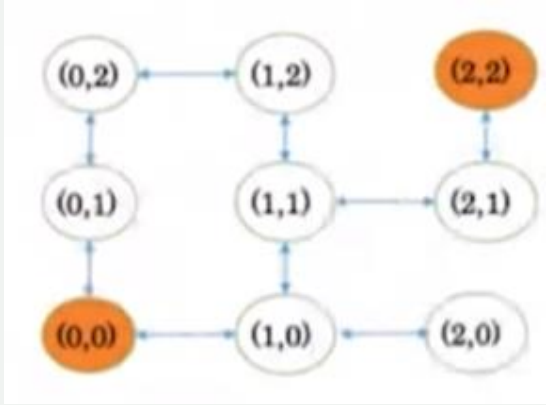
Search Algorithms

Dr. Venkataramana Veeramsetty

List of search algorithms

Random
Search Algorithm: Example



1. Current Node X = Initial Node
2. If X=target node, stop with success
3. Expand X, and get set S of child nodes
4. Select a node X' from set S Randomly
5. Set X=X' go to step 2

#	X	Success Status	S	X'
1	0,0	No	(0, 1), (1,0)	0, 1
2	0,1	No	(0, 2), (0, 0)	0, 2
3	0, 2	No	(1, 2), (0, 1)	1, 2
4	1, 2	No	(1, 1), (1, 2)	1, 1
5	1, 1	No	(2 ,1), (1, 2), (1,0)	2 ,1
6	2, 1	No	(2, 2), (1, 1)	2, 2
7	2, 2	Yes		

Depth First Search Algorithm

A standard DFS implementation puts each vertex of the graph into one of two categories

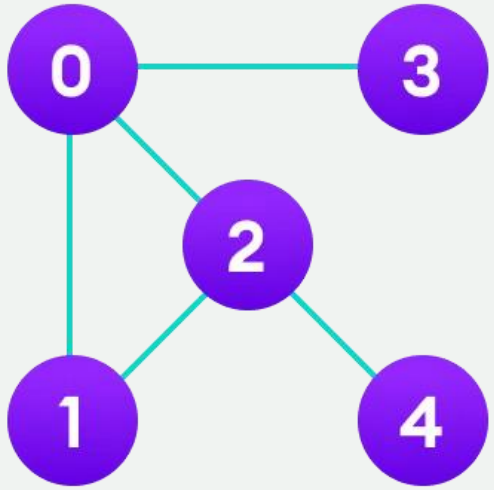
- Visited
- Not Visited

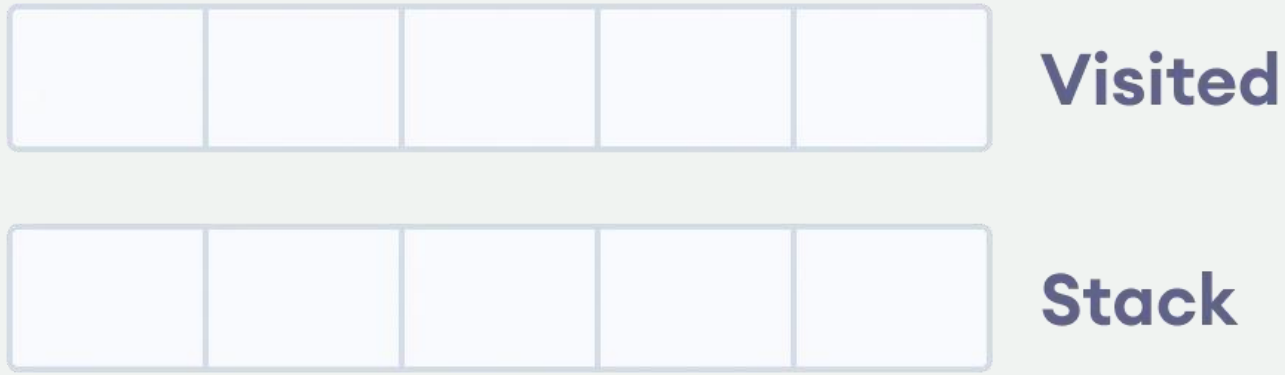
Algorithm

1. Start by putting any one of the graph's vertices on top of 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 and 3 until the stack is empty.

