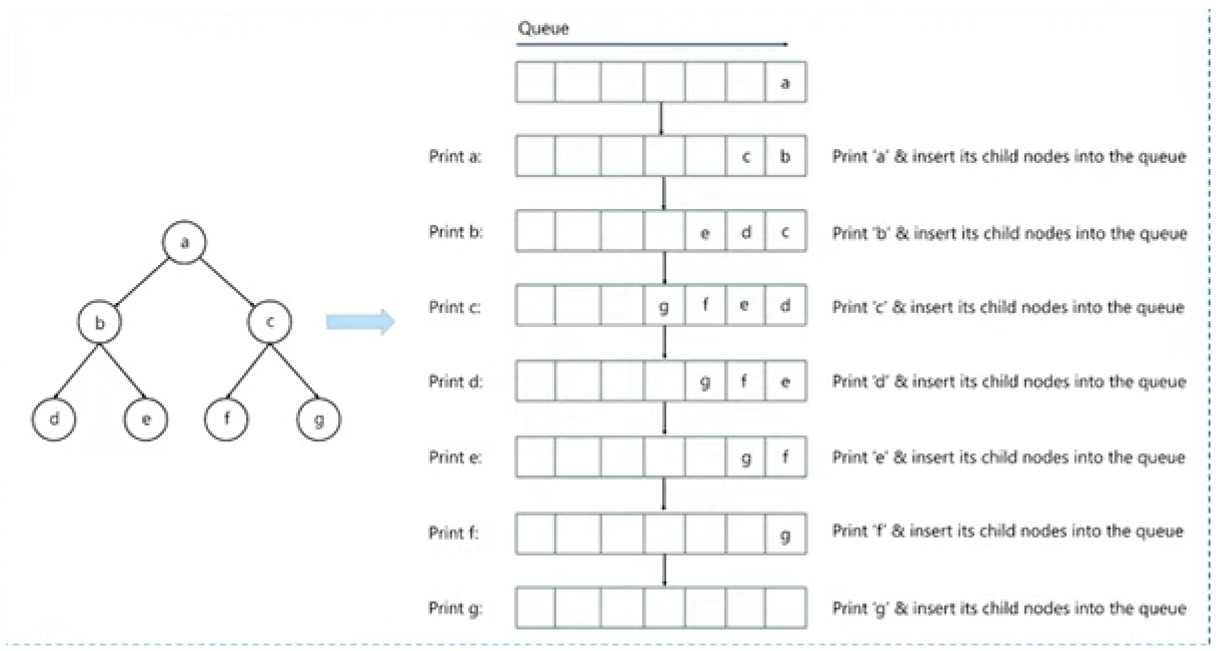
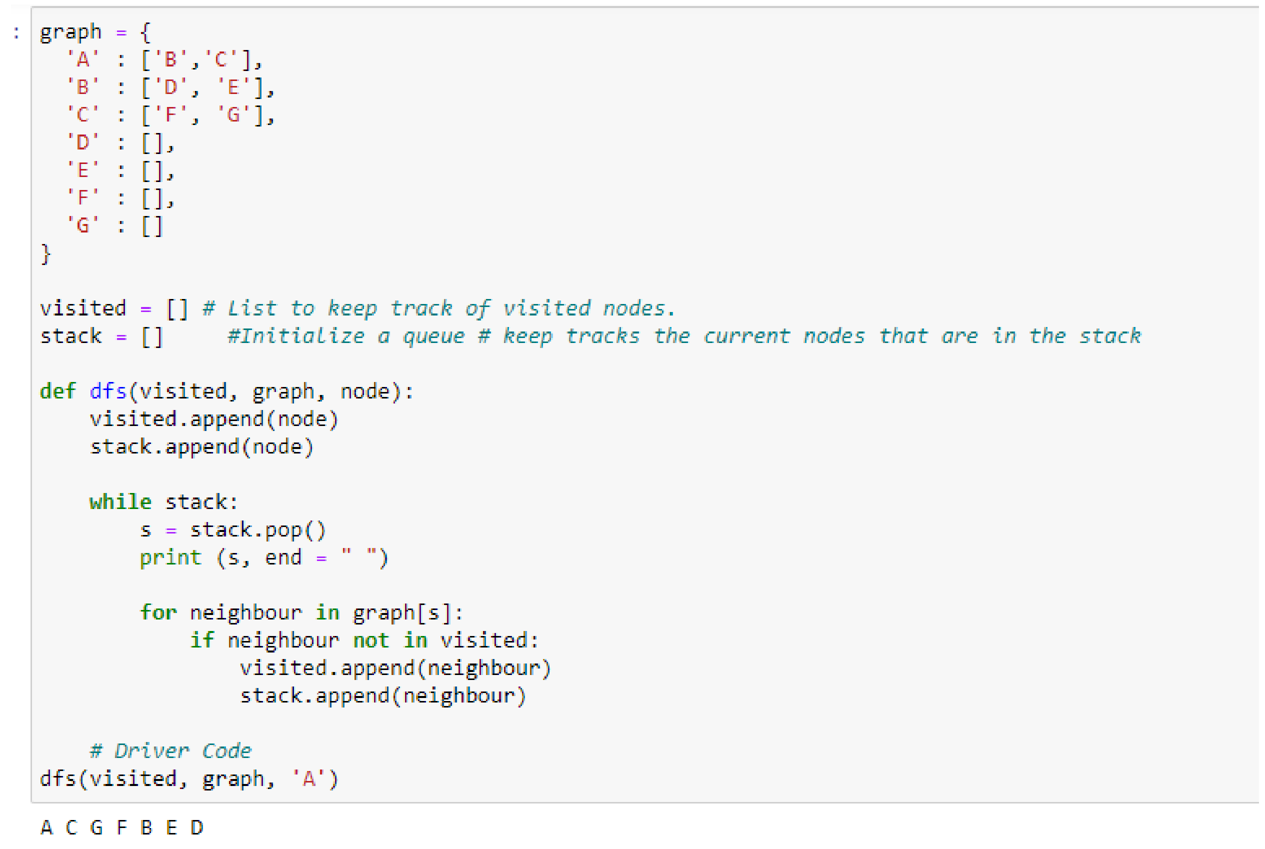
BFS uses a queue to keep track of the next location to visit. . **BFS is implemented using FIFO list.**


