DAY-15

Graph traversal

Breadth First Search (BFS).

-> An recursive algo.

Algoritham:-

-> puts each vertex of the graph into visited or Not visited category.

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2&3 until the queue is empty.

Psaudocode:-

```
Createa a Queue Q.
mark v as visited andput v into Q
while Q is non-empty
    remove the head u of Q
    mark and enqueue all (unvisited)
        neighbors of u.
```

```
import coilections

def bfs(graph, root):
    visited = set()
    queue = collections.deque([root])
    visitted.add(root)

    while queue:
        vertex = queue.popleft()
        print(str(vertex)+" ", end="")

    # if not visited, mark it as visited
    for neighbor in graph[vertex]:
        if neighbor not in visited:
            visited.add(neighbor)
            queue.append(neighbor)
```

Depth First Search Algo.

* same as BFS but will use 'stack' instead of queue.

1. Start by putting any one of the graph's vertices ontop of ~~the~~ a "stack".

2. Take the top item of the stack and add it to the visited list.

3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.

4. keep repeating steps 2 & 3 untill the stack is empty.

~~for each v ∈ G. Adj [u]~~
~~if v.visited == false~~
~~DFS(G.u)~~
~~for~~

code:

```
def dfs(graph ,start ,visited=None):
    if visited is None:
            visited = set()
        visited .add (start'

        print (start)

        for next in graph[start]-visited:
            dfs(graph, next, visited)
        return visited.
```

* Any two nodes connected by an edge are said to be "adjacent"