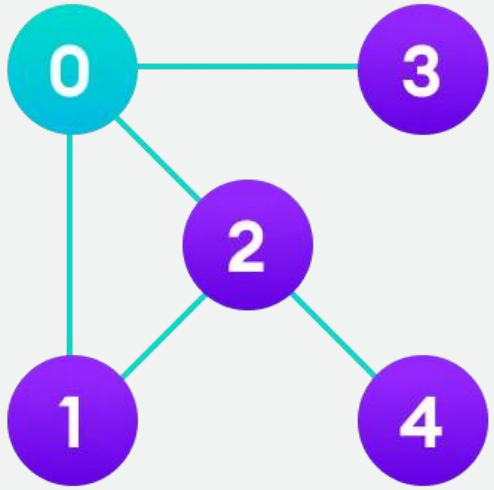
Example

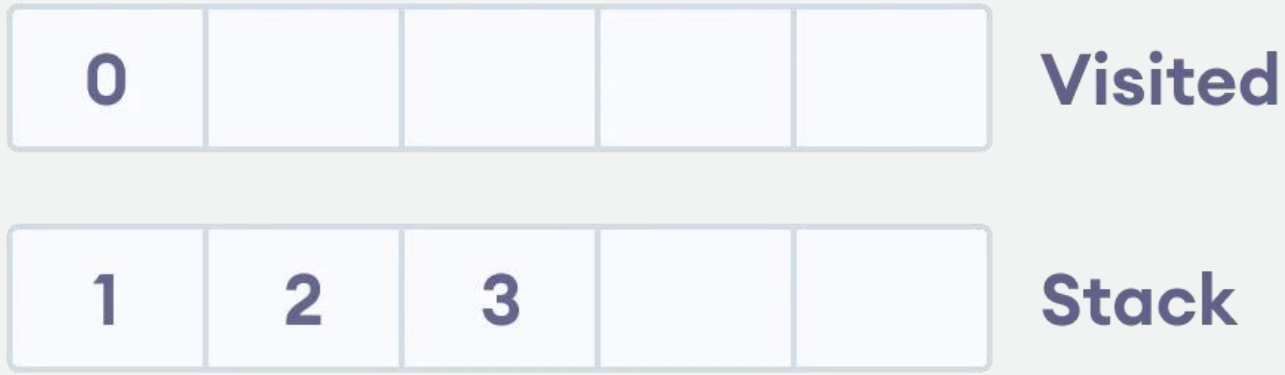
Let's see how the Depth First Search algorithm works with an example.
We use an undirected graph with 5 vertices.



