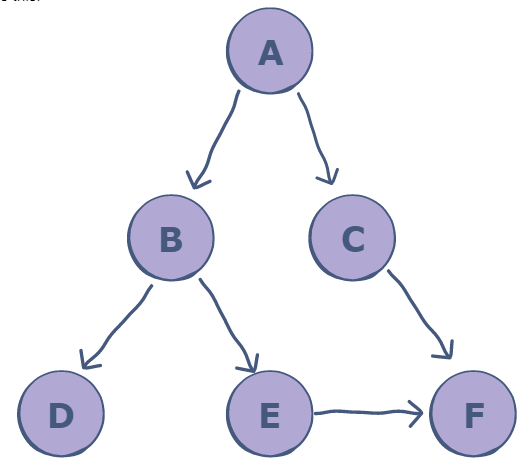
**HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH**

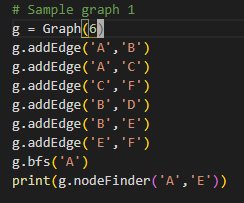
**Ngôn ngữ sử dụng: Python 3.7**

**BÀI TẬP 1: BREADTH-FIRST SEARCH:**

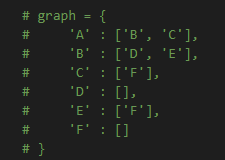
1. **Cách tạo cây**



* Giả sử ta có cây như hình trên, để tạo ra cây đầu tiên ta cần khai báo:
  + Tạo một biến để chứa class Graph()
  + Khai báo số node có trong cây đó (trong trường hợp này là 6 nodes)
  + Cú pháp như sau:
    - Tên biến = Graph(Số node)
    - **g = Graph(6)**



* Sau khi đã khởi tạo class và khai báo số nodes có trong cây đó, tiếp đến ta tiến hành tạo cây để thuật toán BFS có thể chạy được:
  + Sử dụng hàm **addEdge(parentNode, childNode)** để tạo ra cây. Ví dụ từ hình của cây trên ta thấy rằng node A có 2 node con là B và C ta lần lượt khởi tạo như sau:
    - **g.addEdge(‘A’,’B’)**
    - **g.addEdge(‘A’,’C’)**
    - Tương tự ta tiếp tục tiến hành như thế đến khi tạo ra hết cây
  + Hình dưới thể hiện sơ đồ cây được tạo ra dưới dạng dictionary structure của python để ta có thể dễ hình dung hơn



1. **In ra cách di chuyển của thuật toán BFS (BFS traversal):**

* Để có thể in ra cách mà thuật toán BFS di chuyển ở cây trên, ta thực hiện gọi hàm **bfs(initialNode)** với:
  + initialNode là node khởi tạo để thuật toán BFS có thể thực hiện việc quét qua toàn bộ cây
  + Cú pháp: **g.bfs(‘A’)**

1. **Tìm kiếm một thành phần trong BFS:**

* Để tìm kiếm một thành phần nào đó trong cây, sử dụng hàm **nodeFinder(initialNode, target)** 
  + Cú pháp: **g.nodeFinder(‘A’,’E’)**
  + Giải thích: **nodeFinder()** sẽ thực hiện tìm kiếm từ ‘A’ đến hết cây
* **Lưu ý: initialNode nên là rootNode để có thể quét được toàn bộ cây**

1. **Bảng tổng hợp các hàm:**

|  |  |
| --- | --- |
| **Tên hàm và cú pháp** | **Công dụng** |
| Graph(Num\_of\_Nodes) | Khởi tạo số nodes tồn tại trong cây |
| addEdge(parentNode,childNode) | Tạo nhánh cây mới |
| bfs(initialNode) | In ra cách thức di chuyển của thuật toán BFS từ node đã định sẵn trước (initialNode) |
| nodeFinder(initialNode, target) | Tìm một node trong cây bắt đầu di chuyển từ node đã định sẵn trước (nên sử dụng rootNode để BFS có thể quét qua toàn bộ cây).  Hàm trả về **True/False** |

**BÀI TẬP 2: DEPTH-FIRST SEARCH:**

1. **Cách tạo cây**

* Cách thức tương tự như BFS, tiếp tục sử dụng cây phía trên để thực hiện ví dụ

1. **In ra cách di chuyển của thuật toán DFS (DFS traversal):**

* Để có thể in ra cách mà thuật toán BFS di chuyển ở cây trên, ta thực hiện gọi hàm **dfs(initialNode)** với:
  + initialNode là node khởi tạo để thuật toán BFS có thể thực hiện việc quét qua toàn bộ cây
  + Cú pháp: **g.dfs(‘A’)**

1. **Bảng tổng hợp các hàm:**

|  |  |
| --- | --- |
| **Tên hàm và cú pháp** | **Công dụng** |
| Graph(Num\_of\_Nodes) | Khởi tạo số nodes tồn tại trong cây |
| addEdge(parentNode,childNode) | Tạo nhánh cây mới |
| dfs(initialNode) | In ra cách thức di chuyển của thuật toán DFS từ node đã định sẵn trước (initialNode) |

**BÀI TẬP 3: TÌM KIẾM VỚI ĐỘ SÂU CHO TRƯỚC:**

1. **Cách tạo cây**

* Cách thức tương tự như BFS, tiếp tục sử dụng cây phía trên để thực hiện ví dụ

1. **Tùy chỉnh tham số:**

* Có 3 tham số có thể tùy chỉnh được ở chương trình là: **target, maxDepth, src (source)** 
  + **target** mục tiêu cần tìm kiêims
  + **maxDepth** độ sâu tối đa được phép tìm kiếm
  + **src (source)** bắt đầu tìm kiếm từ vị trí

1. **Bảng tổng hợp các hàm:**

|  |  |
| --- | --- |
| **Tên hàm và cú pháp** | **Công dụng** |
| Graph(Num\_of\_Nodes) | Khởi tạo số nodes tồn tại trong cây |
| addEdge(parentNode,childNode) | Tạo nhánh cây mới |
| IDDFS(src,target,maxDepth) | Thực hiện tìm kiếm **target** bắt đầu từ **src** với độ sâu maxDepth được cho phép. Hàm này trả về **True/False** |