3-2 기본 서치 알고리즘

이전 강의 요약

01

그래프

그래프

방향성 그래프

02

트리

디시전 트리

03

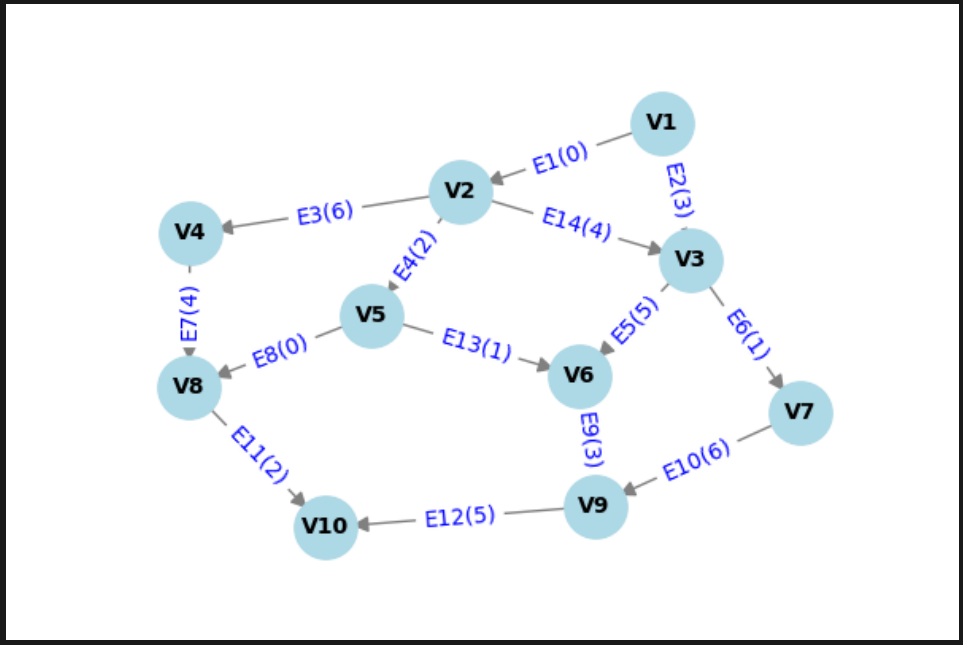
서치 알고리즘의 필요성

04

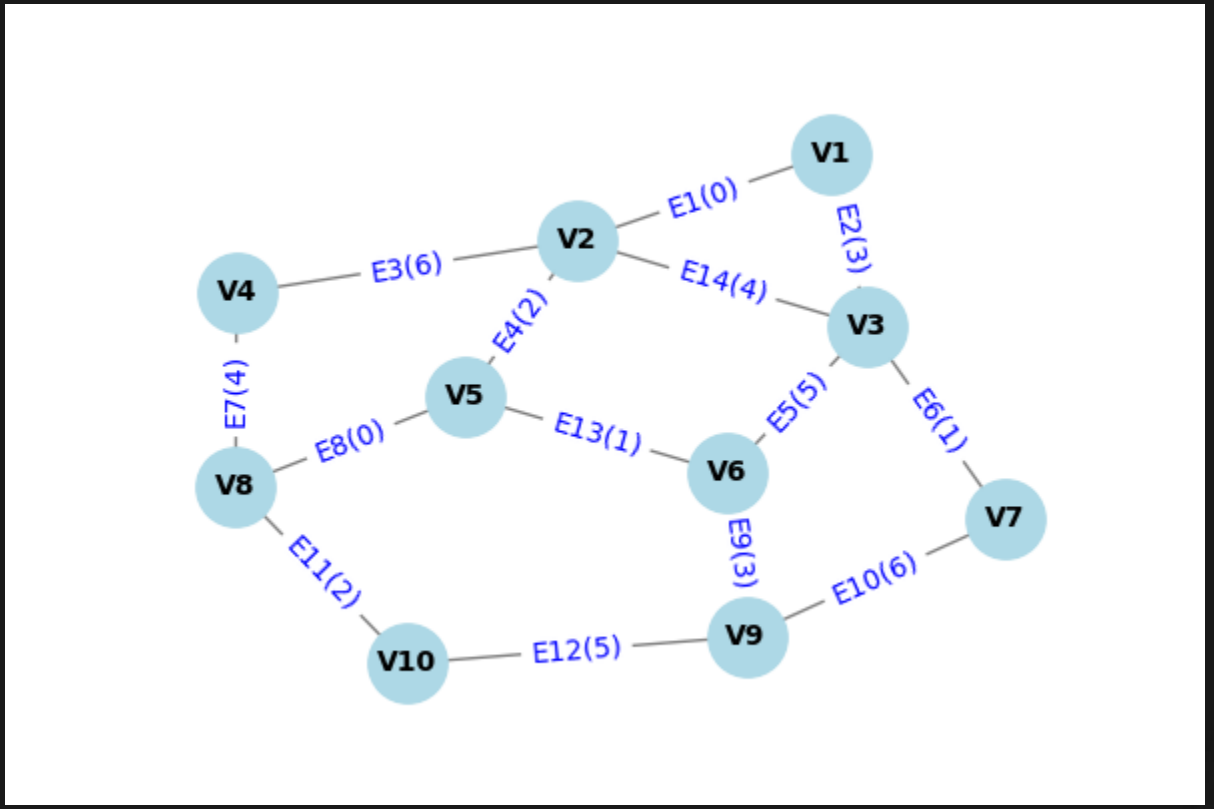
그래프와 트리의 코드 구조

그래프 탐색 알고리즘의 필요성

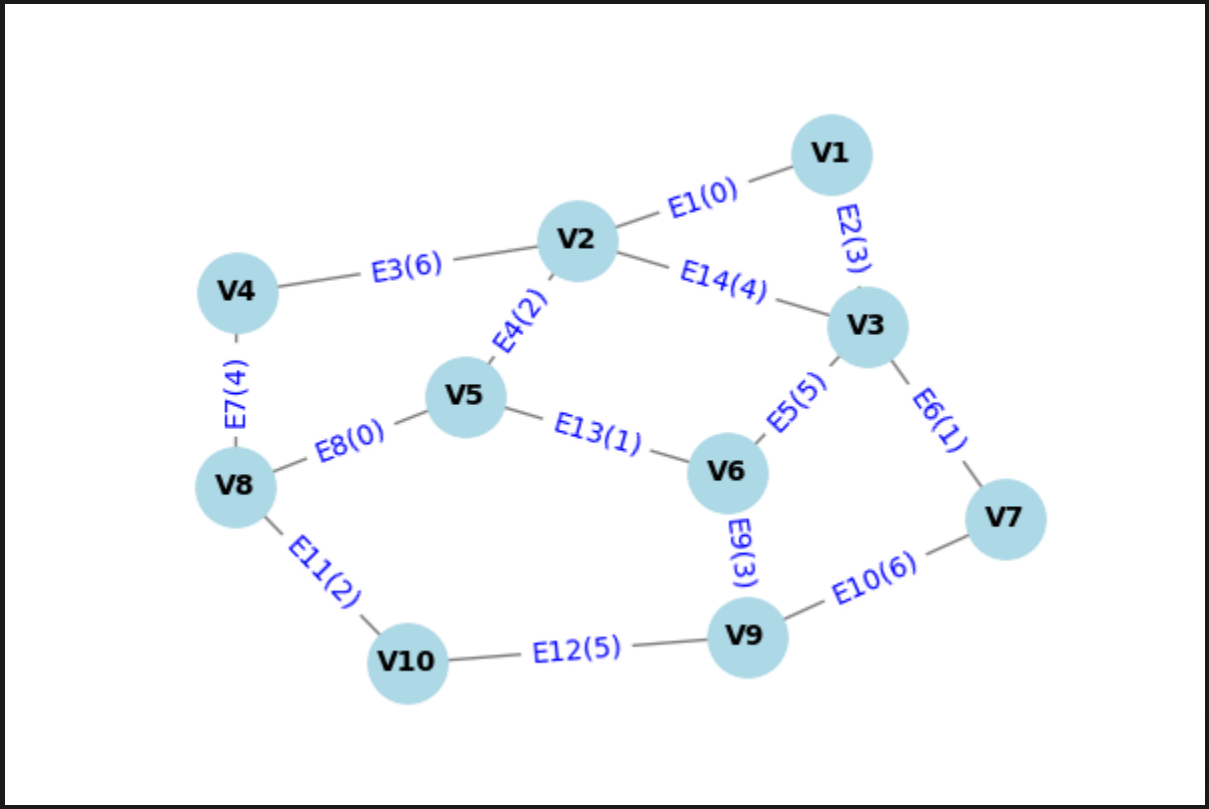
테스크 → 데이터 구조로 문제를 정의 (그래프) → 경로 탐색 → 경로



탐색의 기초 설명

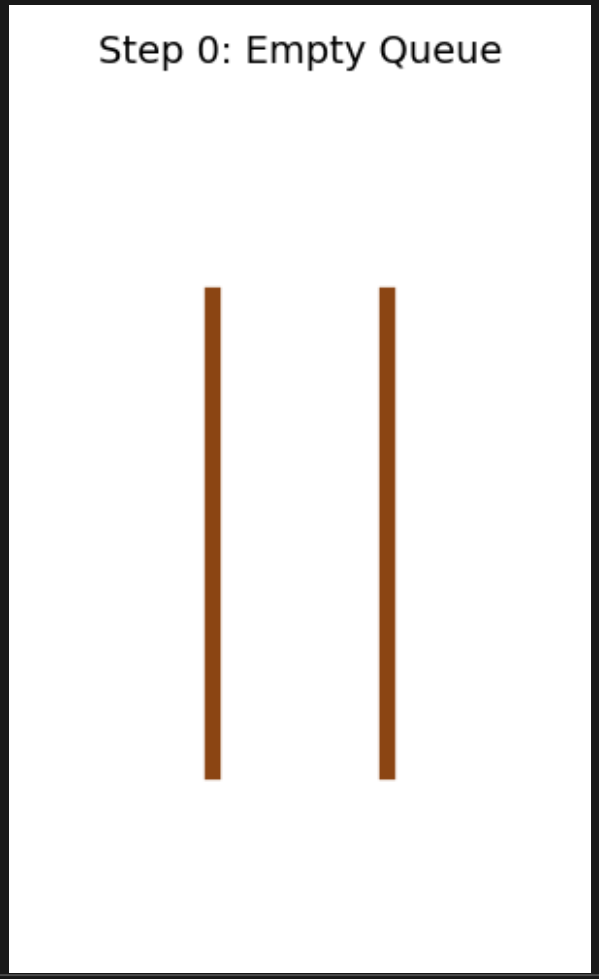

V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
E1(0)
E2(3)
E3(6)
E4(2)
E14(4)
E7(4)
E8(0)
E13(1)
E5(5)
E6(1)
E11(2)
E9(3)
E10(6)
E12(5)

Breadth First Search (BFS)



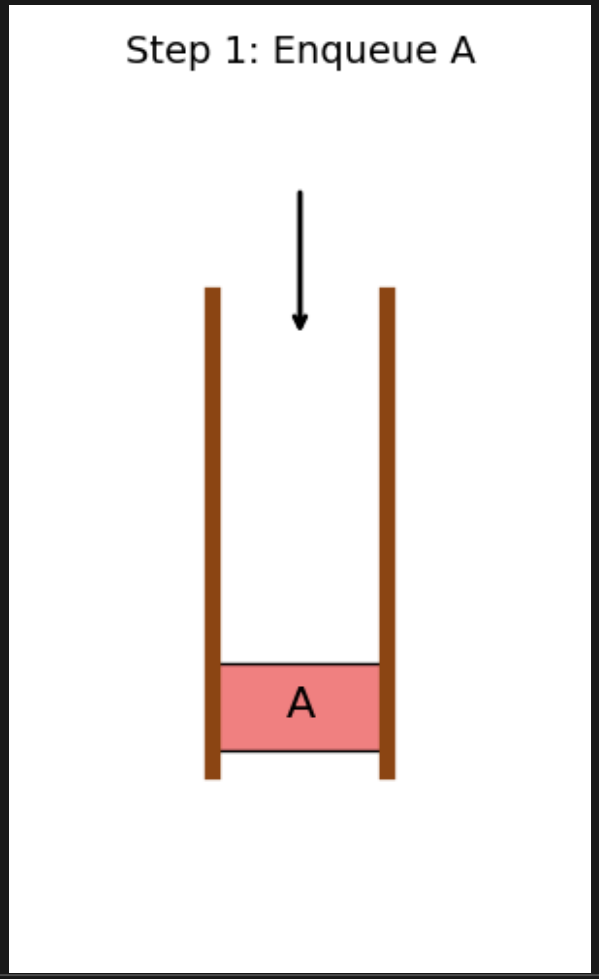
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



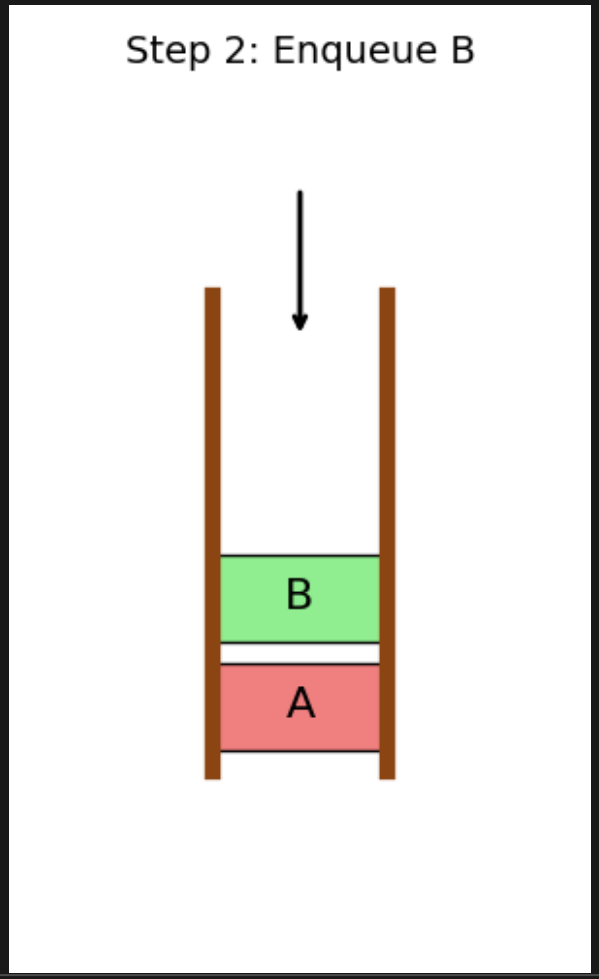
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



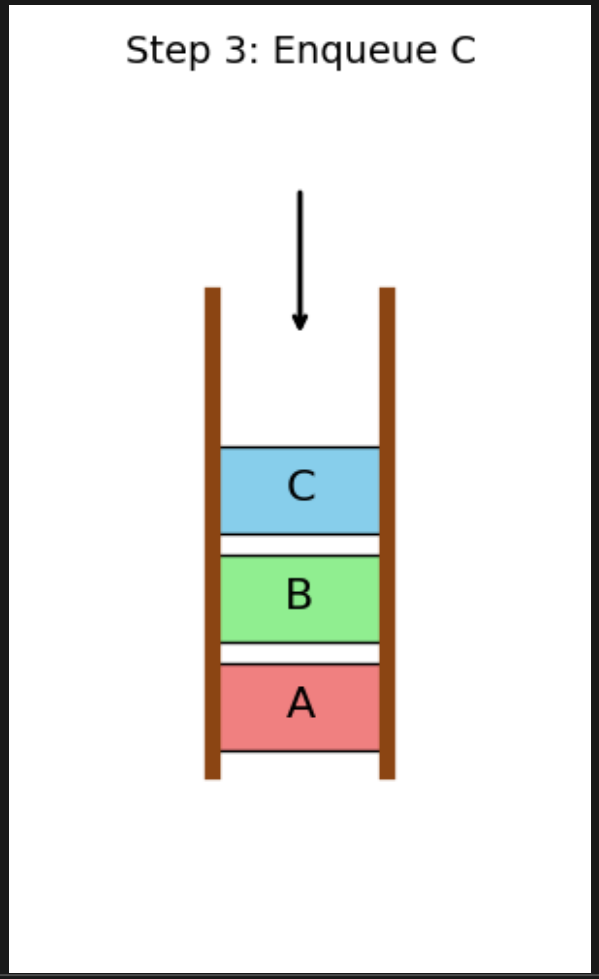
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



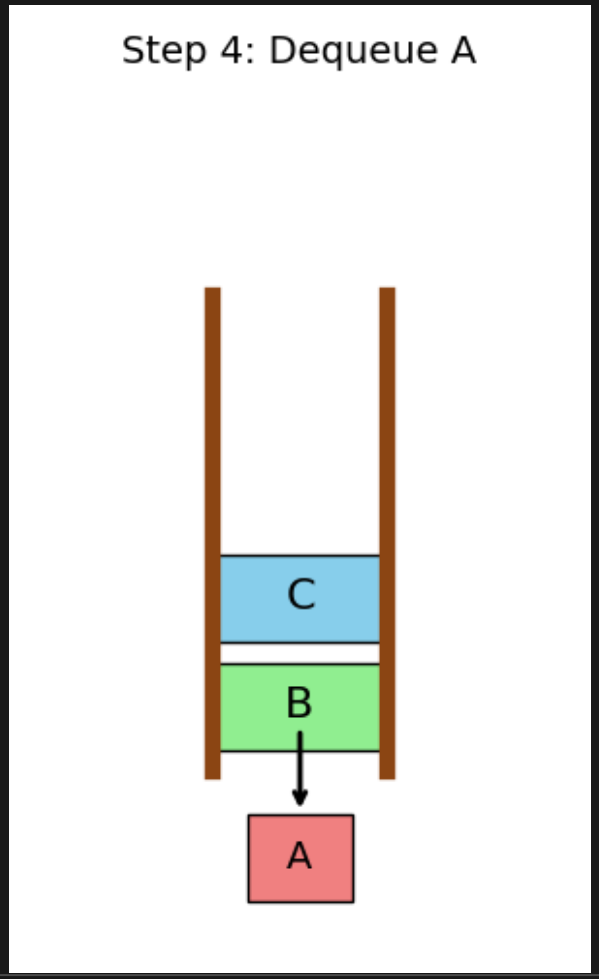
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



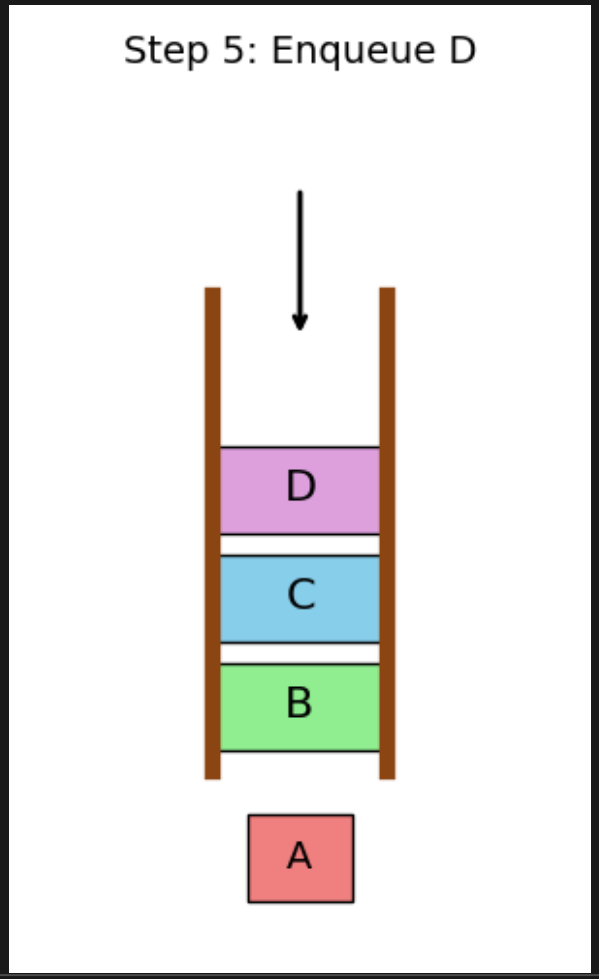
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



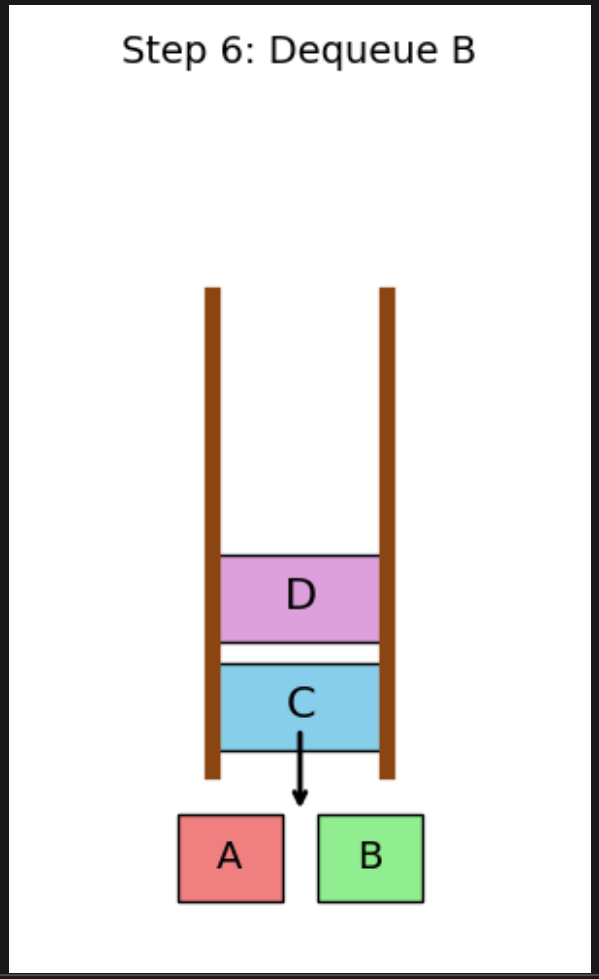
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