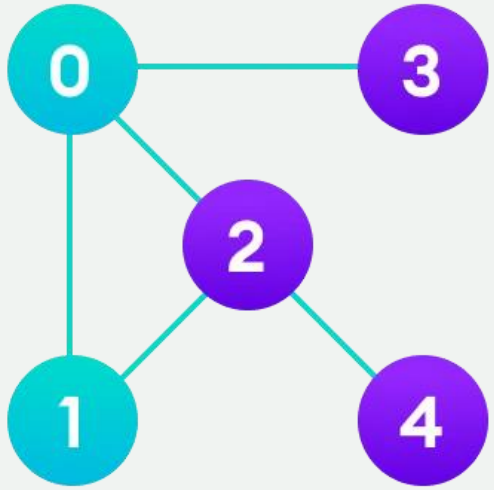


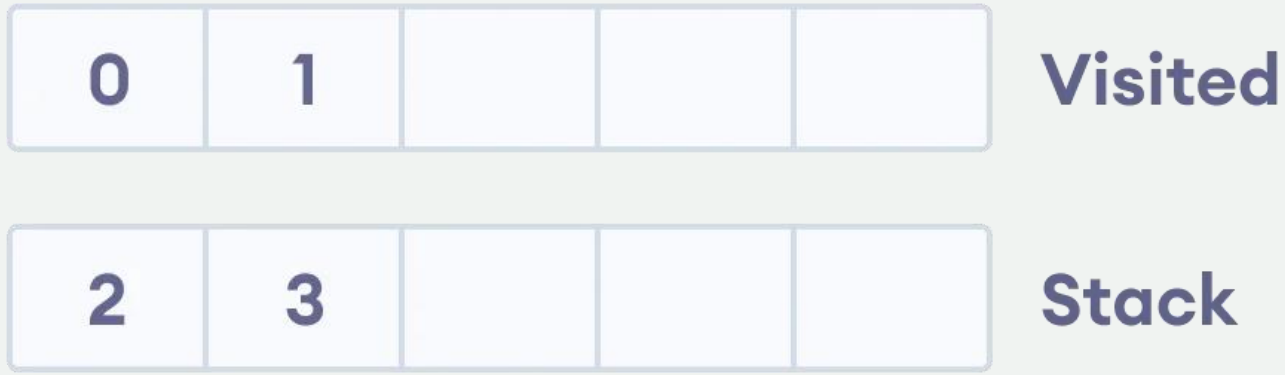
We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



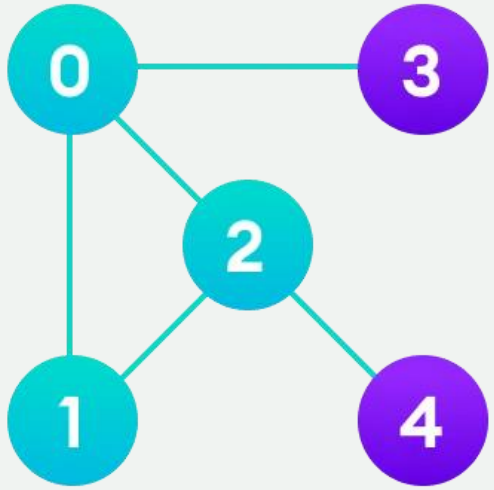


Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead



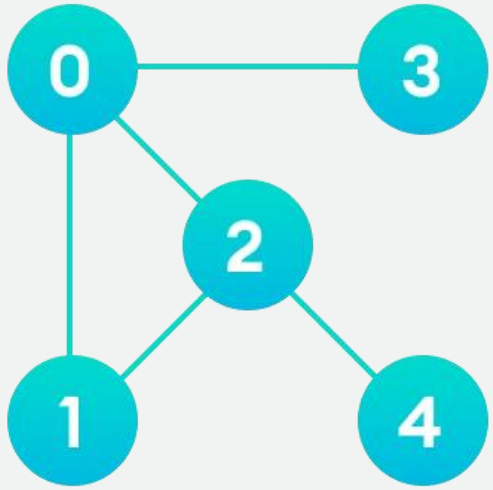


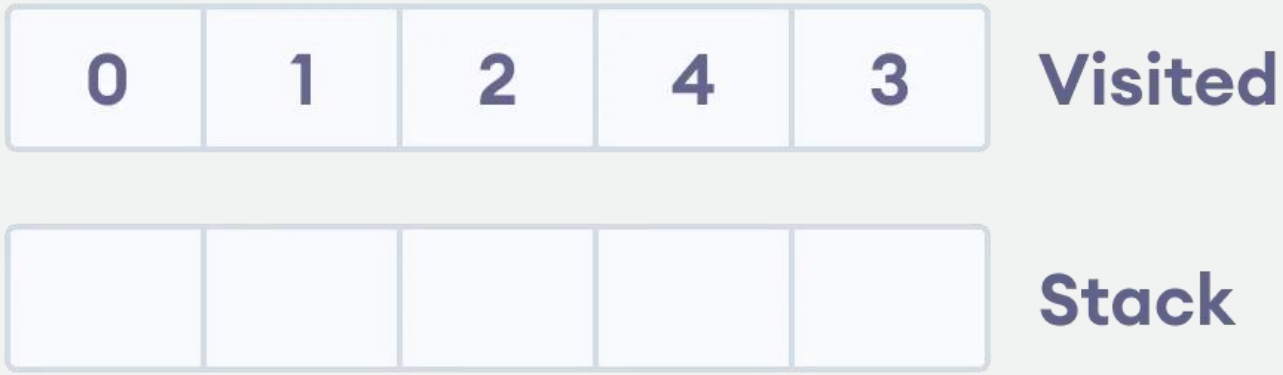
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



