How to implement depth-first search in Python

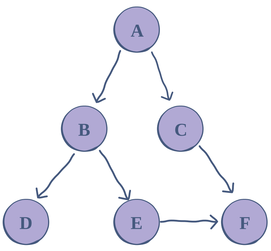
Depth-first search (DFS), is an algorithm for tree traversal on graph or tree data structures. It can be implemented easily using recursion and data structures like dictionaries and sets.

The Algorithm

1. Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes.
2. Repeat until all the nodes are visited, or the node to be searched is found.

Implementation

Consider this graph, implemented in the code below:



In [80]:
```
# Using a Python dictionary to act as an adjacency list
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Set to keep track of visited nodes.

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
dfs(visited, graph, 'A')
```

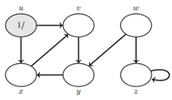
```
A
B
D
E
F
C
```

Explanation

- Lines 2-9: The illustrated graph is represented using an adjacency list - an easy way to do it in Python is to use a *dictionary* data structure. Each vertex has a list of its adjacent nodes stored.
- Line 11: `visited` is a set that is used to keep track of visited nodes.
- Line 21: The `dfs` function is called and is passed the `visited` set, the `graph` in the form of a dictionary, and `A`, which is the starting node.
- Lines 13-18: `dfs` follows the algorithm described above:
    1. It first checks if the current node is unvisited - if yes, it is appended in the `visited` set.
    2. Then for each neighbor of the current node, the `dfs` function is invoked again.
    3. The base case is invoked when all the nodes are visited. The function then returns.

> How Depth-First Search Works?

In this section, we will see visually the workflow of a depth-first search. Here is a graph and the source node is shown as the node u.



In [96]:
```
g = {
    'u': ['v', 'x'],
    'v': ['y'],
    'y': ['x'],
    'x': ['v'],
    'w': ['y', 'z'],
    'z': ['z']
    }
```

To keep track of the visited nodes, we will start with an empty list.

In [75]:
```
class depth_first:
    def __init__(self):
        self.visited = []
```

Now define a function that will loop through all the nodes and if there is an unvisited node, we will go in that node and find out where this node takes us.

In [76]:
```
def dfs(self, graph):
    for ver in graph:
        if ver not in self.visited:
            self.dfs_visit(graph, ver)
    return self.visited
```

Notice, in this function, we called a function 'dfs_visit'. This function is supposed to travel a whole unvisited route offered by an unvisited node and add those unvisited nodes to the 'visited' list. We will implement this function recursively.

Here is the 'dfs_visit' function:

In [77]:
```
def dfs_visit(self, graph, vertex):
    if vertex not in self.visited:
        self.visited.append(vertex)
        for nb in g[vertex]:
            self.dfs_visit(g, nb)
```

Have a careful look! This function will add a node if it is not already in the 'visited' list. Then it will go to one node adjacent to it and call itself.

That way, it will traverse the whole route that was unvisited before and one at a time.

Here is the complete code:

In [97]:
```
class depth_first:
    def __init__(self):
        self.visited = []

    def dfs(self, graph):
        for ver in graph:
            if ver not in self.visited:
                self.dfs_visit(graph, ver)
        return self.visited

    def dfs_visit(self, graph, vertex):
        if vertex not in self.visited:
            self.visited.append(vertex)
            for nb in g[vertex]:
                self.dfs_visit(g, nb)
```

In [98]:
```
d = depth_first()
print(d.dfs(g))
```

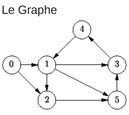
```
['u', 'v', 'y', 'x', 'w', 'z']
```

Resultat final :

In [ ]:
```
['u','v','y','x','w','z']
```

Mon exemple :

Le Graphe



Implementation du code :

In [99]:
```
S = {
    0: [1, 2],
    1: [2,5,3],
    2:[5],
    3:[4],
    4:[1],
    5:[3]
    }
```

In [100…
```
visited = set() # Set to keep track of visited nodes.

def dfs(visited, S, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in S[node]:
            dfs(visited,S, neighbour)

# Driver Code
dfs(visited, S, 1)
```

```
1
2
5
3
4
```

In [101…
```
class depth_first:
    def __init__(self):
        self.visited = []

    def dfs(self, S):
        for ver in S:
            if ver not in self.visited:
                self.dfs_visit(S, ver)
        return self.visited

    def dfs_visit(self, S, vertex):
        if vertex not in self.visited:
            self.visited.append(vertex)
            for nb in S[vertex]:
                self.dfs_visit(S, nb)
```

In [102…
```
m = depth_first()
print(m.dfs(S))
```

```
[0, 1, 2, 5, 3, 4]
```

Résultat obtenu:

In [ ]:
```
[0,1,2,5,3,4]
```