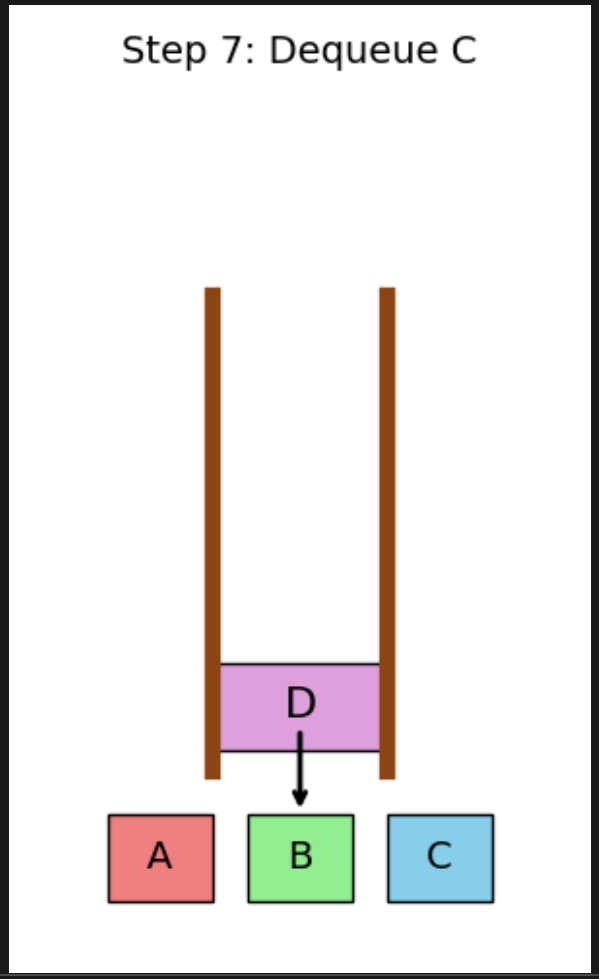
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



Breadth First Search (BFS)

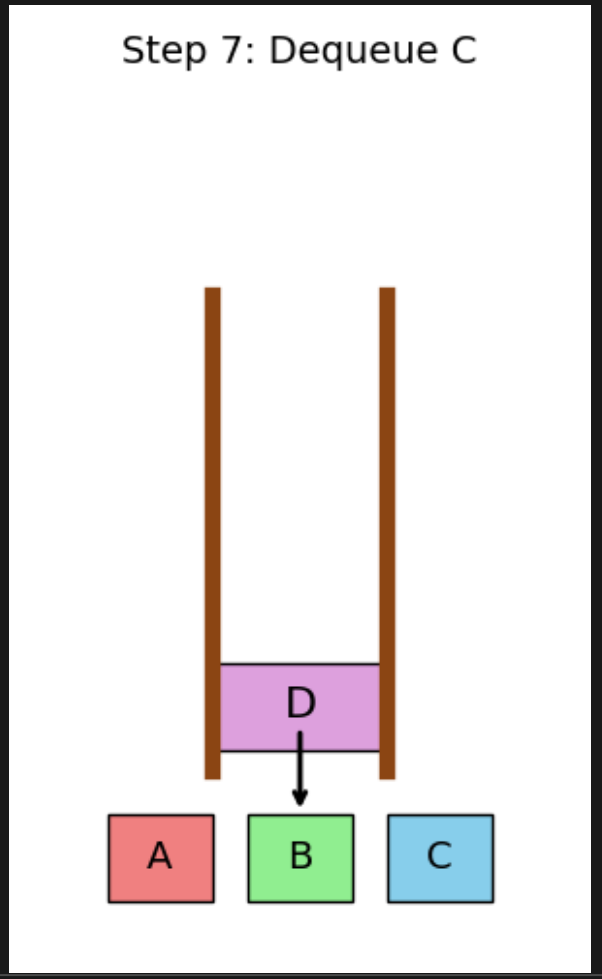
Queue: First In First Out (FIFO)



Breadth First Search (BFS)

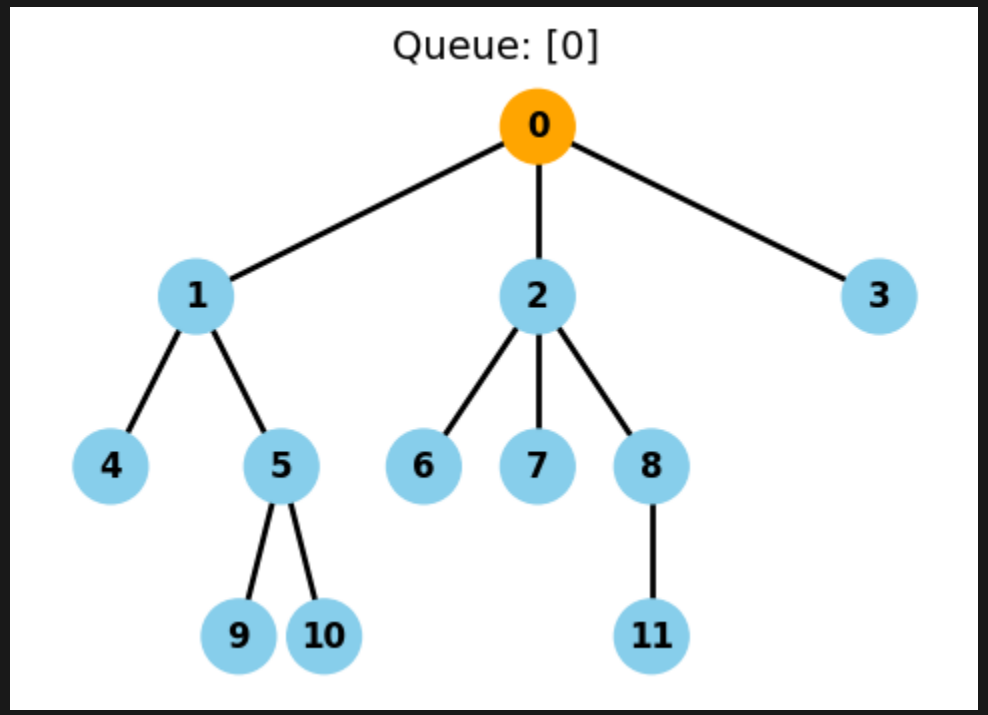
Queue: First In First Out (FIFO)

```
from collections import deque

queue = deque()
queue.append("A")
queue.appendleft("B")
queue.pop()
queue.popleft()
```



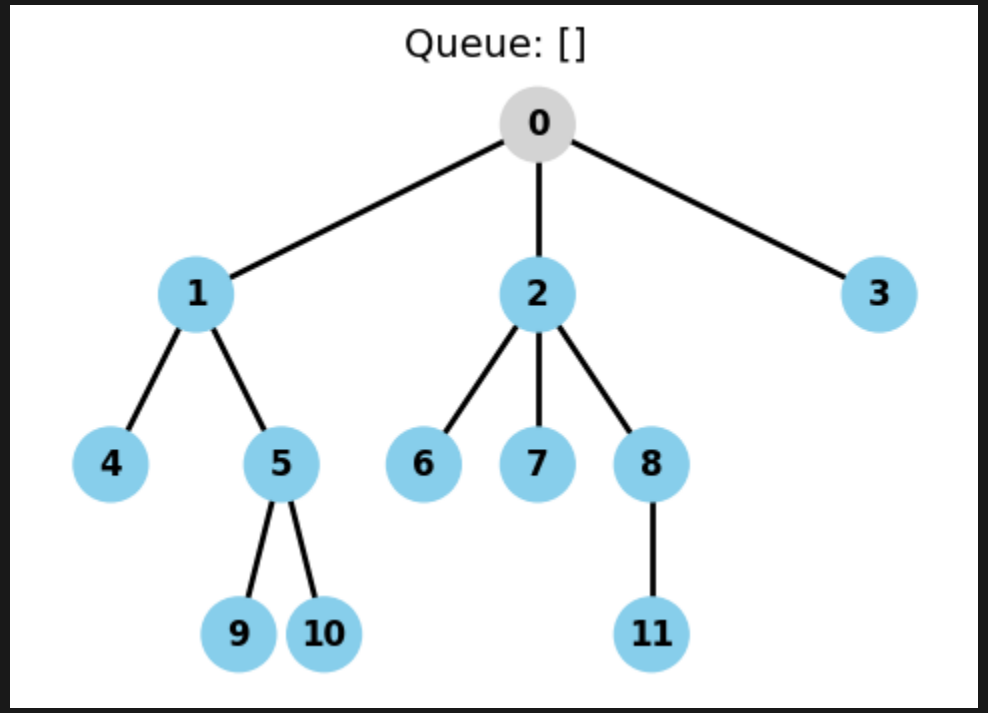
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



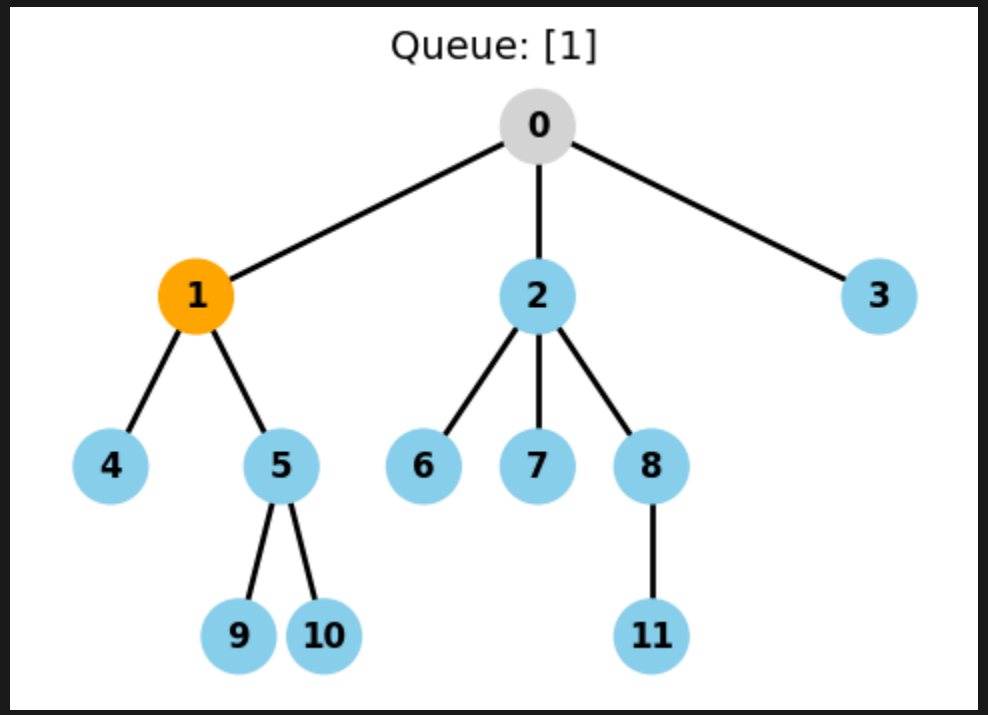
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



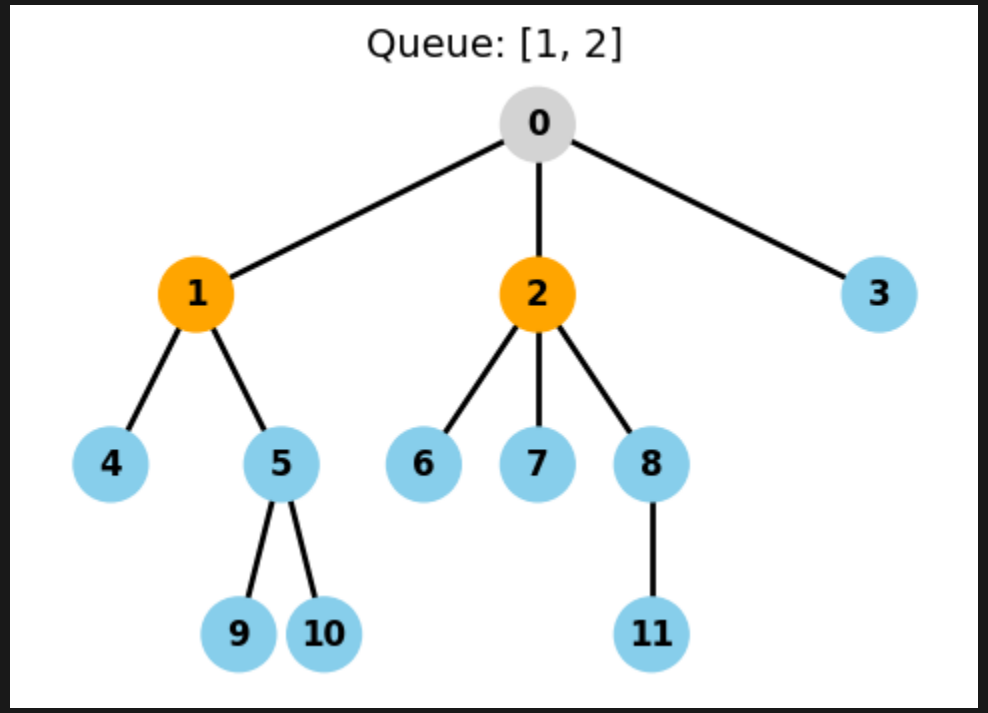
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



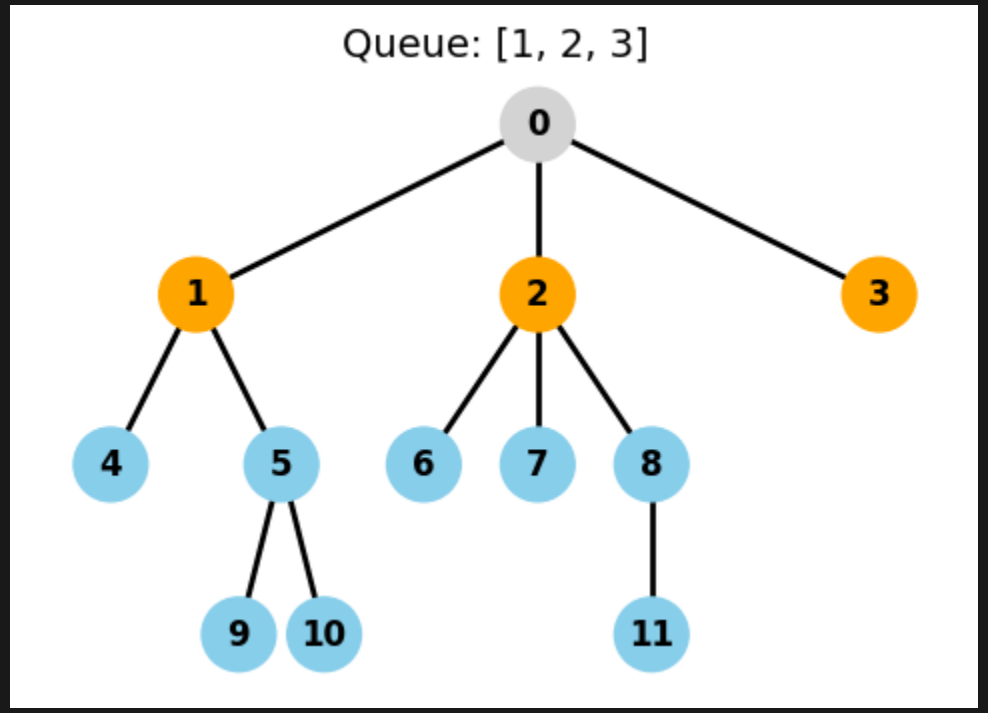
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



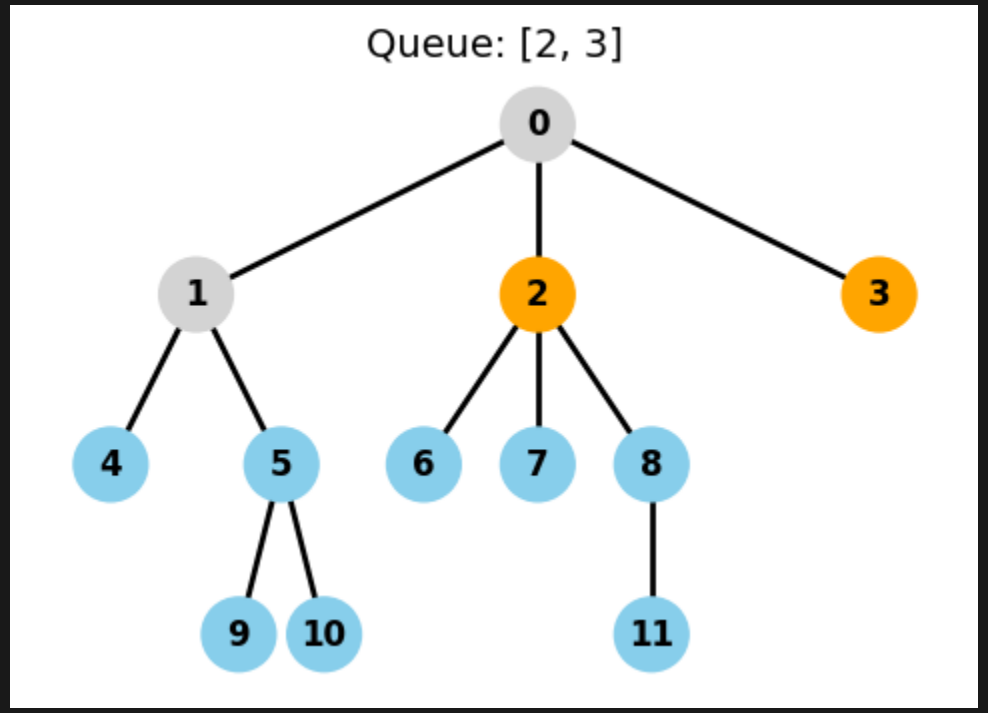
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



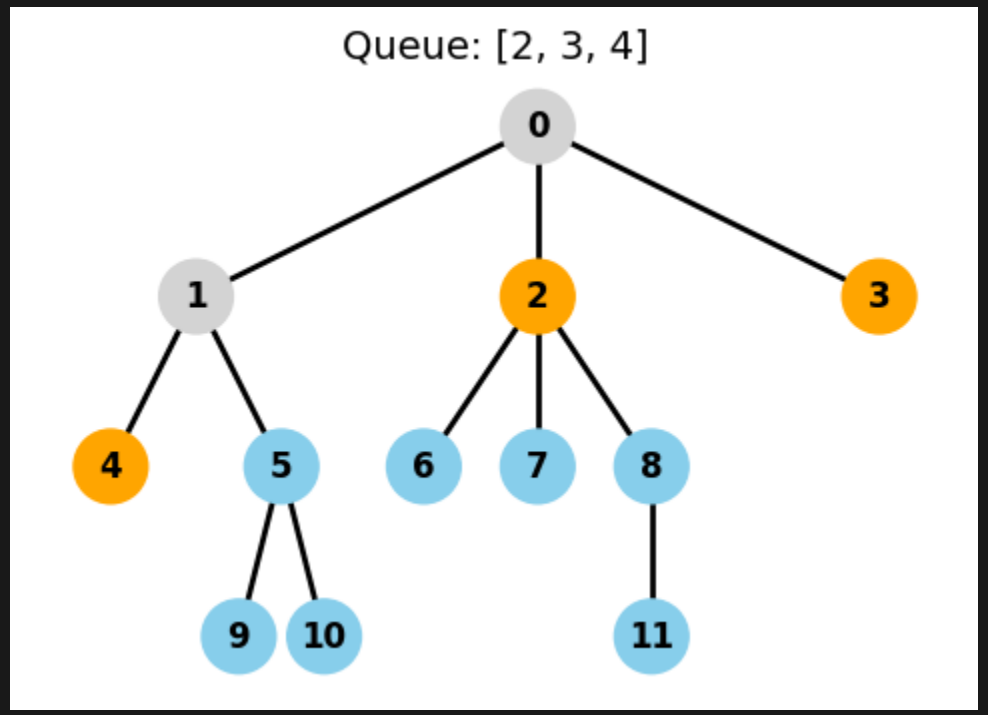
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



