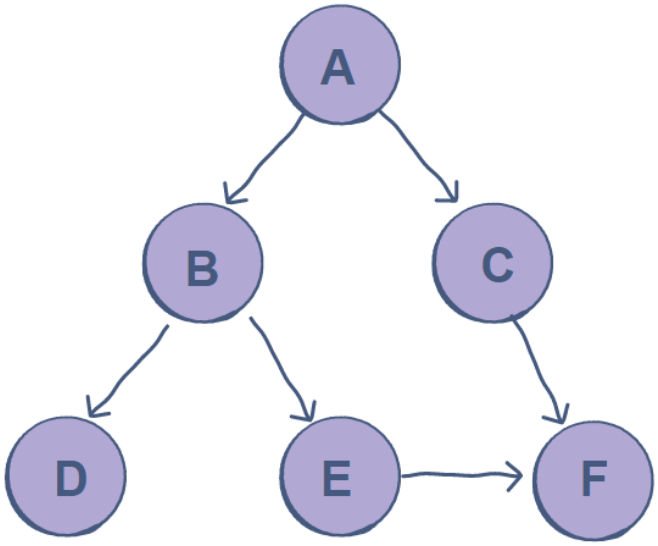
Quá trình thực hiện:

- Khởi tạo:
  - `fringe`: gồm một kế hoạch ứng với trạng thái bắt đầu
  - `closed set`: rỗng
- Trong khi `fringe` chưa rỗng:
  - Lấy một kế hoạch ra khỏi `fringe` theo một chiến lược nào đó
  - Nếu kế hoạch này đi tới đích (dùng hàm kiểm tra trạng thái đích): RETURN kế hoạch!
  - Nếu trạng thái cuối của kế hoạch này chưa có trong `closed set`:
    - Đưa trạng thái vào `closed set`
    - Mở rộng ra (dựa vào hàm successor) và đưa các kế hoạch mới vào `fringe`
- Nếu ra được khỏi vòng lặp nghĩa là: không tìm thấy lời giải

1.2 Sample source code (BFS)

Đoạn code sau cài đặt BFS theo phương pháp đơn giản.

1. Tạo ra **Fringe** và **Closed set** rỗng.
2. Lấy 1 đỉnh ***s*** từ đầu **Fringe**.
3. Nếu đỉnh này là đích thì dừng.
4. Duyệt qua lần lượt các đỉnh kề của đỉnh **s** này.
   a. Nếu đỉnh này chưa nằm trong close set: thêm đỉnh này vào **Queue** và **Fringe**.
5. Lặp lại bước 2-4 cho đến khi queue rỗng hoặc tìm thấy đỉnh đích




```
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}

visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, start,end):
  queue.append(start)
  while queue:
    s = queue.pop(0)

    print (s, end = " ")
    if s == end:
        return
    visited.append(s)
    for neighbour in graph[s]:
      if neighbour not in visited:
        queue.append(neighbour)

# Driver Code
bfs(visited, graph, 'A','E')
```

Đoạn code tiếp theo vẫn cài đặt BFS nhưng có xuất ra đường đi

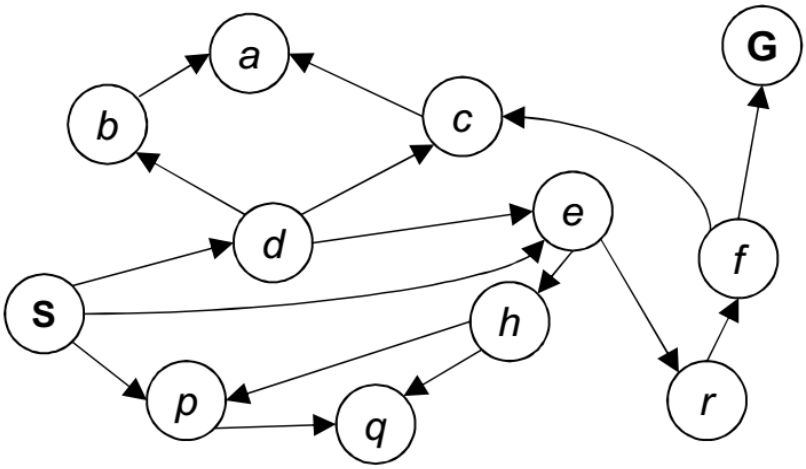
```
# graph is in adjacent list representation
```

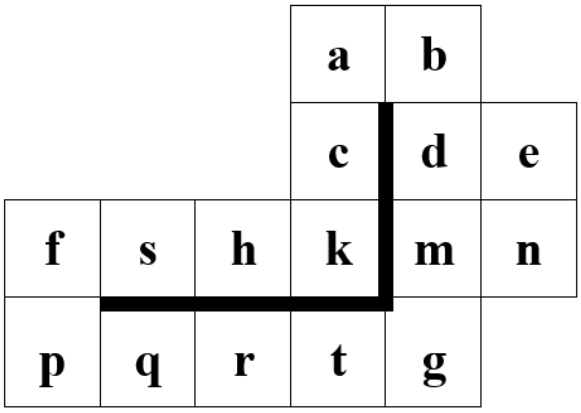
```
graph = {
        '1': ['2', '3', '4'],
        '2': ['5', '6'],
        '5': ['9', '10'],
        '4': ['7', '8'],
        '7': ['11', '12']
        }

def bfs(graph, start, end):
    # maintain a queue of paths
    visited = []
    queue = []

    # push the first path into the queue
    queue.append([start])
    while queue:
        # get the first path from the queue
        path = queue.pop(0)
        # get the last node from the path
        node = path[-1]
        # path found
        if node == end:
            return path
        visited.append(node)
        # enumerate all adjacent nodes, construct a new path and push
it into the queue
        for neighbour in graph.get(node, []):
            if neighbour not in visited:
                new_path = list(path)
                new_path.append(neighbour)
                queue.append(new_path)

print (bfs(graph, '1', '11'))
```

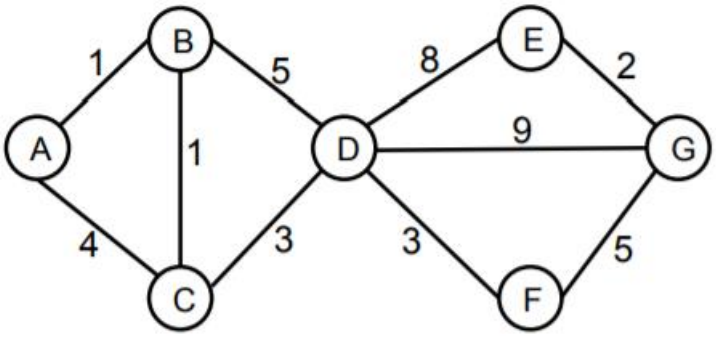
2. Yêu cầu bài tập

2.1 Sinh viên cài đặt lại hai đoạn code mẫu bên trên. Cho biết kết quả thực thi với các đồ thị sau.

Tìm đường đi từ S ➔ G. A ➔ G Cho đồ thị 3







2.2 Dựa vào đoạn code mẫu về BFS. Sinh viên cài đặt thuật toán DFS và cho biết kết quả thực thi DFS với các đồ thị như phần 2.1

3. Qui định nộp

- Sinh viên nộp một tập tin nén, có tên là ***<MSSV>.zip*** hoặc ***<MSSV>.rar*** chứa source code và báo cáo của chương trình.
- Sinh viên nộp kèm một file báo cáo ghi mức độ hoàn thành công việc của mình

Bài giống nhau hay nộp file rác sẽ 0 điểm MÔN HỌC.