0	1	2			Visited
4	3				Stack

After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.





Applications

FOR FINDING THE PATH

TO TEST IF THE GRAPH IS BIPARTITE

FOR DETECTING CYCLES IN A GRAPH

Python Implementation

```python
# DFS algorithm in Python


# DFS algorithm
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


graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])}

dfs(graph, '0')
```

```
0
1
4
3
2
3
2
{'0', '1', '2', '3', '4'}
```

Breadth first search

A standard BFS implementation puts each vertex of the graph into one of two categories:

- Visited
- Not Visited

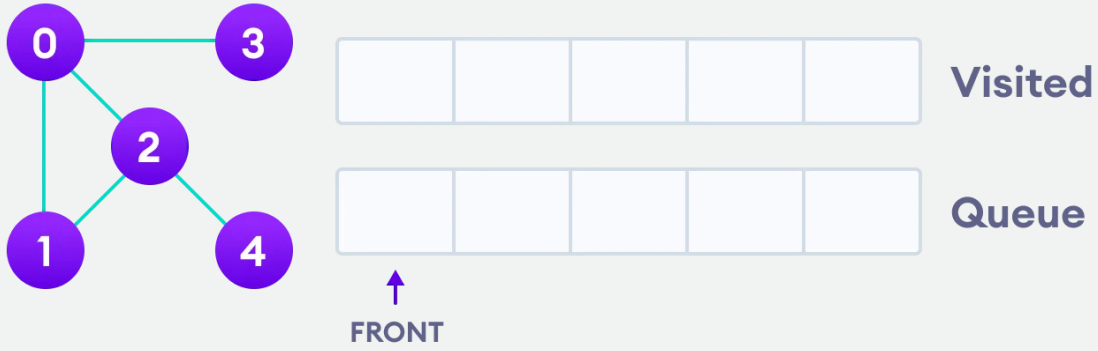
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The algorithm works as follows:

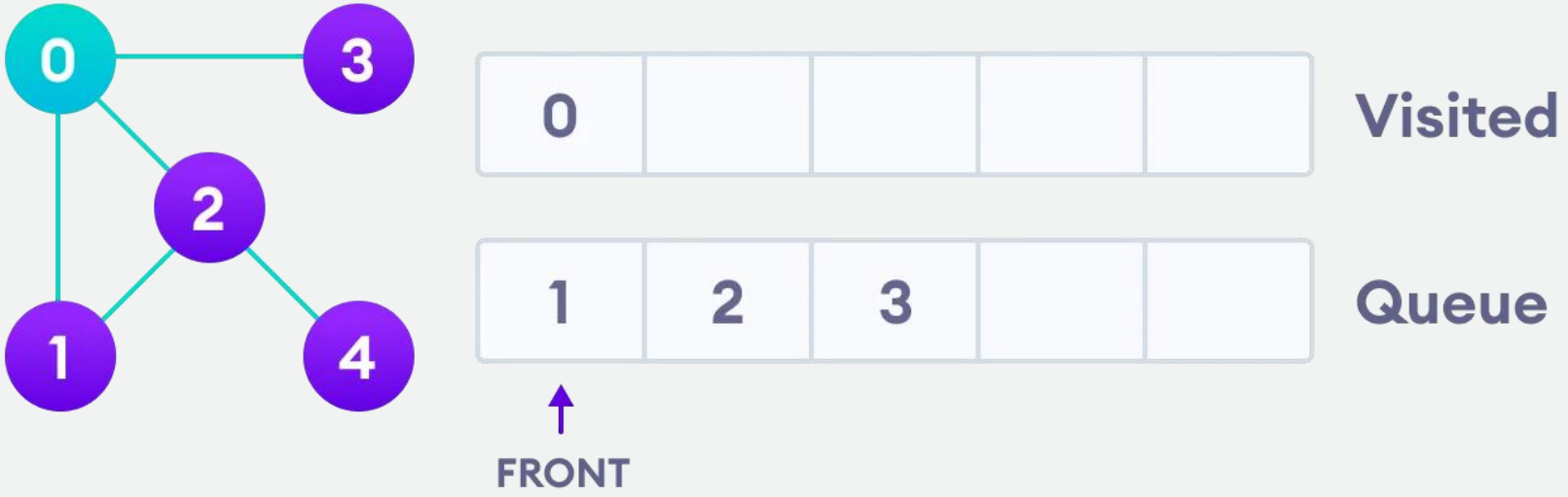
1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until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