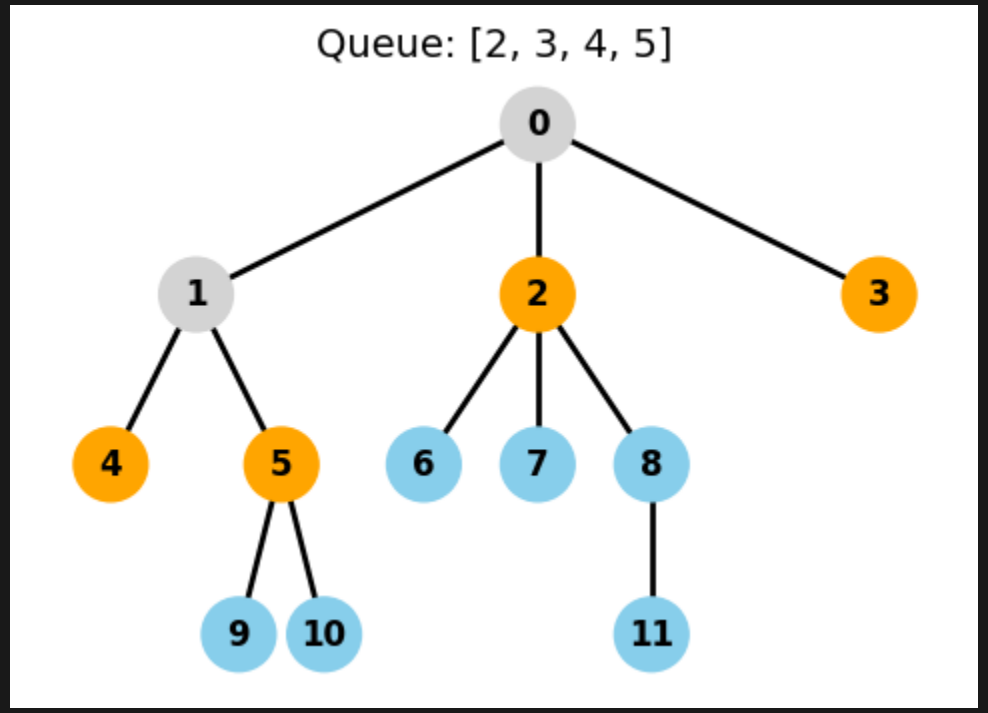
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



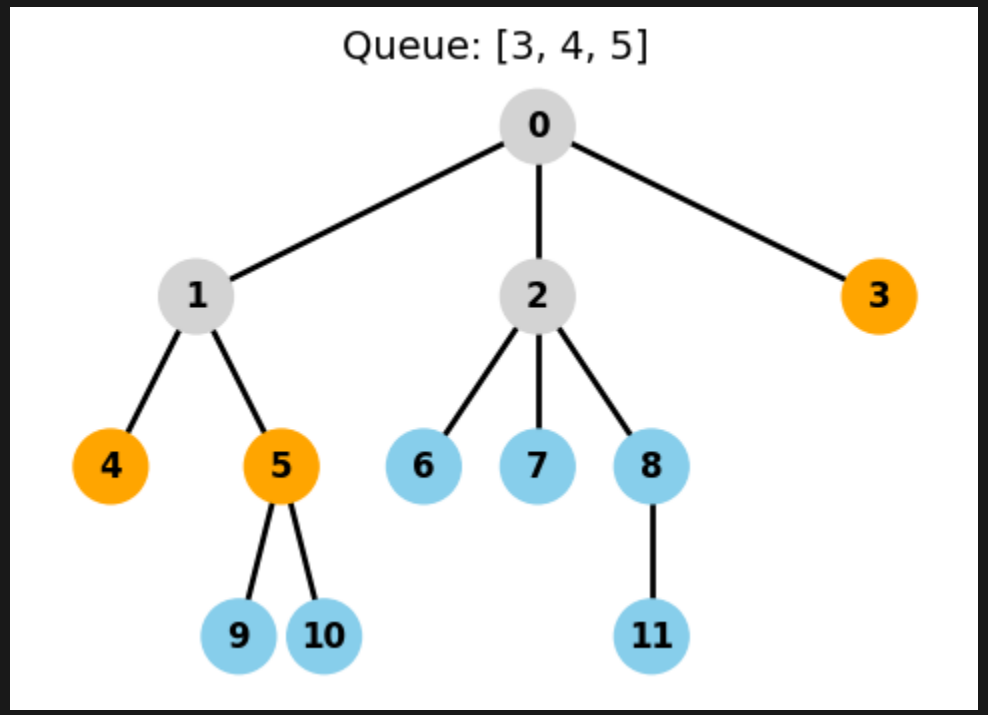
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



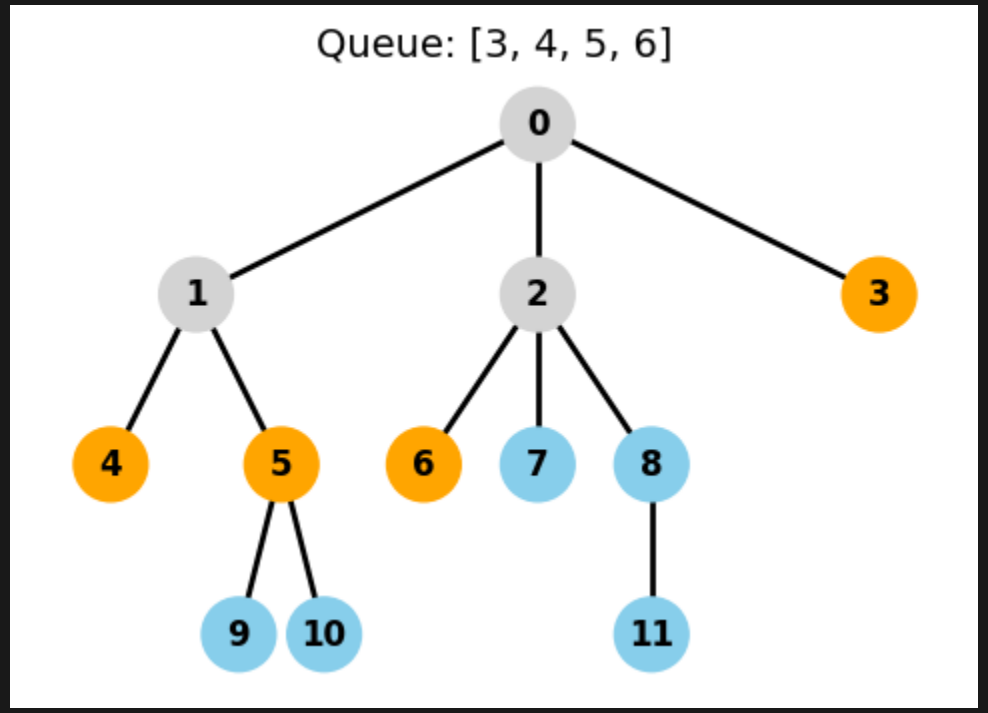
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



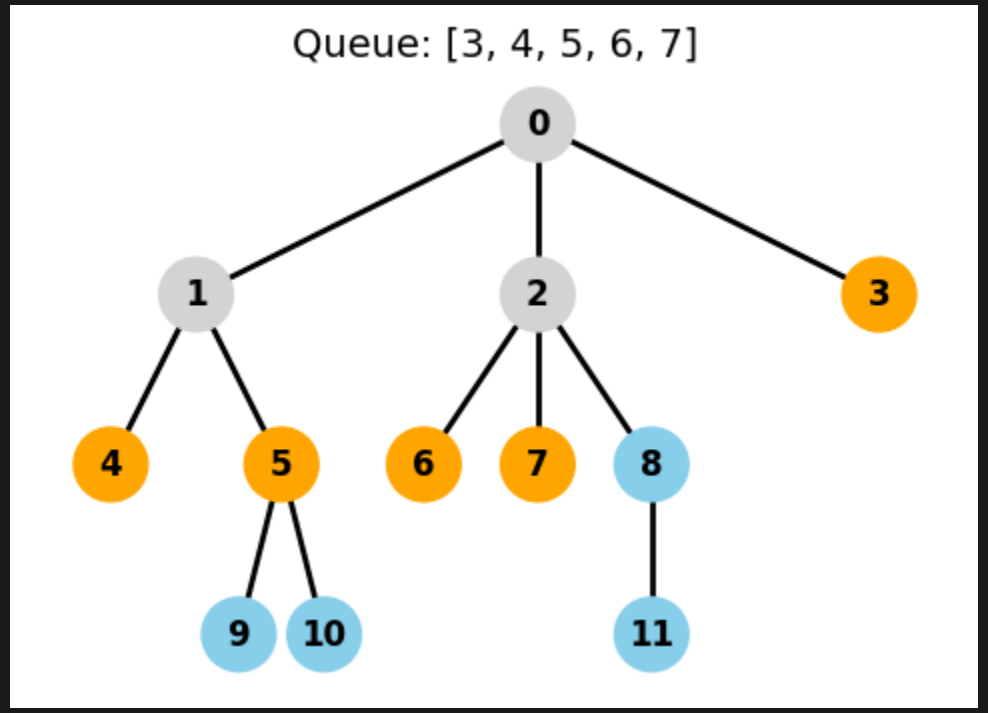
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



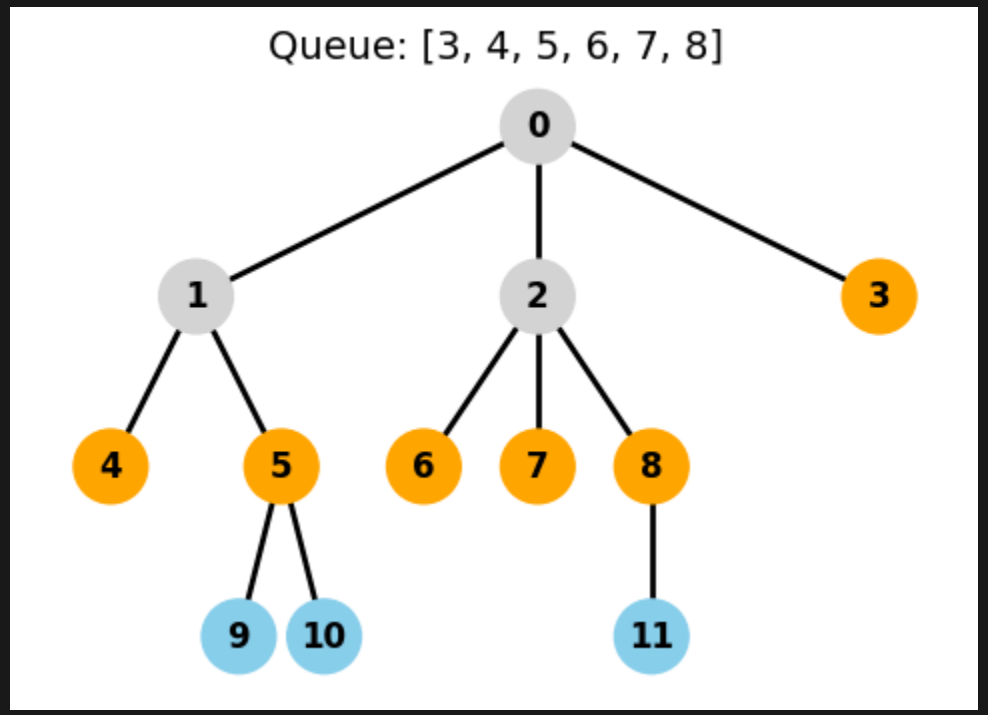
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



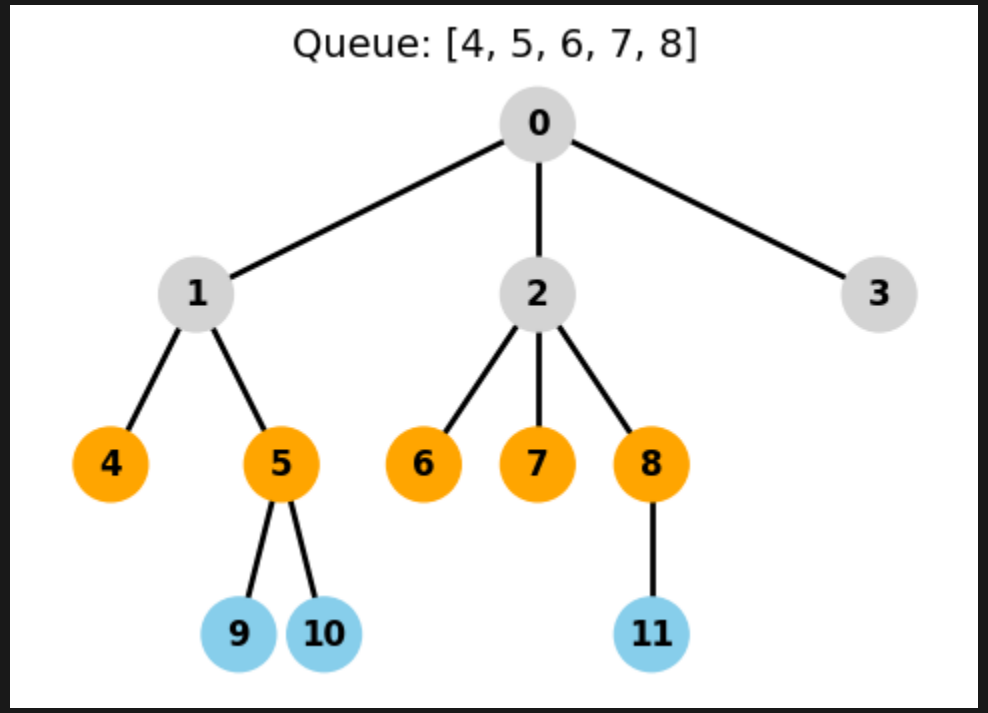
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



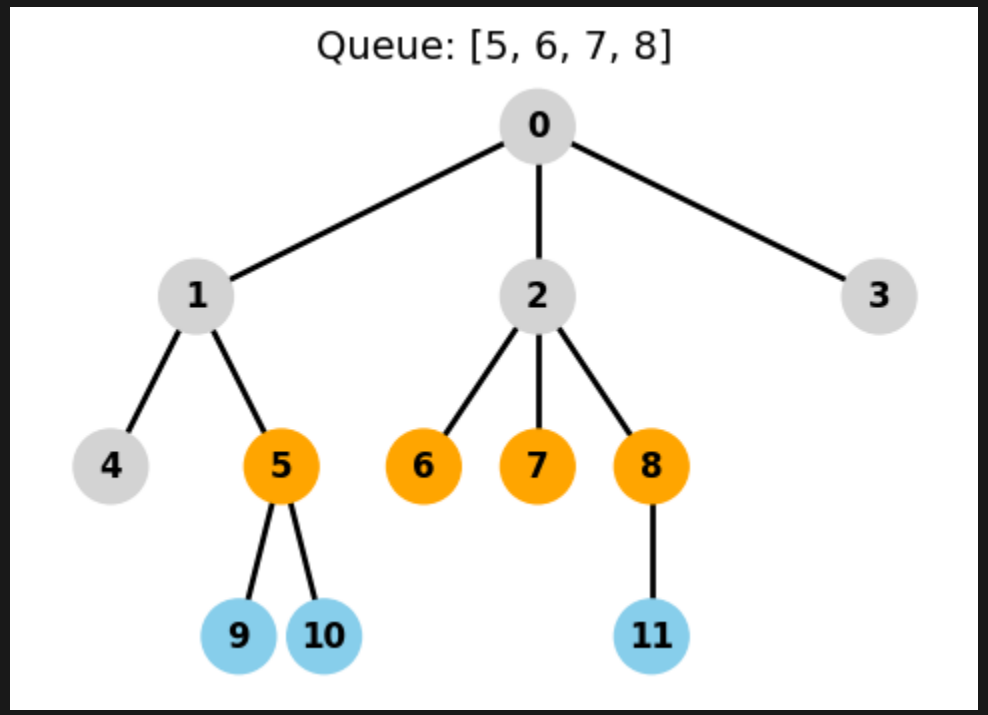
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



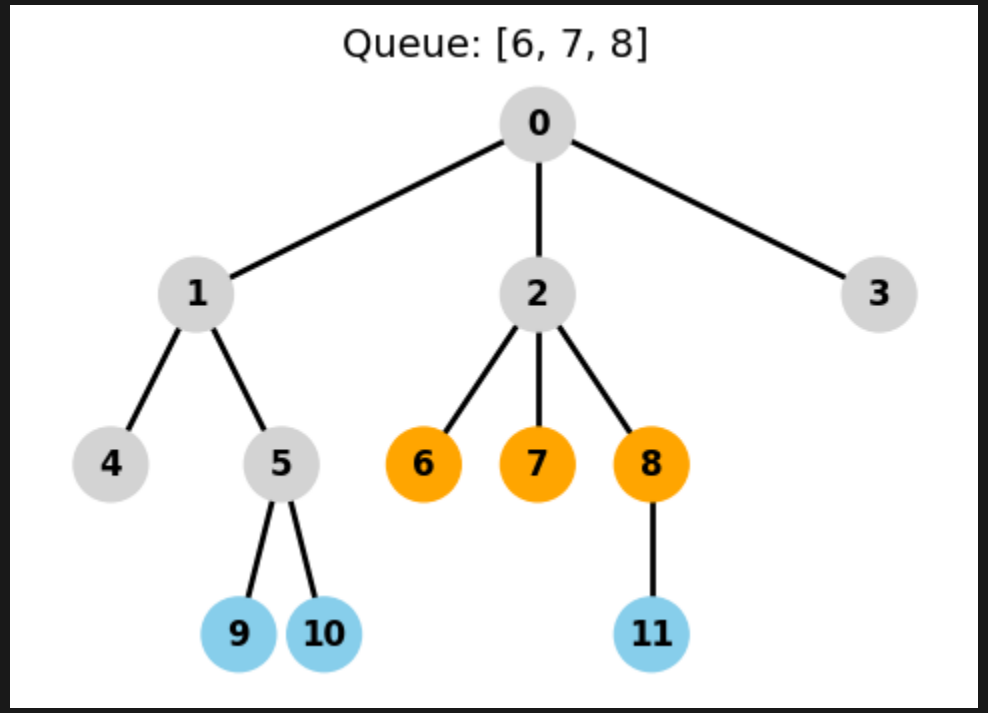
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



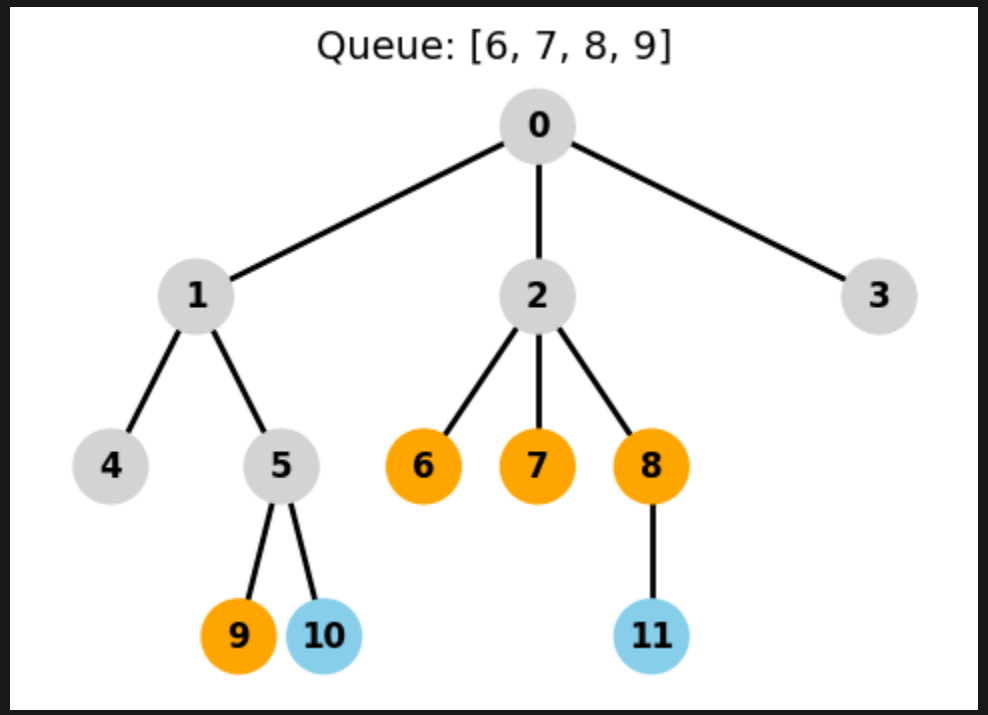
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