Queue
a
b
c
d
e
f
g
Print a:
Print b:
Print c:
Print d:
Print e:
Print f:
Print g:
Print 'a' & insert its child nodes into the queue
Print 'b' & insert its child nodes into the queue
Print 'c' & insert its child nodes into the queue
Print 'd' & insert its child nodes into the queue
Print 'e' & insert its child nodes into the queue
Print 'f' & insert its child nodes into the queue
Print 'g' & insert its child nodes into the queue

In [19]:
```python
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F', 'G'],
  'D' : [],
  'E' : [],
  'F' : [],
  'G' : []
}

visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue # keep tracks the current nodes that are in the queue

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

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

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

DFS: Unlike Bfs, DFS uses Lifo method and uses STACK to keep track of nodes.



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

visited = [] # List to keep track of visited nodes.
stack = []     #Initialize a queue # keep tracks the current nodes that are in the stack

def dfs(visited, graph, node):
    visited.append(node)
    stack.append(node)

    while stack:
        s = stack.pop()
        print (s, end = " ")

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                stack.append(neighbour)

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

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

Practice : Given a gold mine of n*m dimensions. Each field in this mine contains a positive integer which is the amount of gold in tons. Initially the miner is at first column but can be at any row. He can move only (right->,right up /,right down\) that is from a given cell, the miner can move to the cell diagonally up towards the right or right or diagonally down towards the right. Find out maximum amount of gold he can collect.

Examples:

```
Input : mat[][] = {{1, 3, 3},
                   {2, 1, 4},
                  {0, 6, 4}};
Output : 12
{(1,0)->(2,1)->(1,2)}

Input: mat[][] = { {1, 3, 1, 5},
                   {2, 2, 4, 1},
                {5, 0, 2, 3},
                {0, 6, 1, 2}};
Output : 16
(2,0) -> (1,1) -> (1,2) -> (0,3) OR
(2,0) -> (3,1) -> (2,2) -> (2,3)

Input : mat[][] = {{10, 33, 13, 15},
               {22, 21, 04, 1},
               {5, 0, 2, 3},
               {0, 6, 14, 2}};
Output : 83
```