BFS example

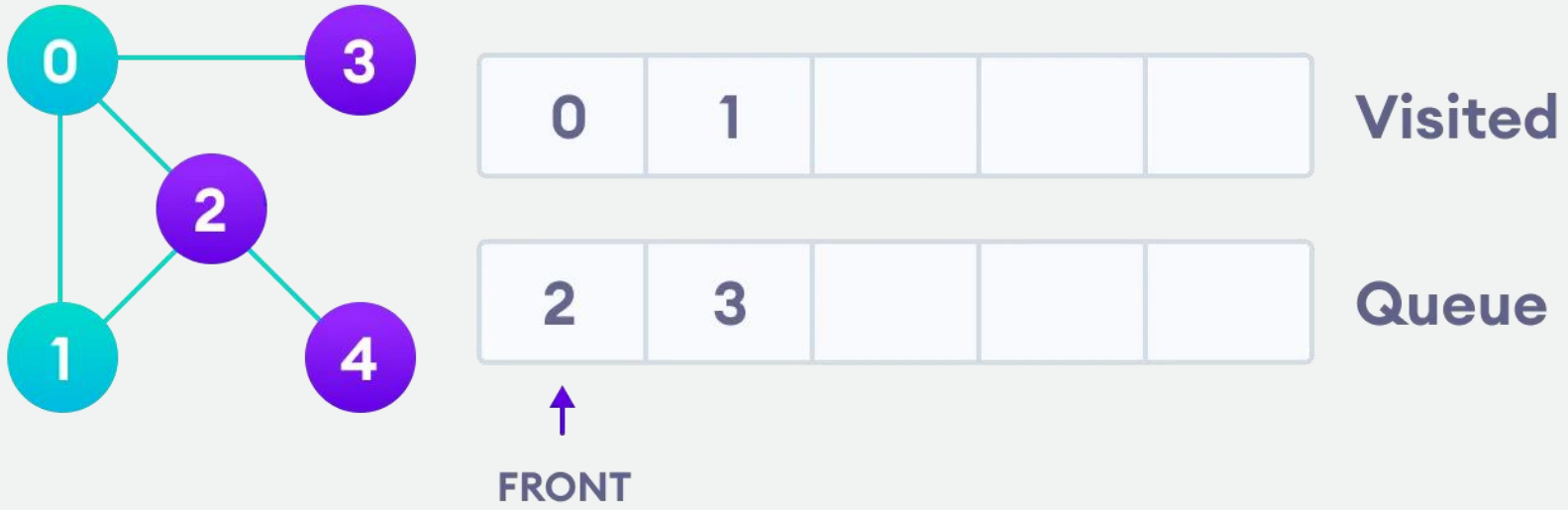
Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.

