Aim: Write a program to implement the following searching techniques: BFS and DFS.

Code: BFS

```
graph = {'A': ['B','C'],
         'B': ['A','D','E'],
         'C': ['F','G','A'],
         'D': ['B'],
         'E': ['H','B'],
         'F': ['C'],
         'G': ['C'],
         'H': ['E']
        }
def bfs(graph, root):
    visited, queue = set([root]), collections.deque([root])
    while queue:
        vertex = queue.popleft()
        visit(vertex)
        for node in graph[vertex]:
            if node not in visited:
                visited.add(node)
                queue.append(node)

def visit(n): print(n)
bfs(graph, 'A')
```

DFS

```
graph = {'A': ['B','C'],
         'B': ['A','D','E'],
         'C': ['F','G','A'],
         'D': ['B'],
         'E': ['H','B'],
         'F': ['C'],
         'G': ['C'],
         'H': ['E']
        }
def dfs(graph,start,end,route,list):
    route+=[start]
    if start == end:
        list.extend(route)
    else:
        for node in graph[start]:
            if node not in route:
                dfs(graph,node,end,route,list)
def dfs_route(graph,start,end):
     list = []
     dfs(graph,start,end,[],list)
     return list
print(dfs_route(graph,'A','G'))
```

Output:

In [29]:
```
graph = {'A': ['B','C'],
         'B': ['A','D','E'],
         'C': ['F','G','A'],
         'D': ['B'],
         'E': ['H','B'],
         'F': ['C'],
         'G': ['C'],
         'H': ['E']
        }
def bfs(graph, root):
    visited, queue = set([root]), collections.deque([root])
    while queue:
        vertex = queue.popleft()
        visit(vertex)
        for node in graph[vertex]:
            if node not in visited:
                visited.add(node)
                queue.append(node)

def visit(n): print(n)
bfs(graph, 'A')
```

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

In [17]:
```
graph = {'A': ['B','C'],
         'B': ['A','D','E'],
         'C': ['F','G','A'],
         'D': ['B'],
         'E': ['H','B'],
         'F': ['C'],
         'G': ['C'],
         'H': ['E']
        }

def dfs(graph,start,end,route,list):
    route+=[start]
    if start == end:
        list.extend(route)
    else:
        for node in graph[start]:
            if node not in route:
                dfs(graph,node,end,route,list)

def dfs_route(graph,start,end):
      list = []
      dfs(graph,start,end,[],list)
      return list

print(dfs_route(graph,'A','G'))
```

```
['A', 'B', 'D', 'E', 'H', 'C', 'F', 'G']
```