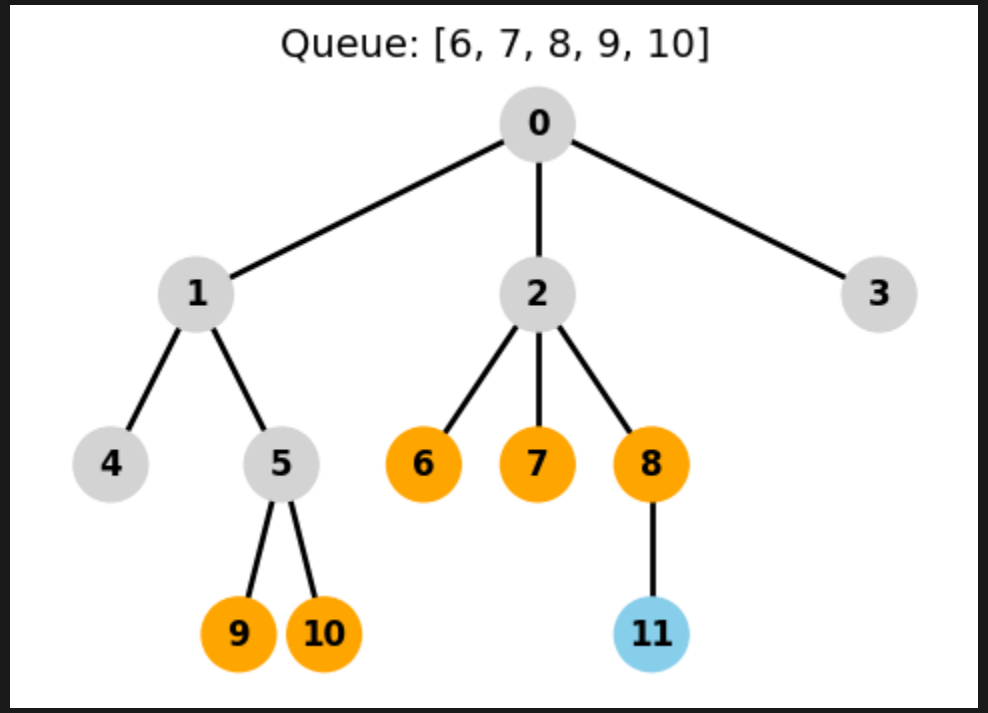
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



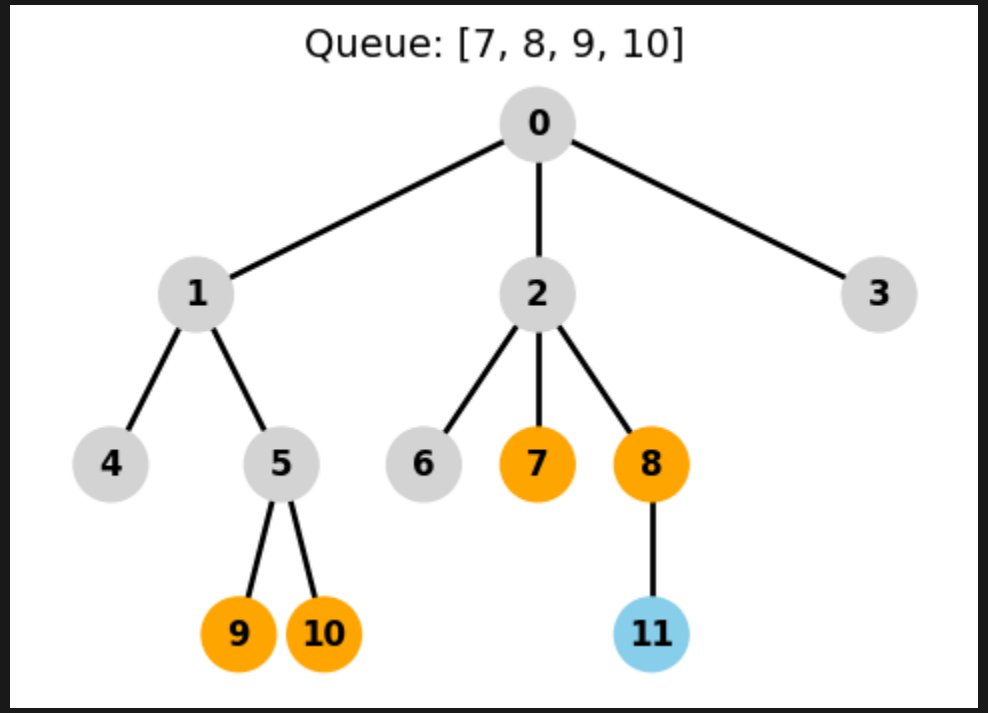
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



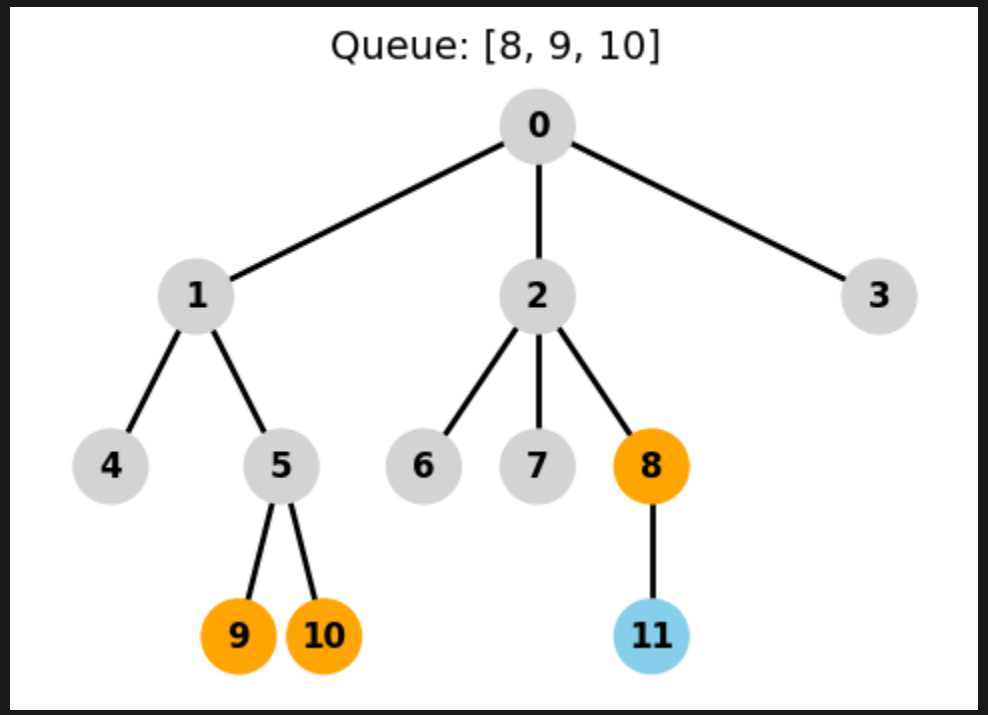
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



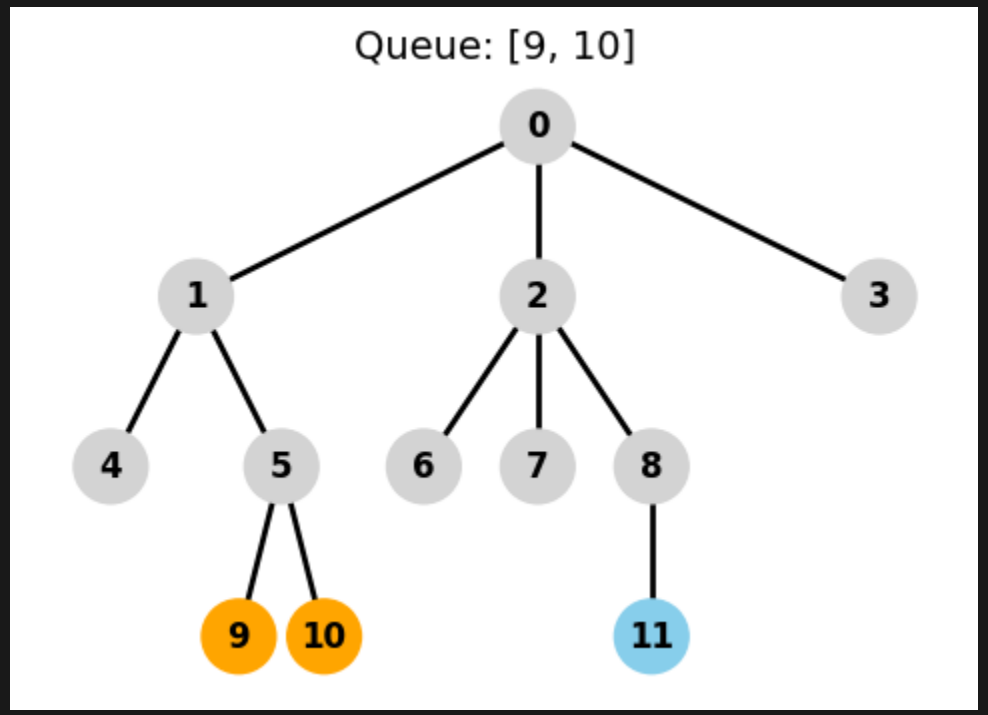
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



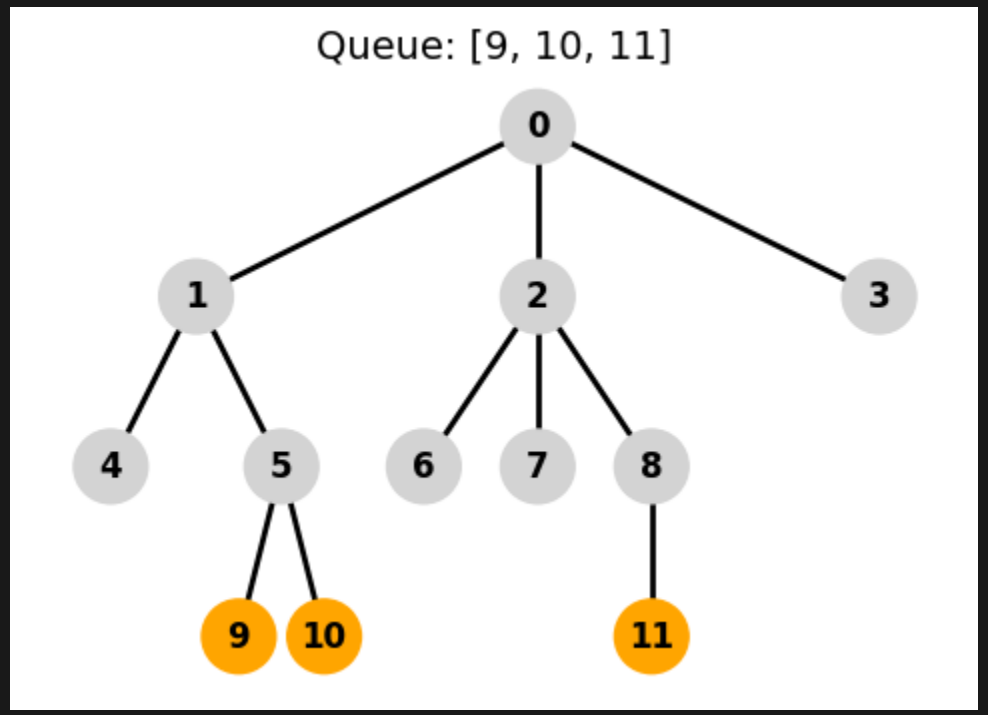
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



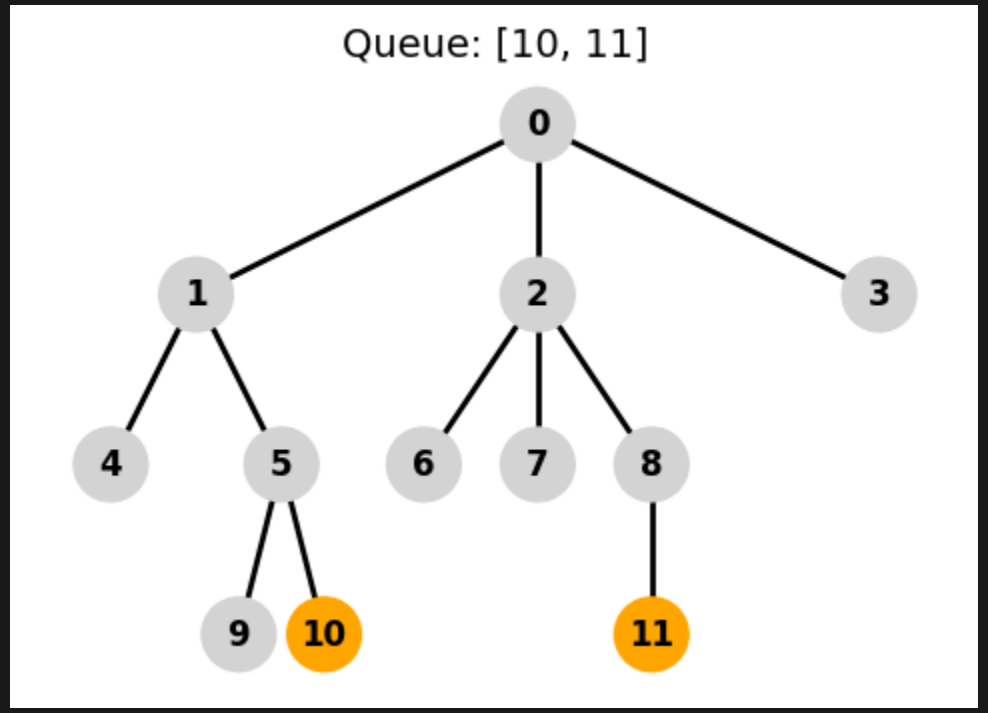
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



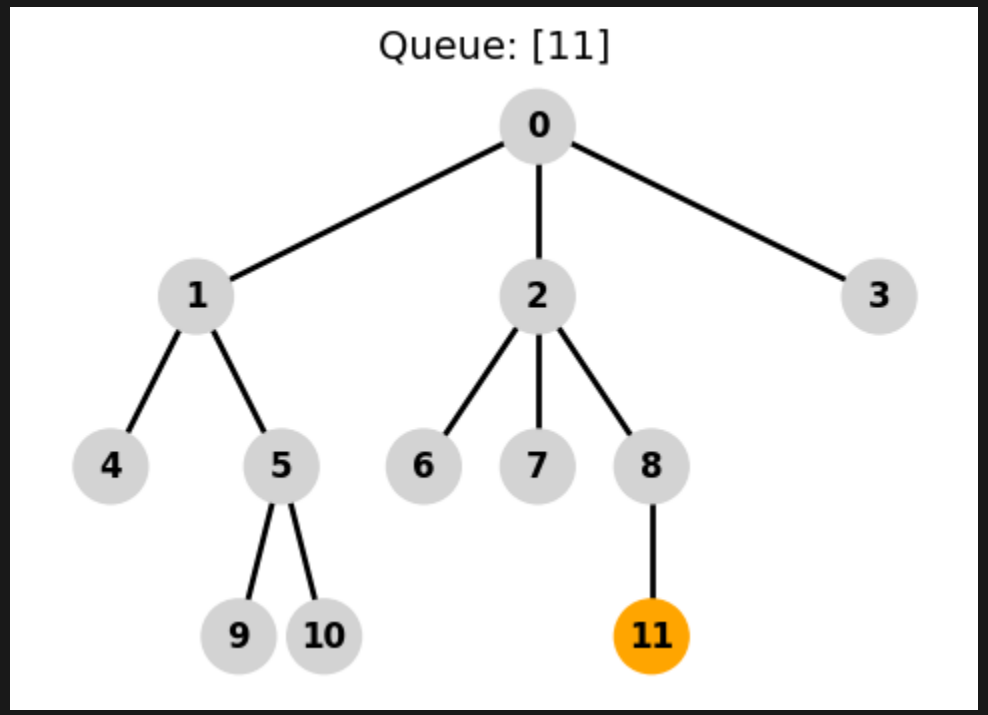
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



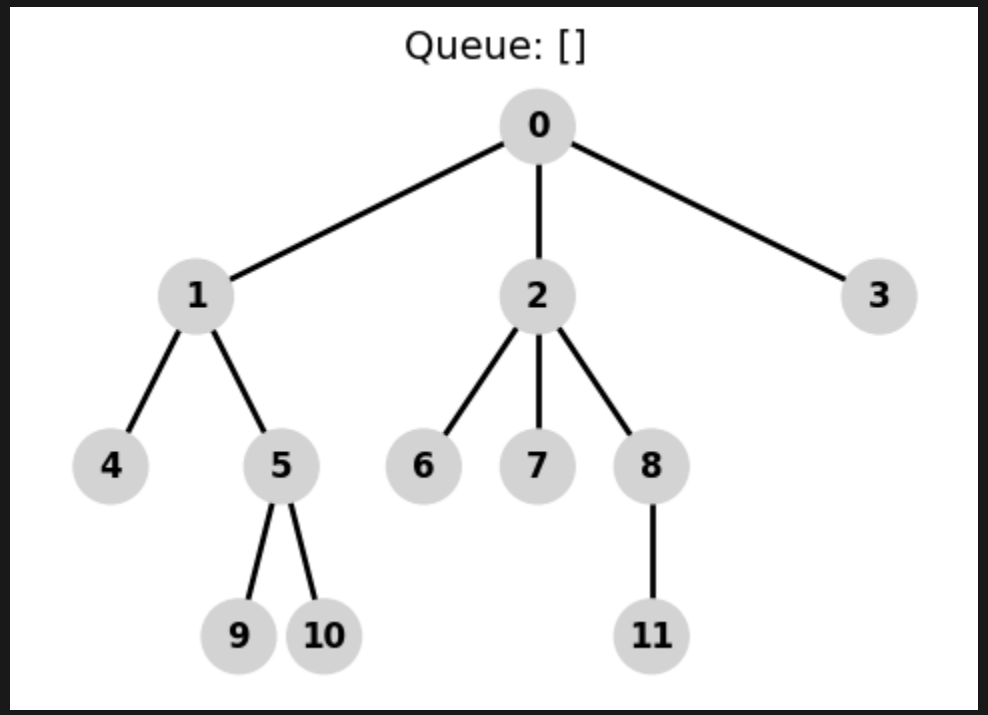
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



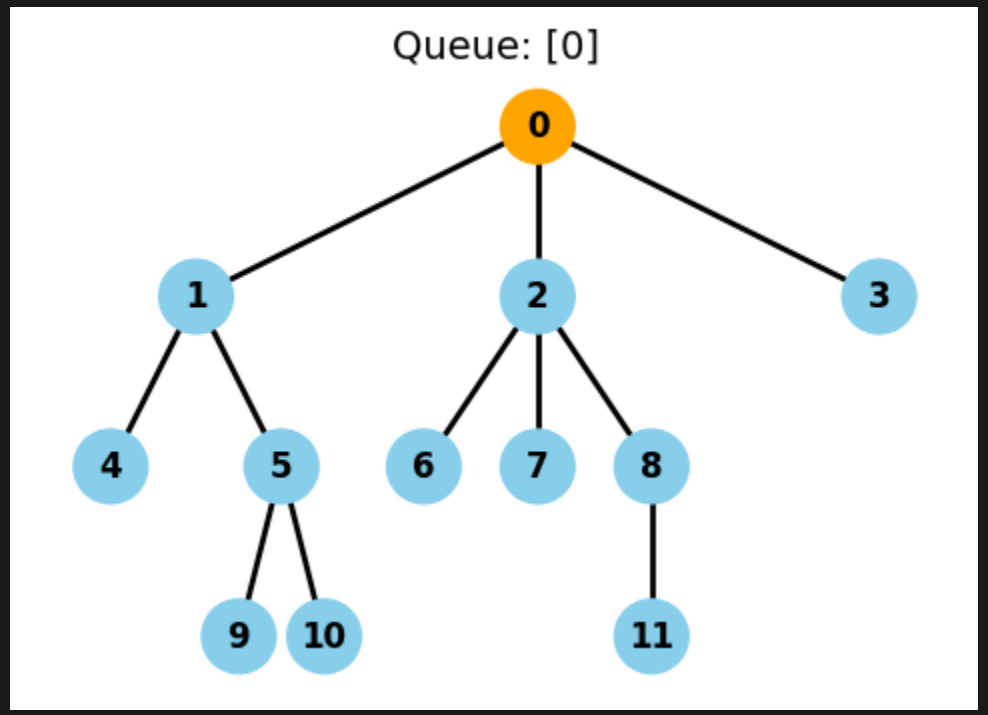
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



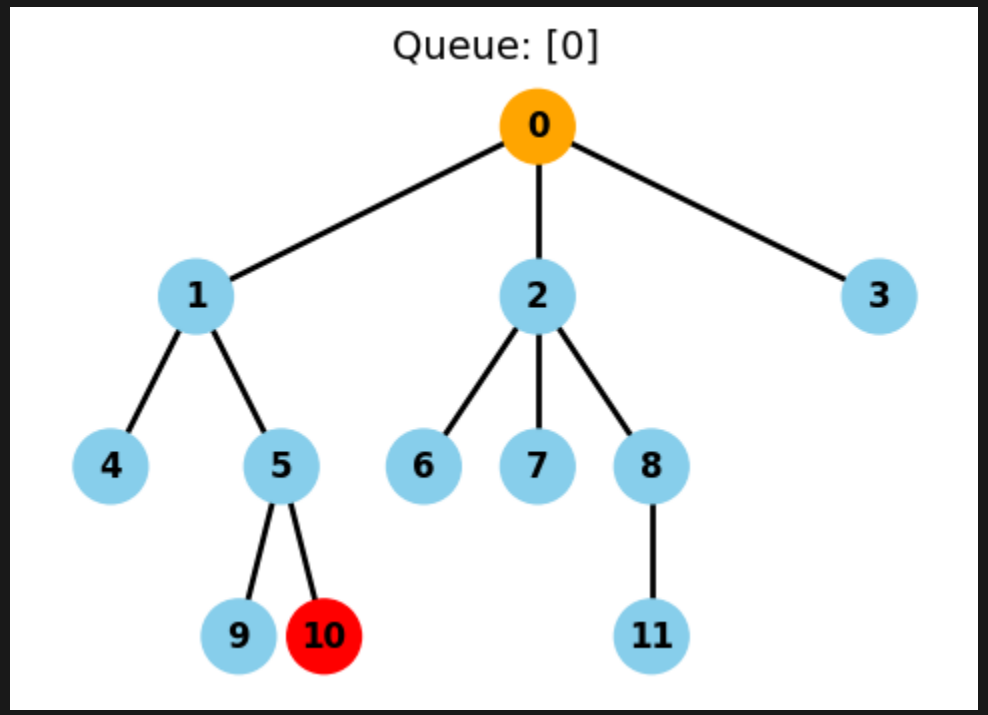
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