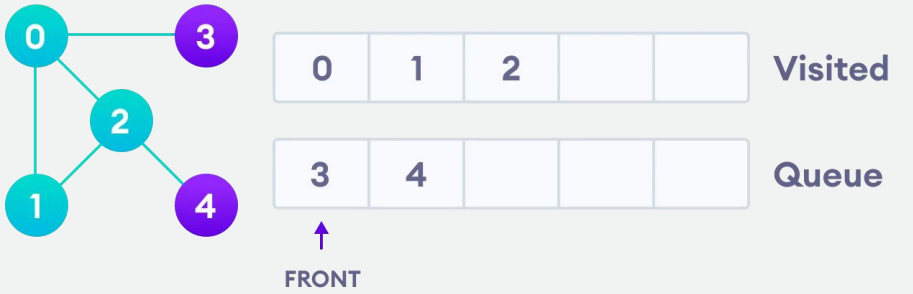


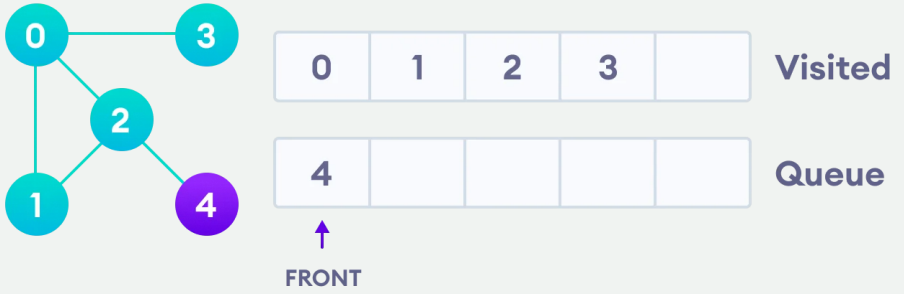
Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.



Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.

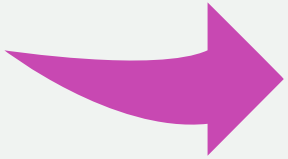
Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.





Applications

Path finding algorithms



Cycle detection in an undirected graph

For GPS navigation

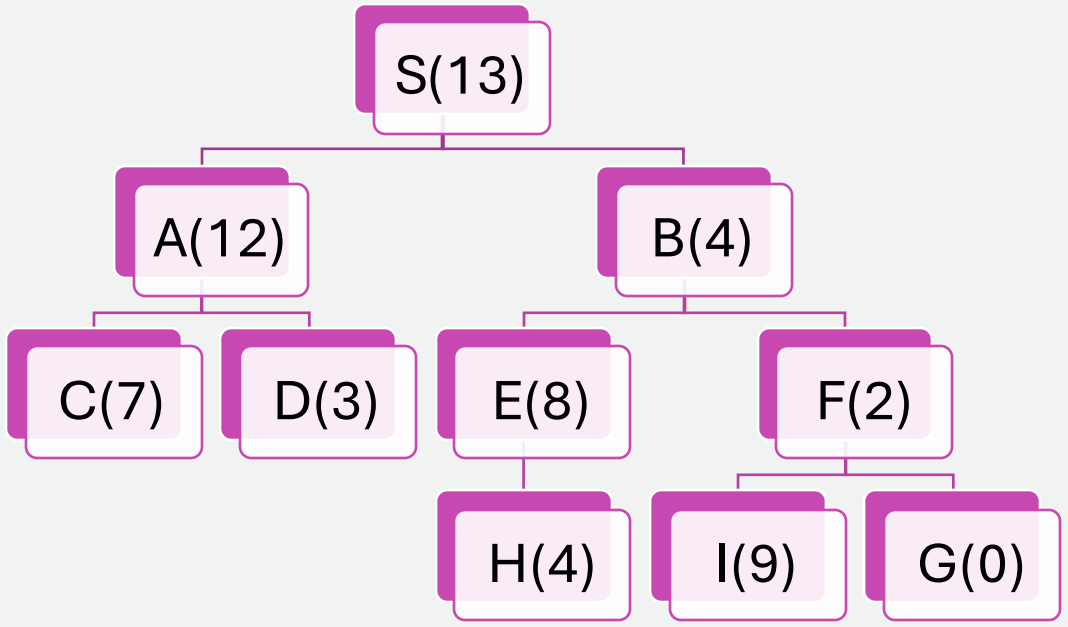
Best First Search Algorithm

- Create 2 empty lists: OPEN and CLOSED
- Start from the initial node (say N) and put it in the ‘ordered’ OPEN list

Repeat the next steps until the GOAL node is reached

1. If the OPEN list is empty, then EXIT the loop returning ‘False’
2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning ‘True’. The solution can be found by backtracking the path
4. If N is not the GOAL node, expand node N to generate the ‘immediate’ next nodes linked to node N and add all those to the OPEN list
5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

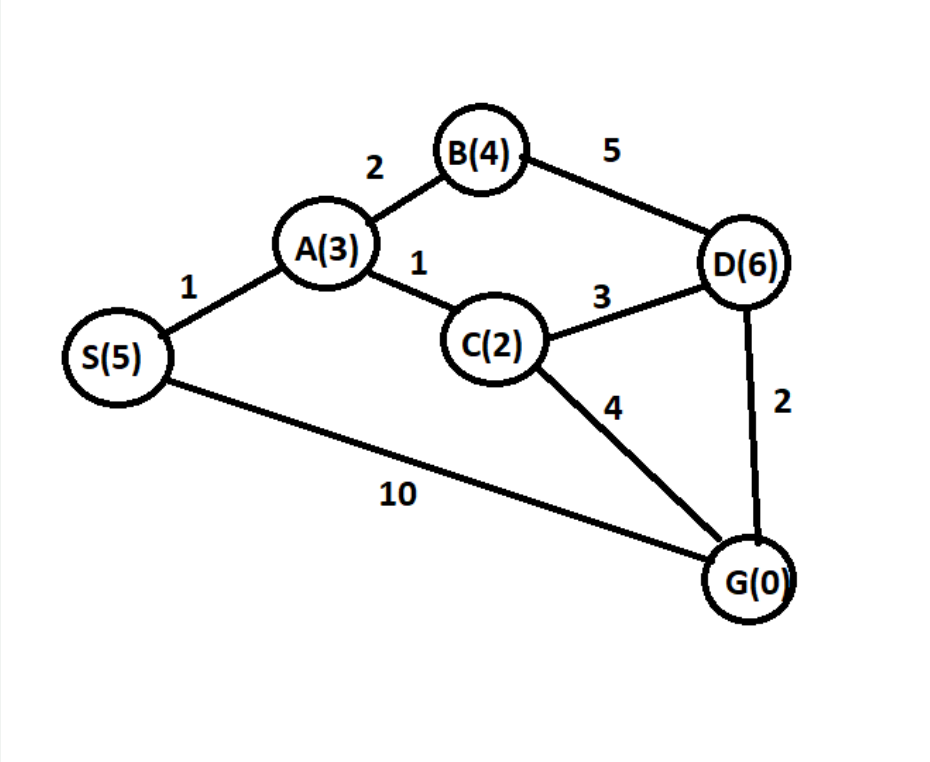
Example



Step	Open	Closed
Initialization	[A,B]	[S]
Iteration 1	[A,E,F]	[S,B]
Iteration 2	[A,E,I,G]	[S,B,F]
Iteration 3	[A,E,I]	[S,B,F,G]

Path: S-B-F-G

A* Algorithm



A* algorithm finds the shortest path through the search space using heuristic function

Step 1:.

S-A : F(n)=g(n)+h(n) = 1+3=4

S-G : F(n)=g(n)+h(n) = 10+0=10 Hold

Step 2:

S-A-C: F(n)=g(n)+h(n) = 2+2=4

S-A-B: F(n)=g(n)+h(n) = 3+4=7 Hold

Step 3:

S-A-C – G : F(n)=g(n)+h(n) = 6+0 =6 (Best Path)

S-A-C-D: F(n)=g(n)+h(n) = 5+6=11 Hold

Thank You