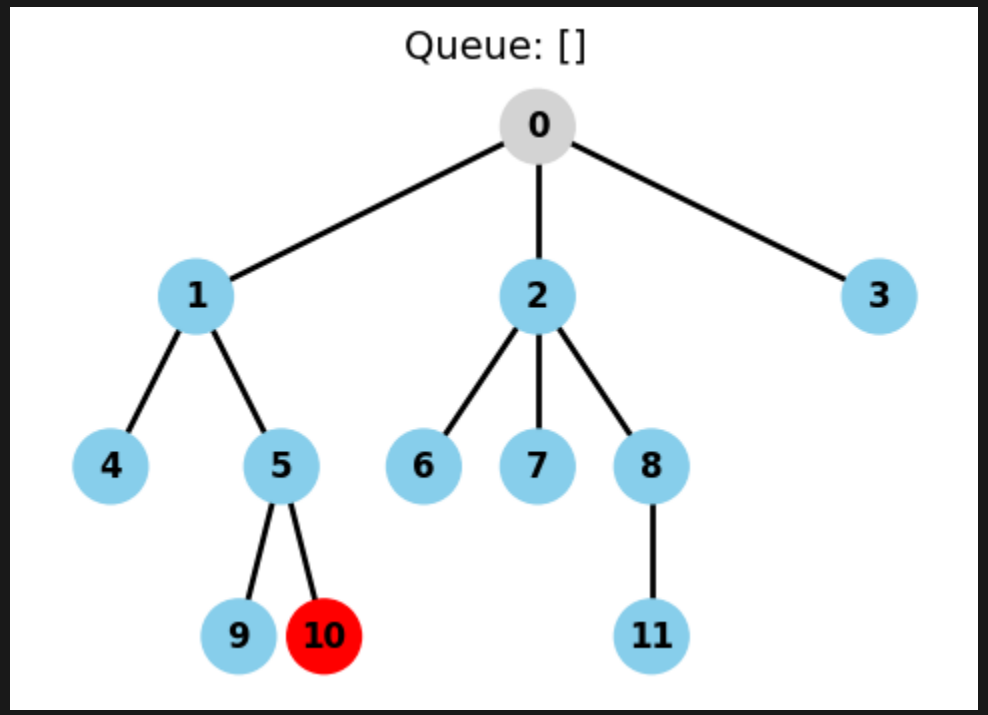
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



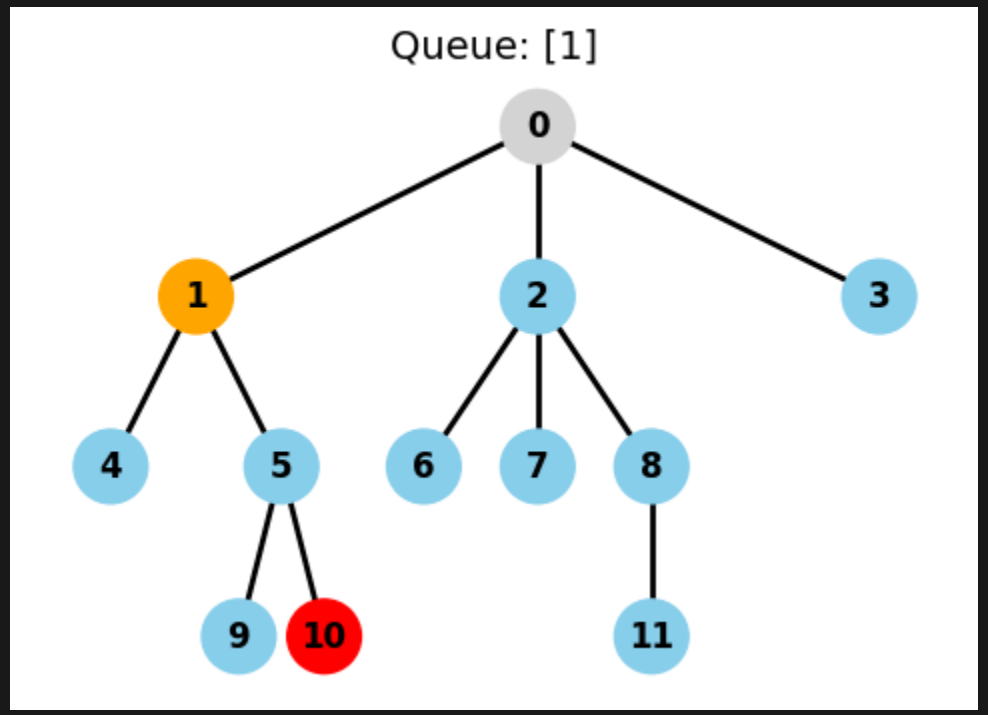
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



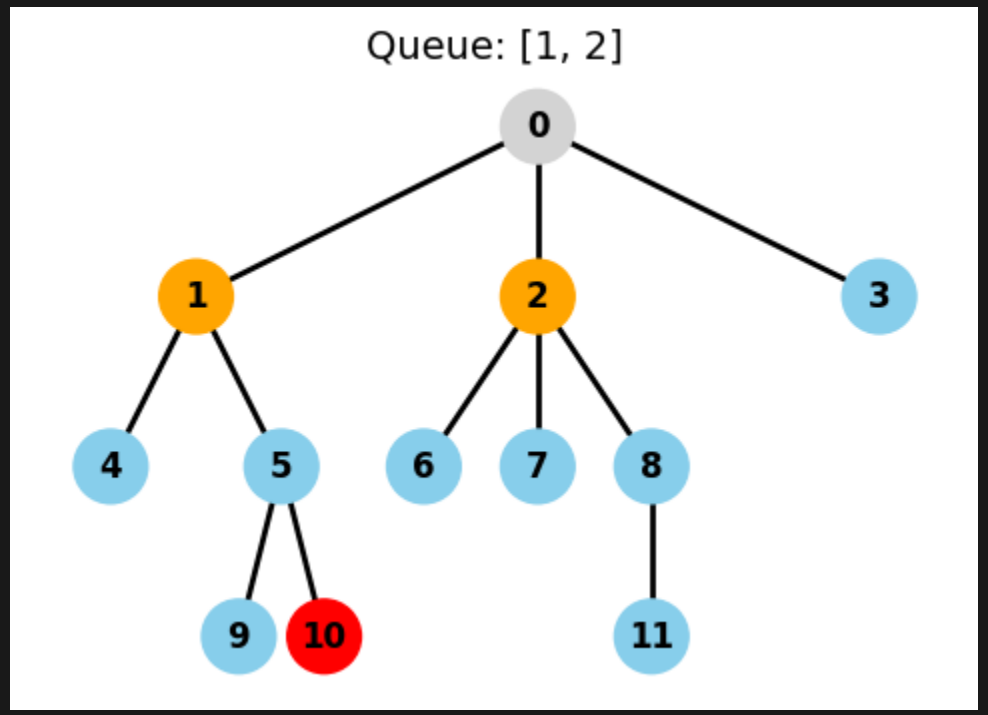
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



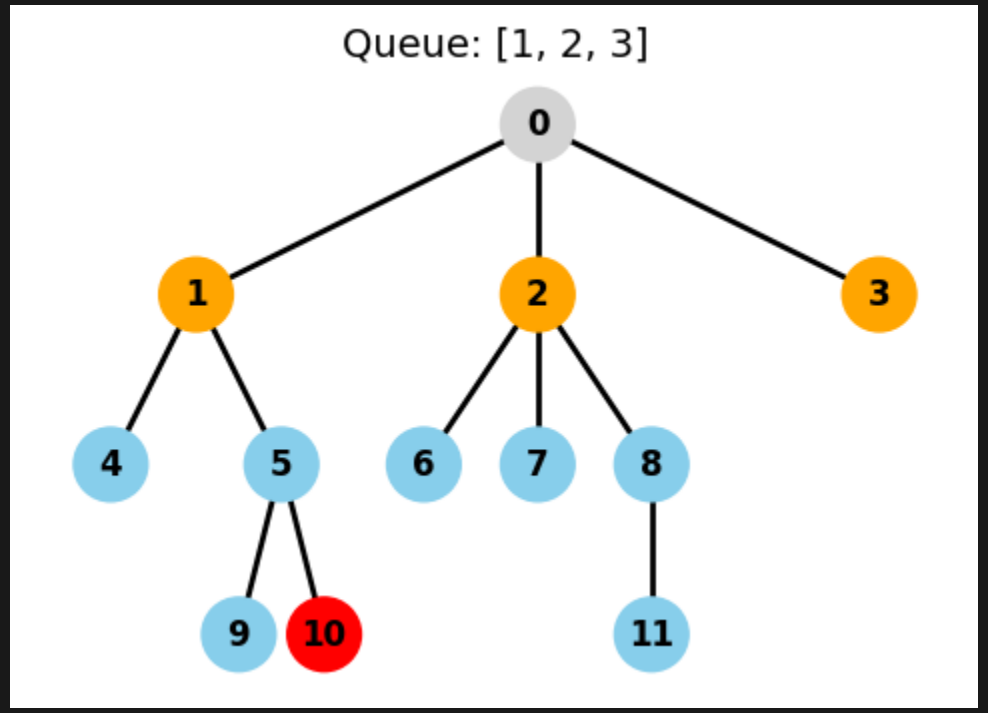
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



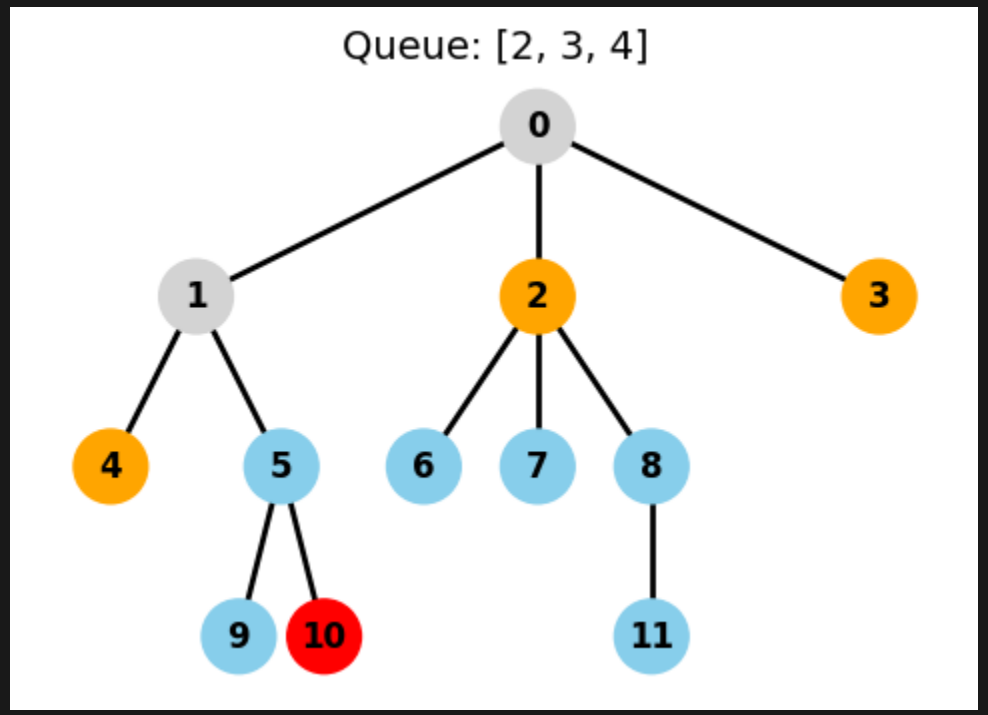
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



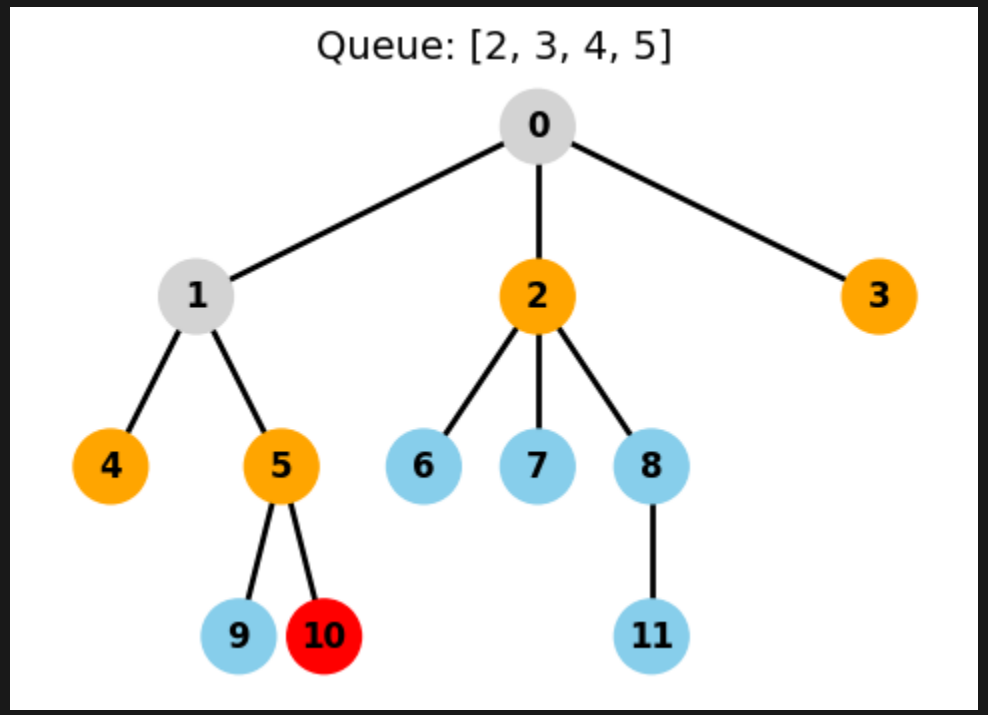
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



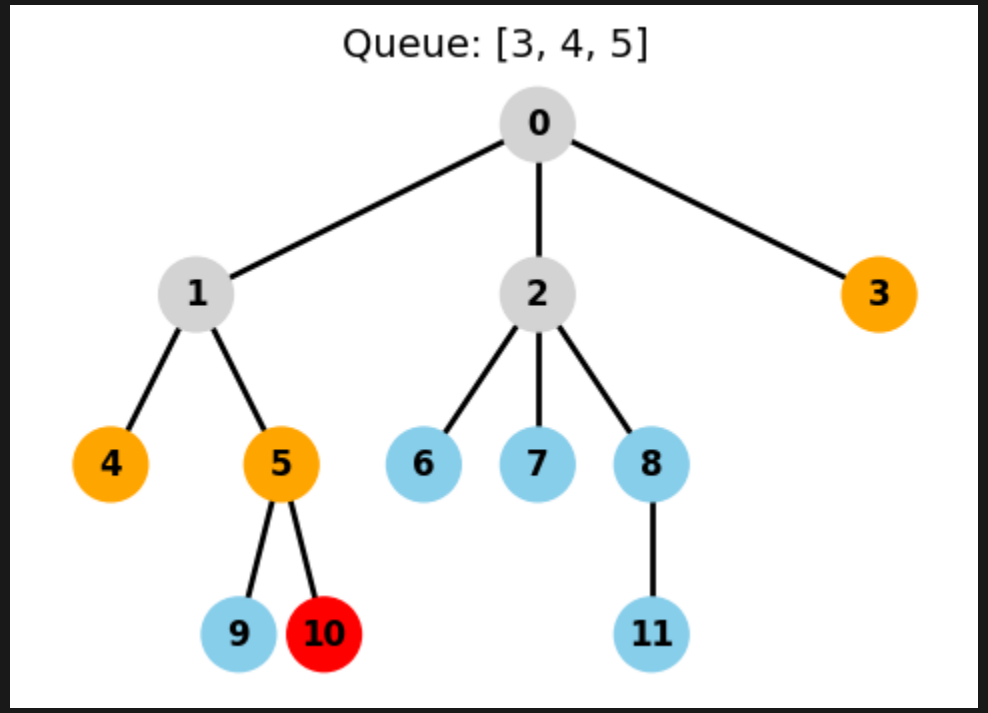
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



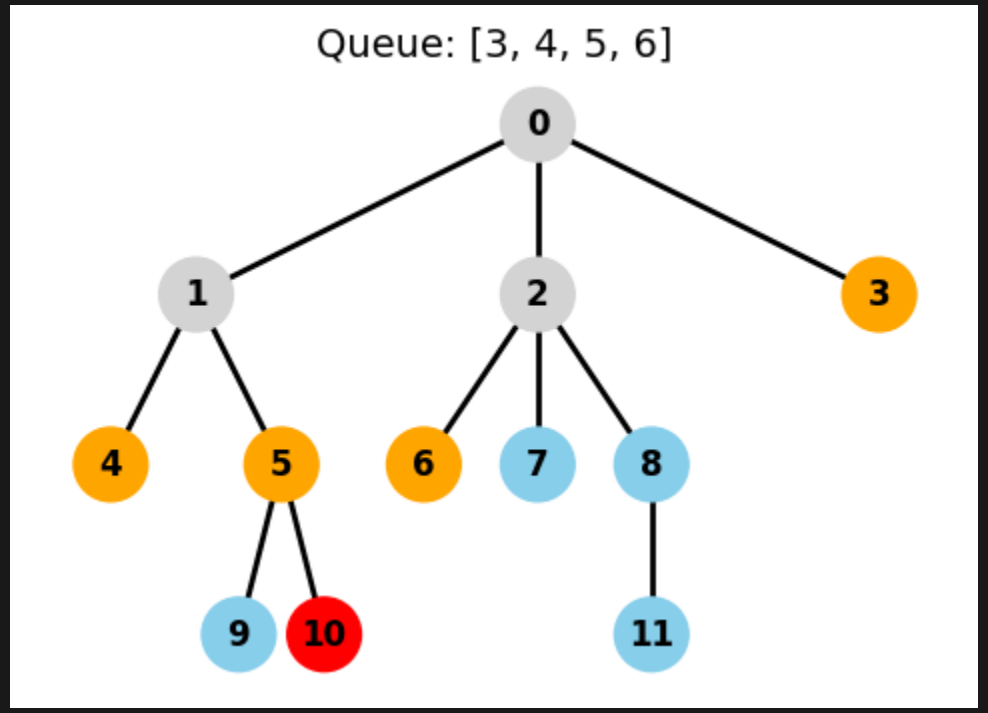
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



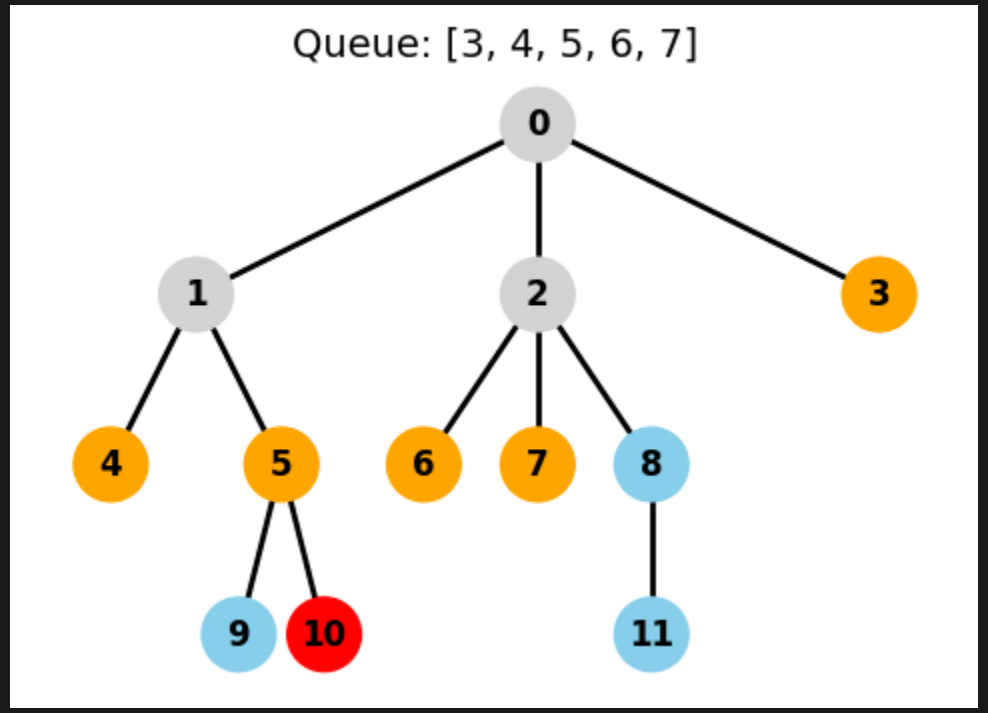
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



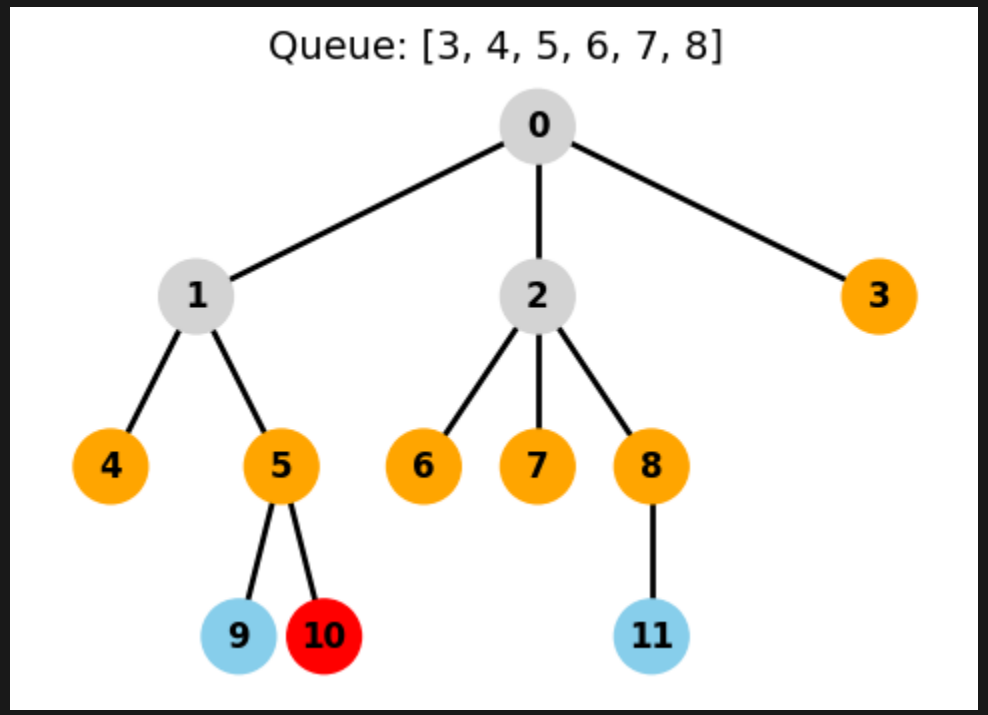
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



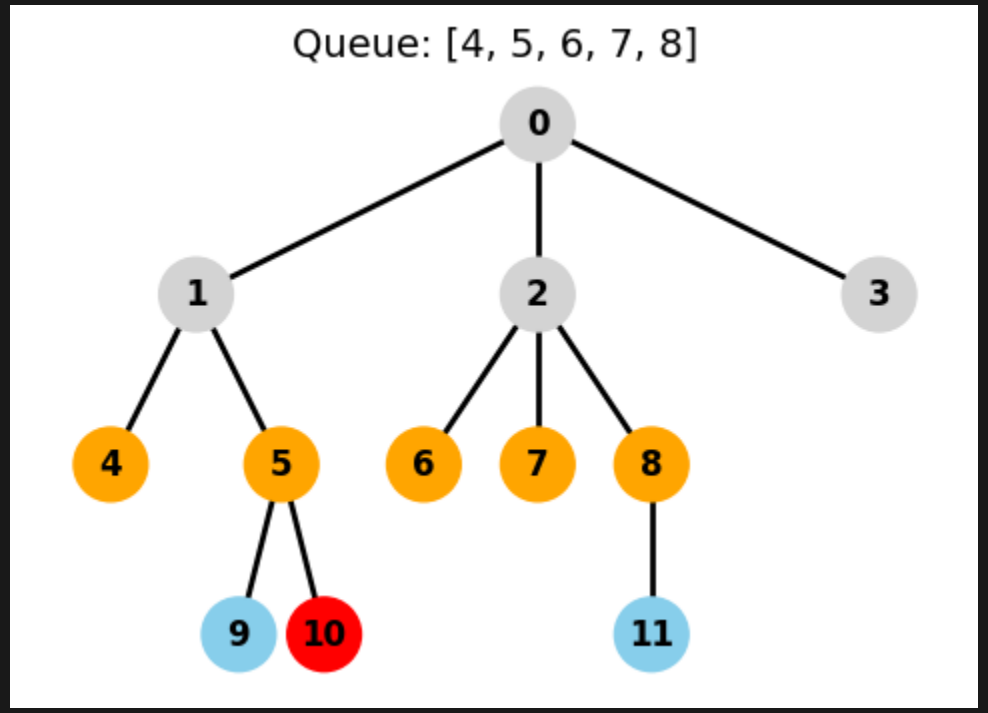
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



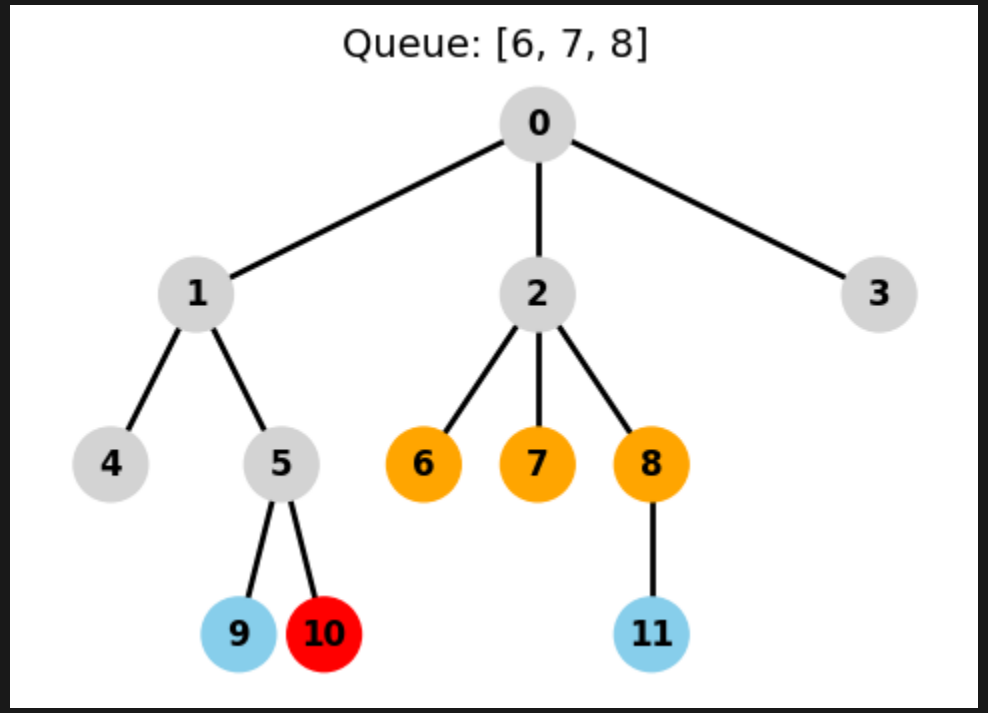
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



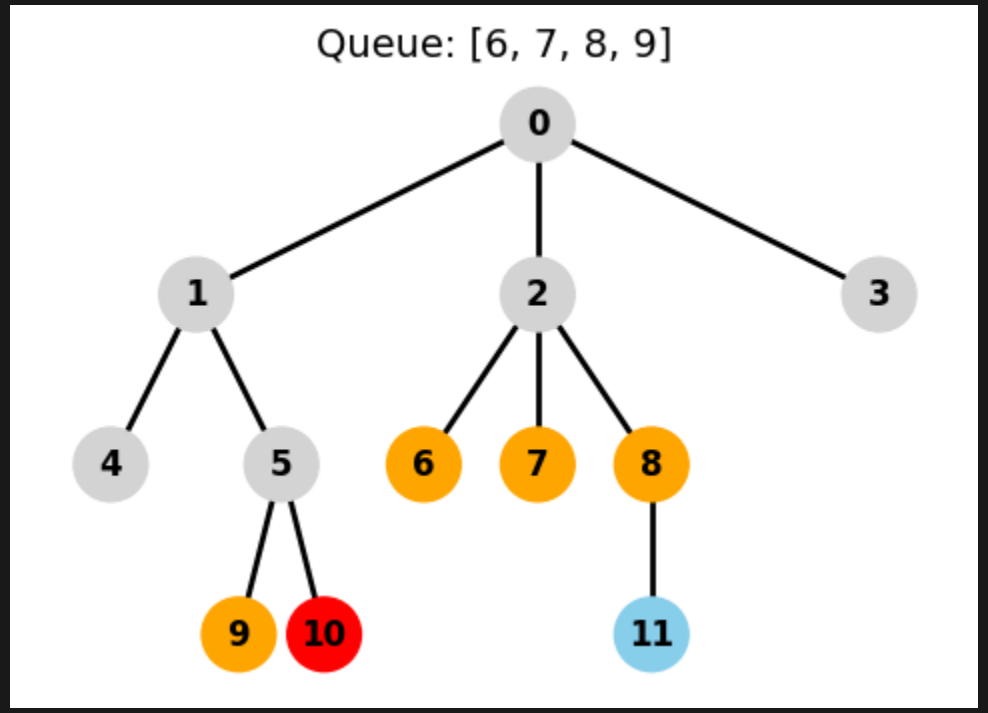
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



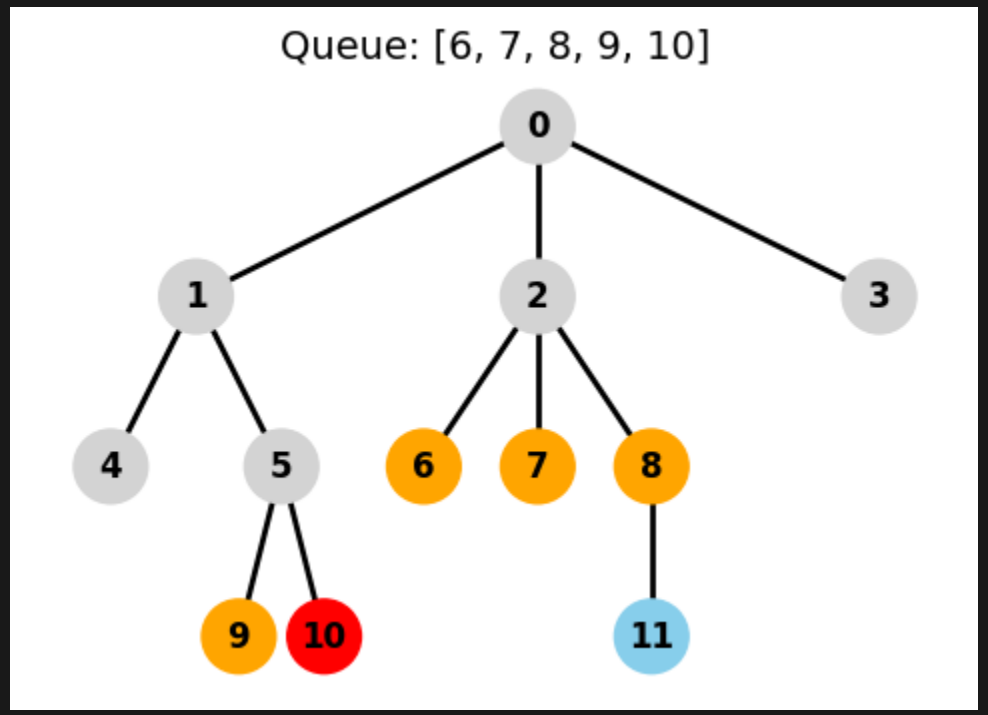
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



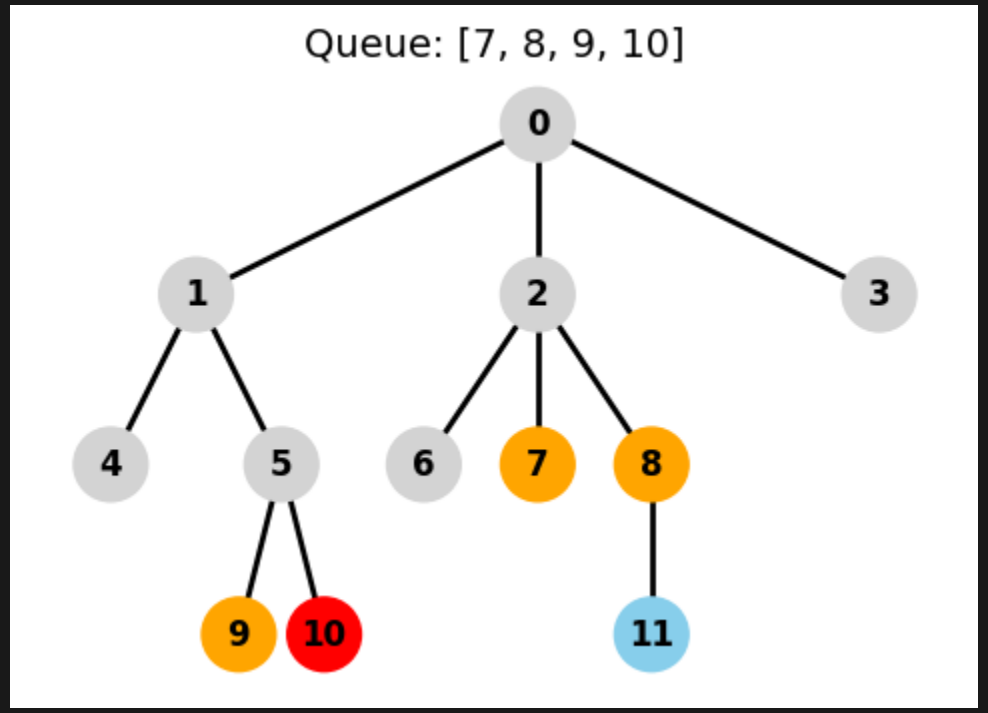
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



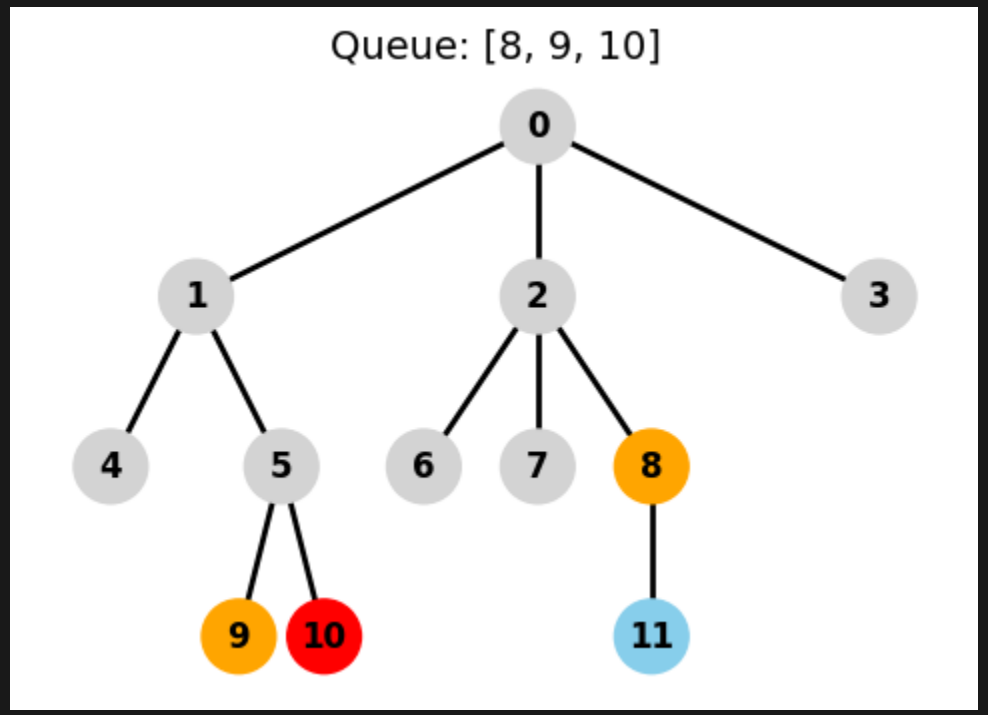
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



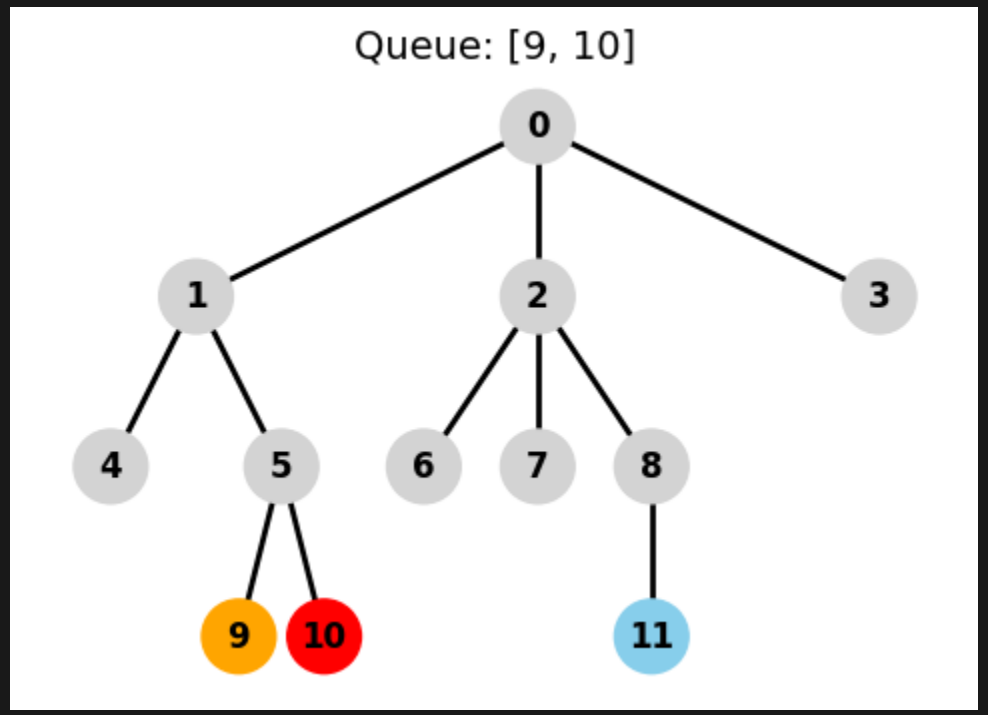
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



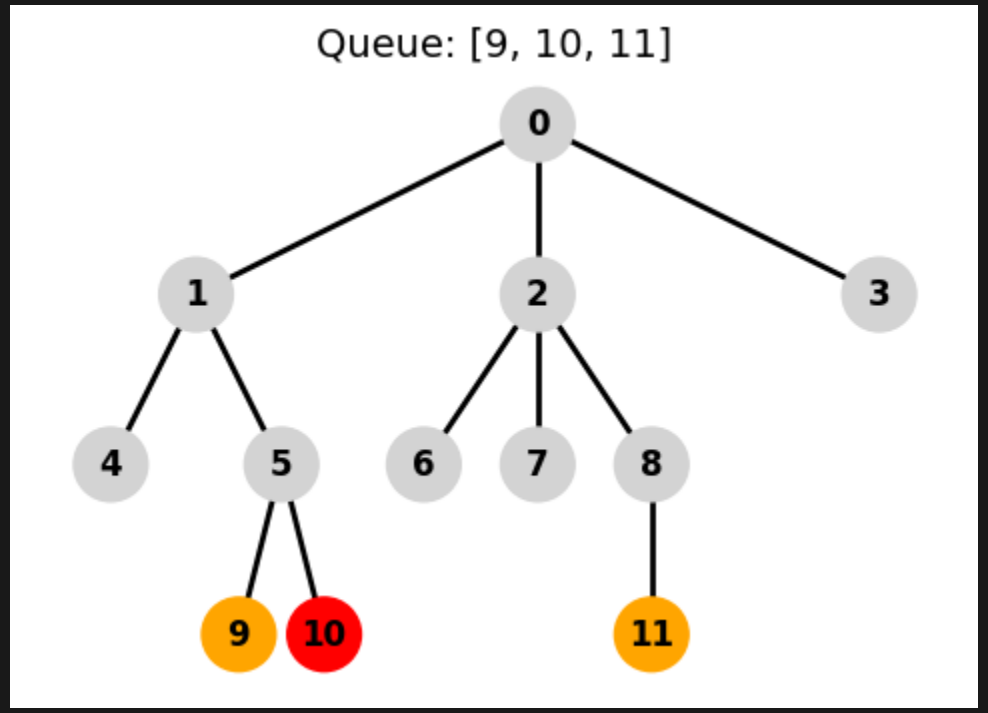
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



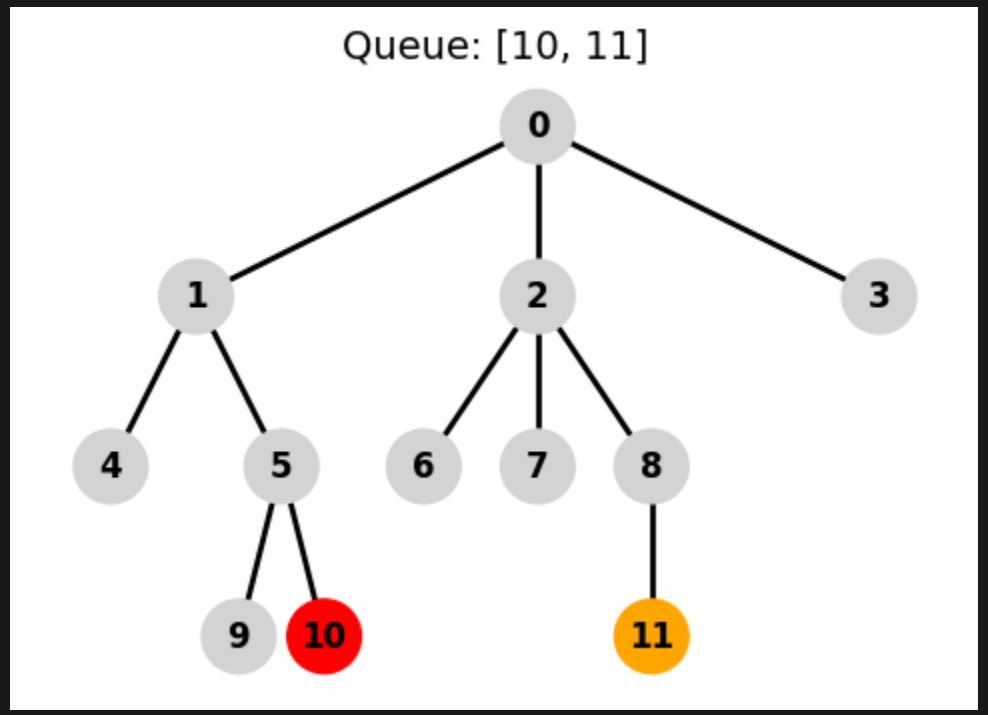
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



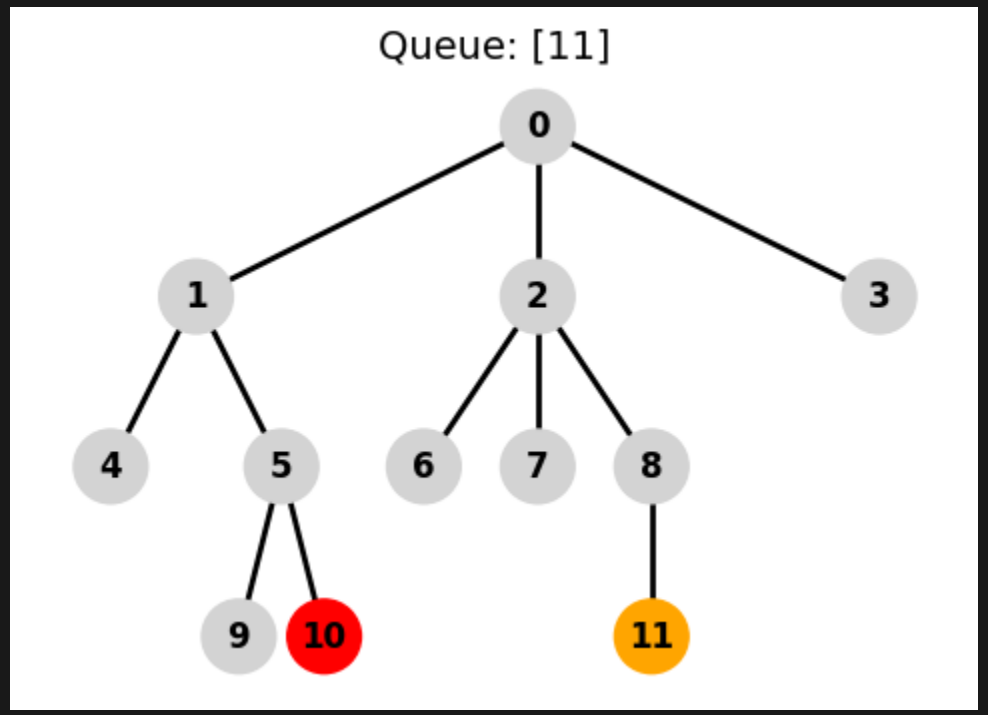
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



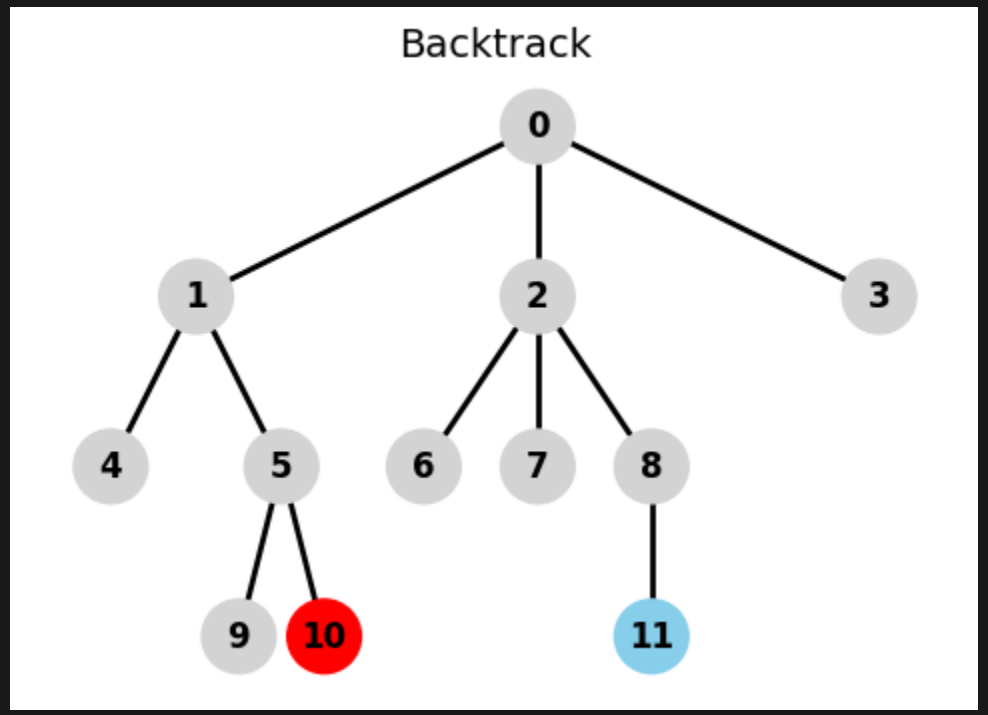
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



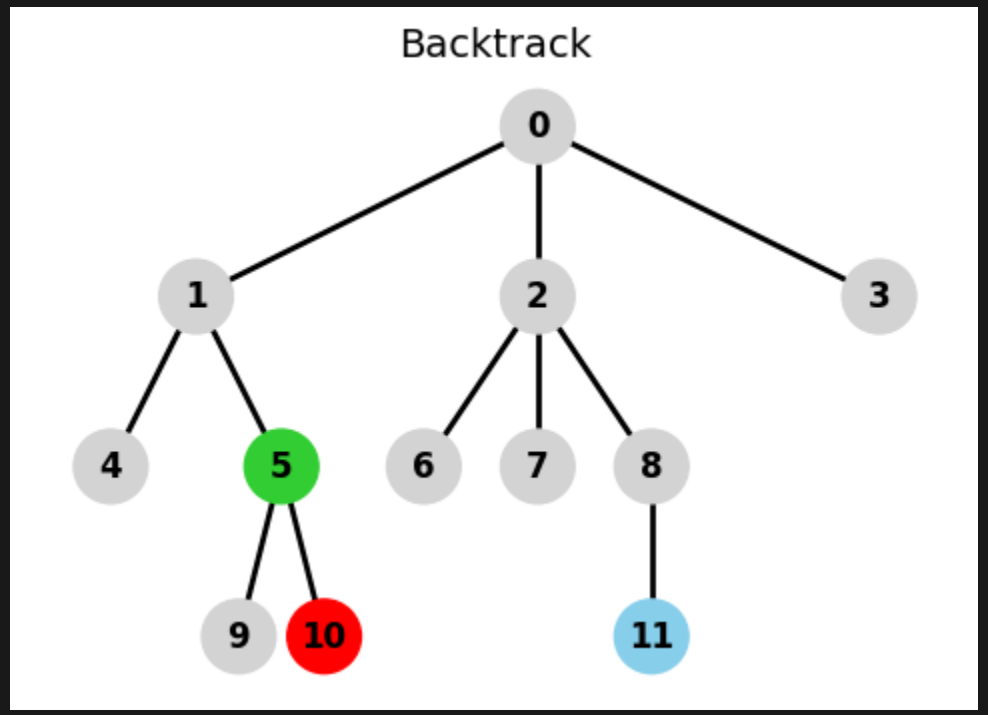
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



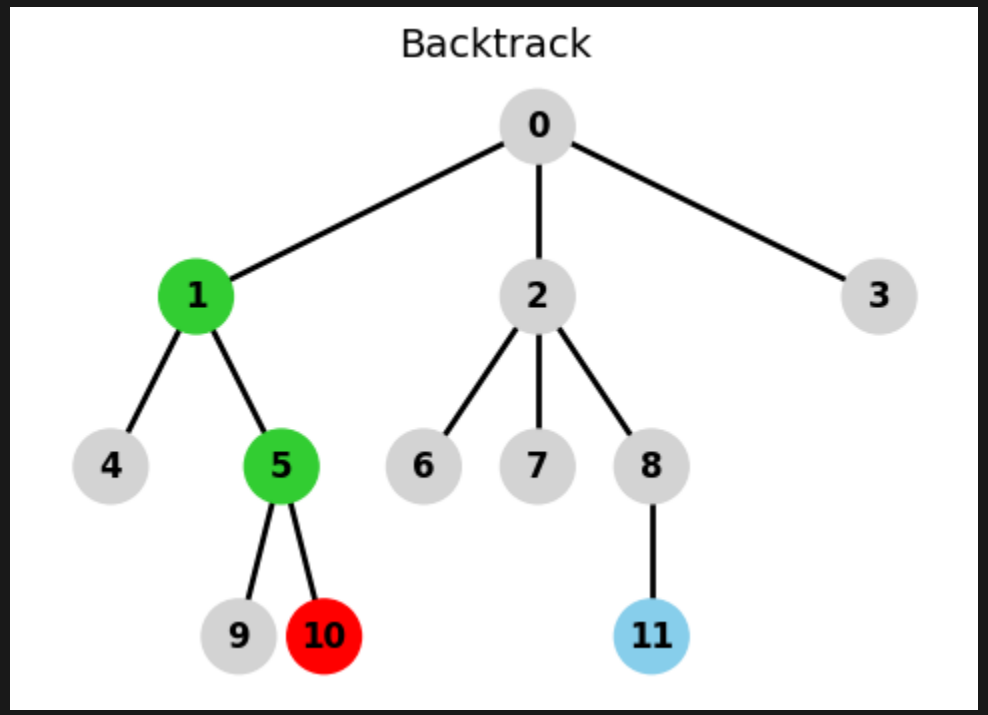
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



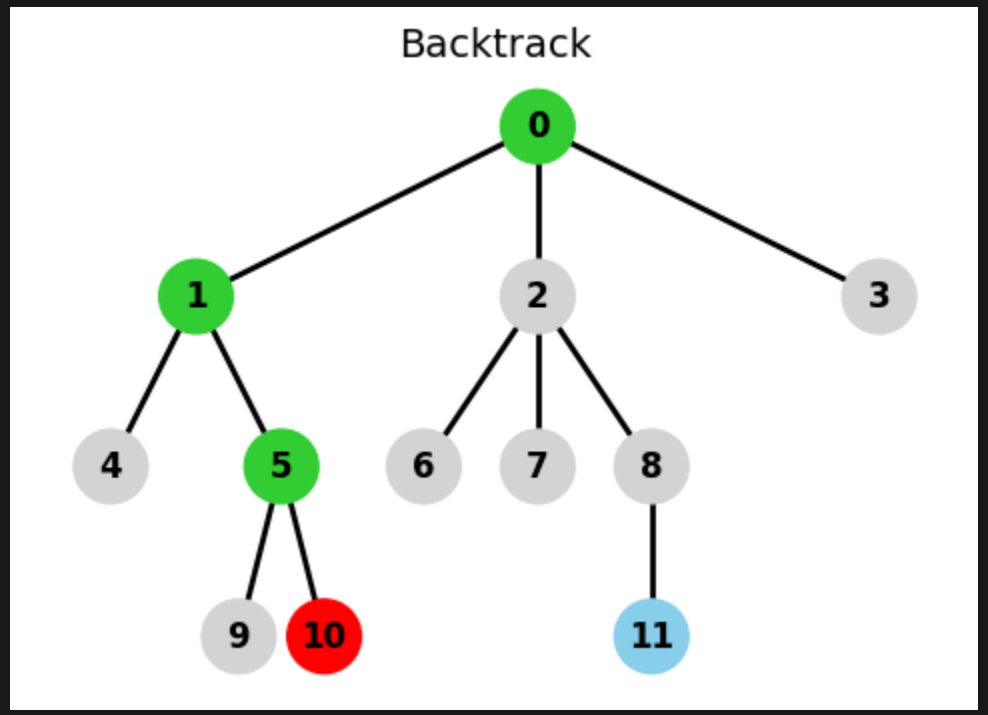
Breadth First Search (BFS)

Queue: First In First Out (FIFO)



Breadth First Search (BFS)

Queue: First In First Out (FIFO)

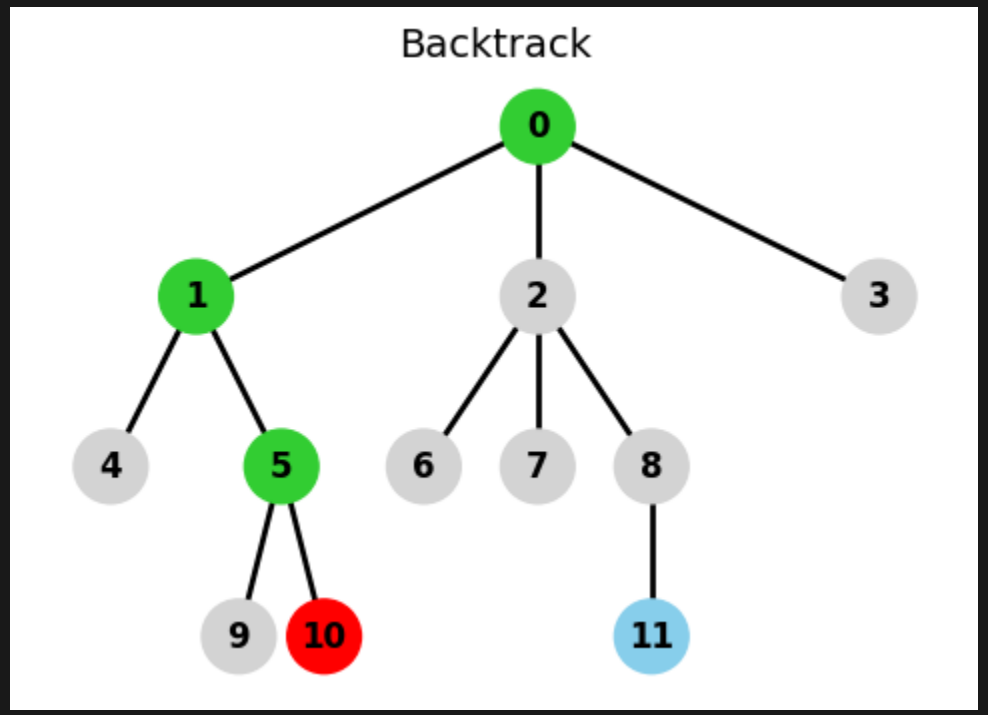


Breadth First Search (BFS)

Queue: First In First Out (FIFO)

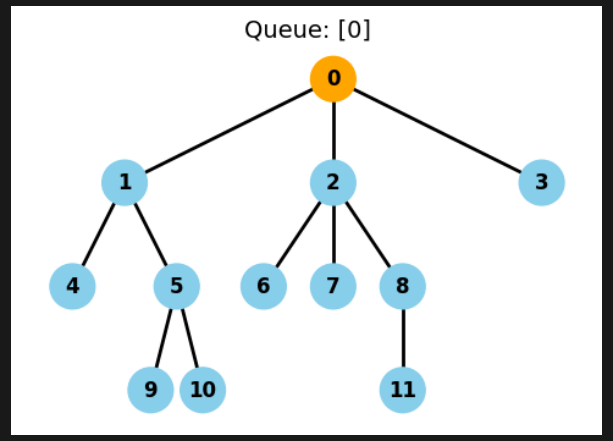
특징

- 시간 복잡도: O(V+E)
- 공간 복잡도: O(V)
- 목표 노드가 시작점 근처에 있을 때 빠르게 발견 가능
- 가중치가 모두 동일할 경우 최단 경로 탐색 가능



Breadth First Search (BFS)

수도 코드 (pseudo code)

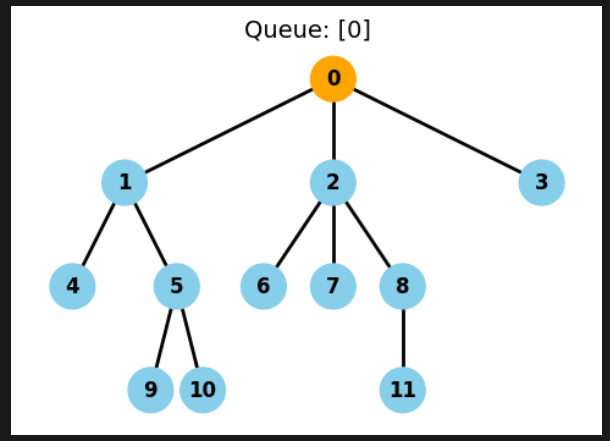



```
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'G'],
    'F': ['C'],
    'G': ['E']
}

bfs(graph, start='A')
```

```
from collections import deque

def bfs(graph, start_node):
    visited = set()
    queue = deque()

    queue.append(start_node)
    visited.add(start_node)

    while queue:
        current_node = queue.popleft()

        for neighbor in graph.get(current_node, []):
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
```

Breadth First Search (BFS)

수도 코드 (pseudo code)

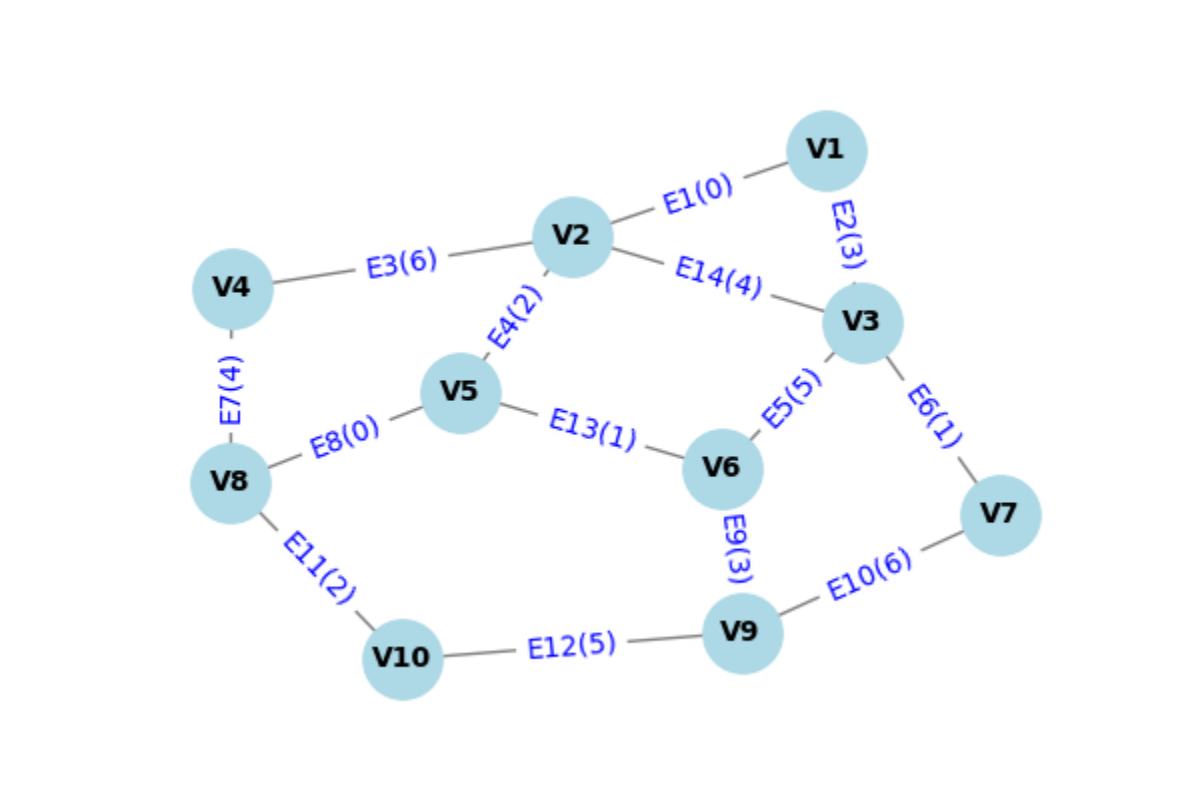



```
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'G'],
    'F': ['C'],
    'G': ['E']
}

bfs(graph, start='A')
```

```
from collections import deque

def bfs(graph, start_node):
    visited = set()
    queue = deque()

    queue.append(start_node)
    visited.add(start_node)

    while queue:
        current_node = queue.popleft()

        for neighbor in graph.get(current_node, []):
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
```

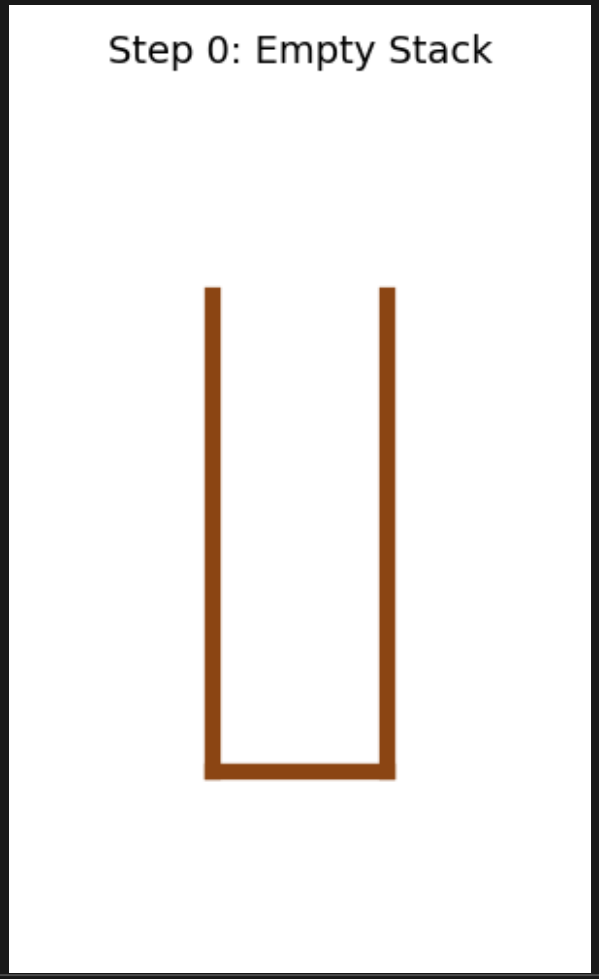
```
from collections import deque

def bfs_path(graph, start, goal):
    visited = set()
    queue = deque()
    parent = {}

    queue.append(start)
    visited.add(start)
    parent[start] = None

    while queue:
        current = queue.popleft()

        if current == goal:
            path = []
            while current is not None:
                path.append(current)
                current = parent[current]
            path.reverse()
            return path

        for neighbor in graph.get(current, []):
            if neighbor not in visited:
                visited.add(neighbor)
                parent[neighbor] = current
                queue.append(neighbor)

    return None
```

Depth First Search (DFS)



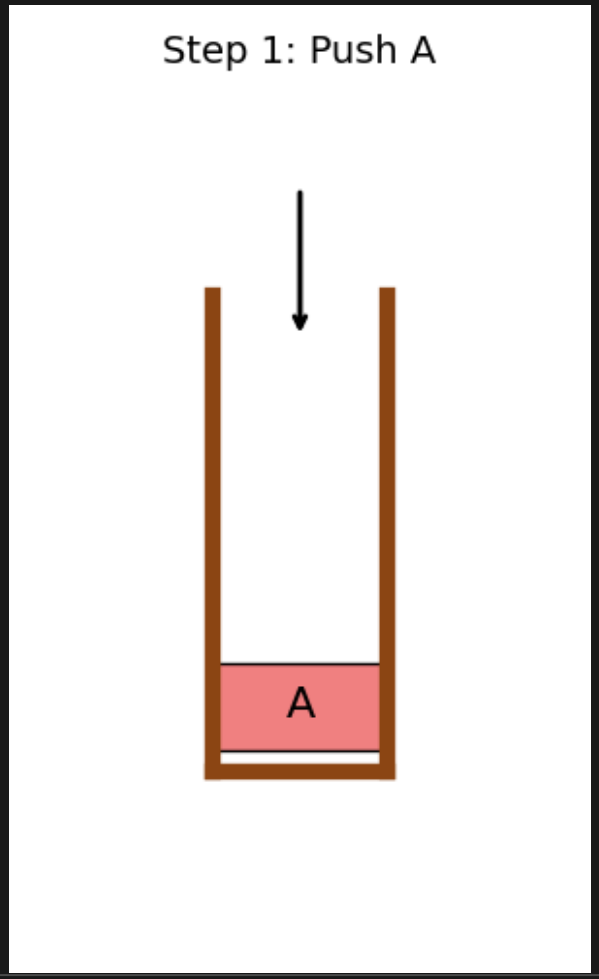
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



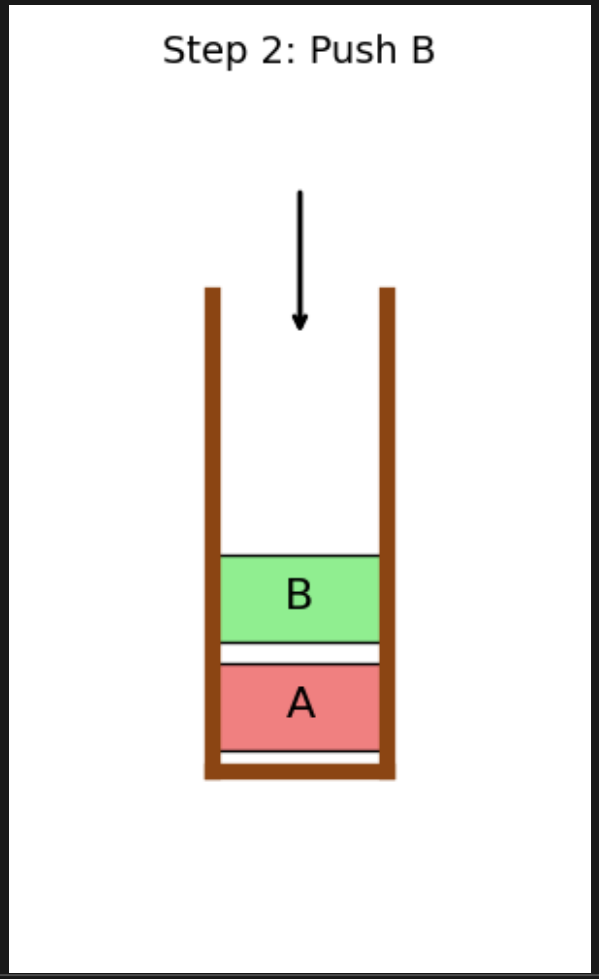
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



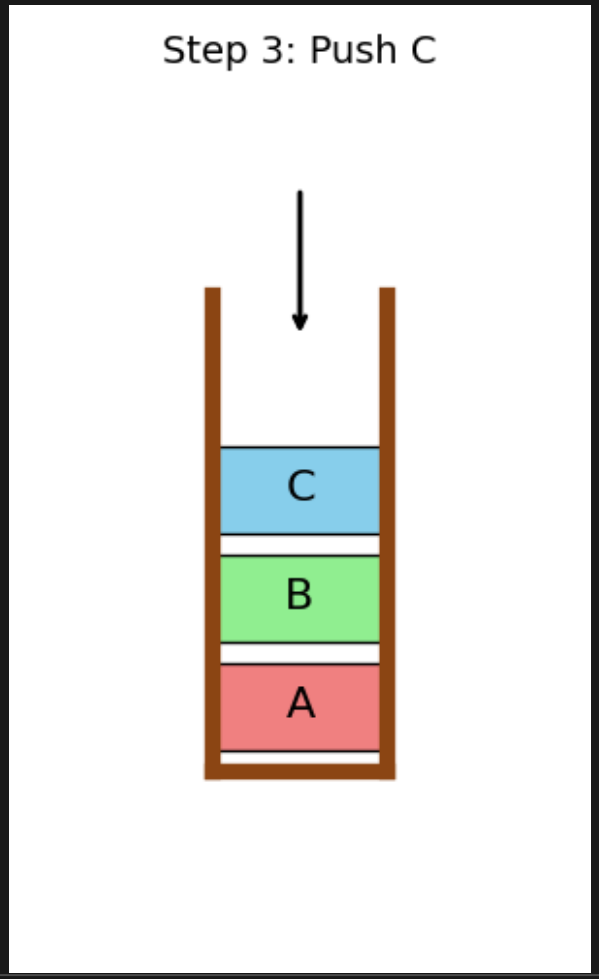
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



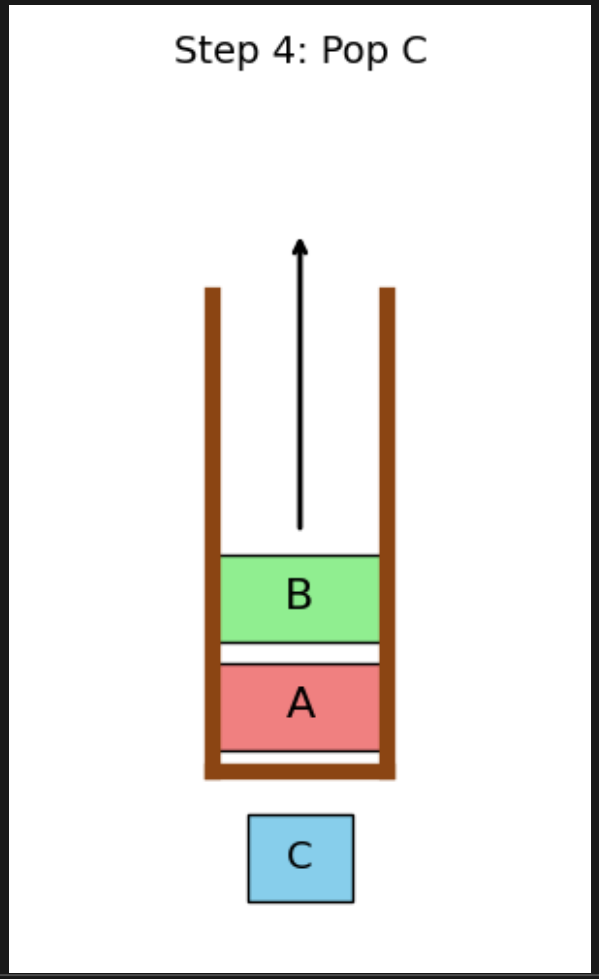
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



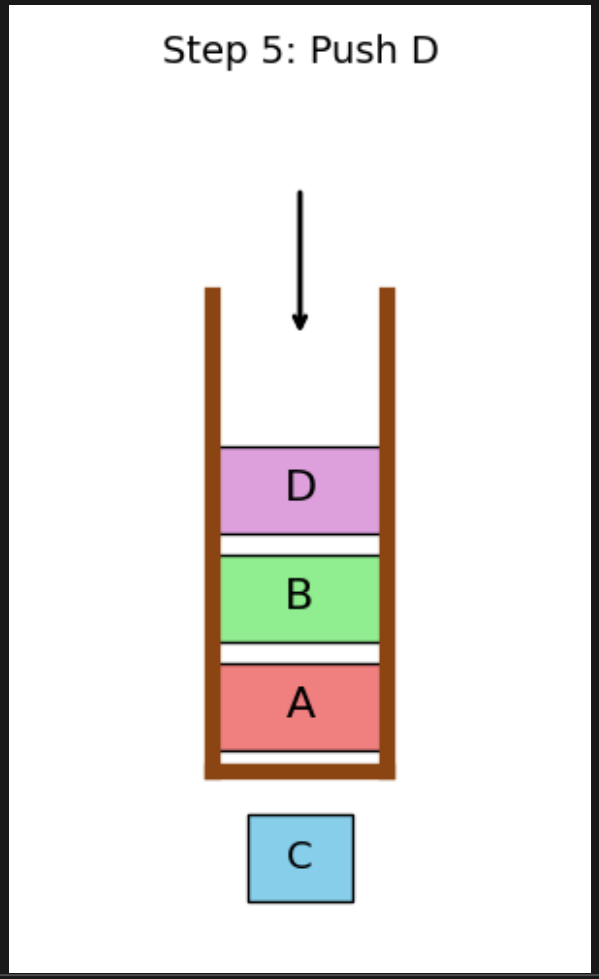
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



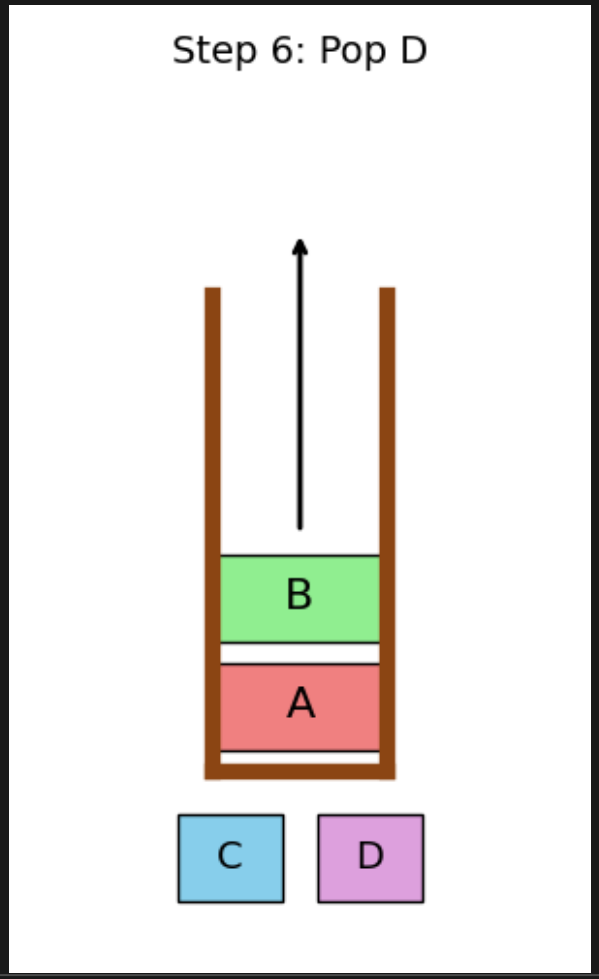
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



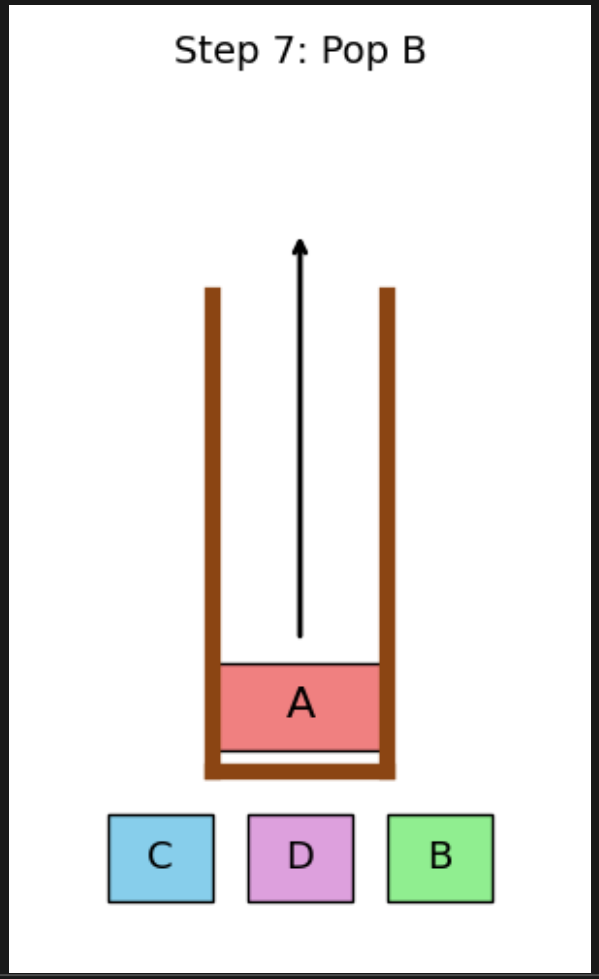
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



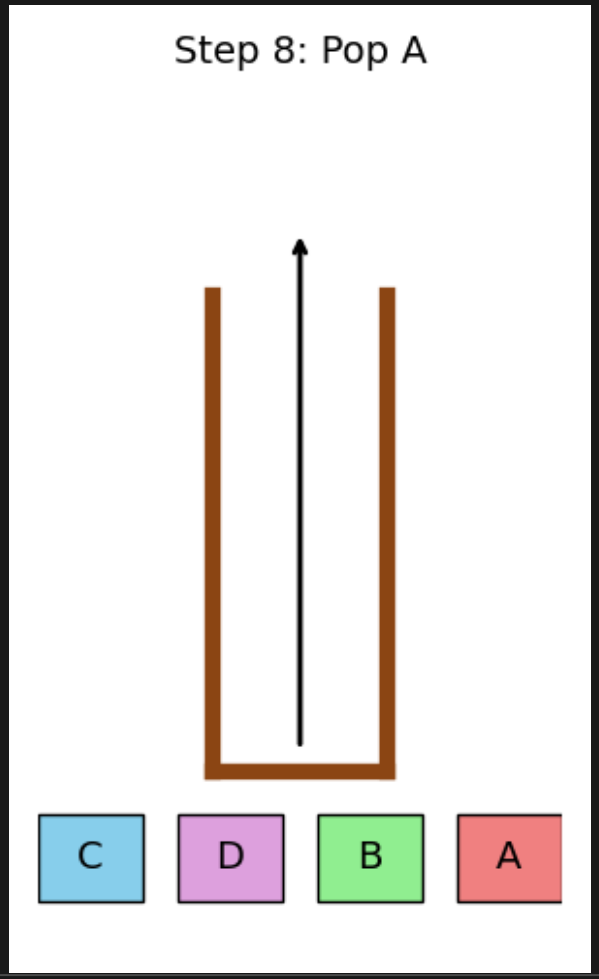
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



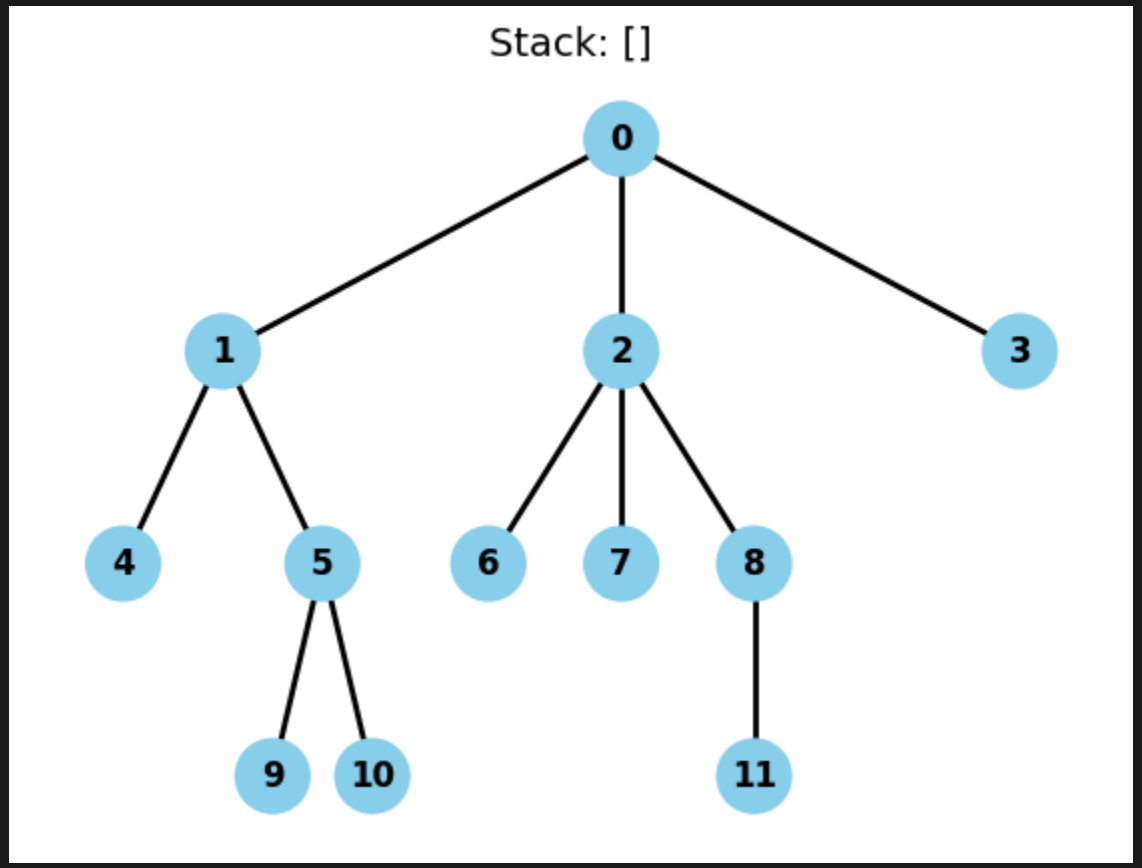
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



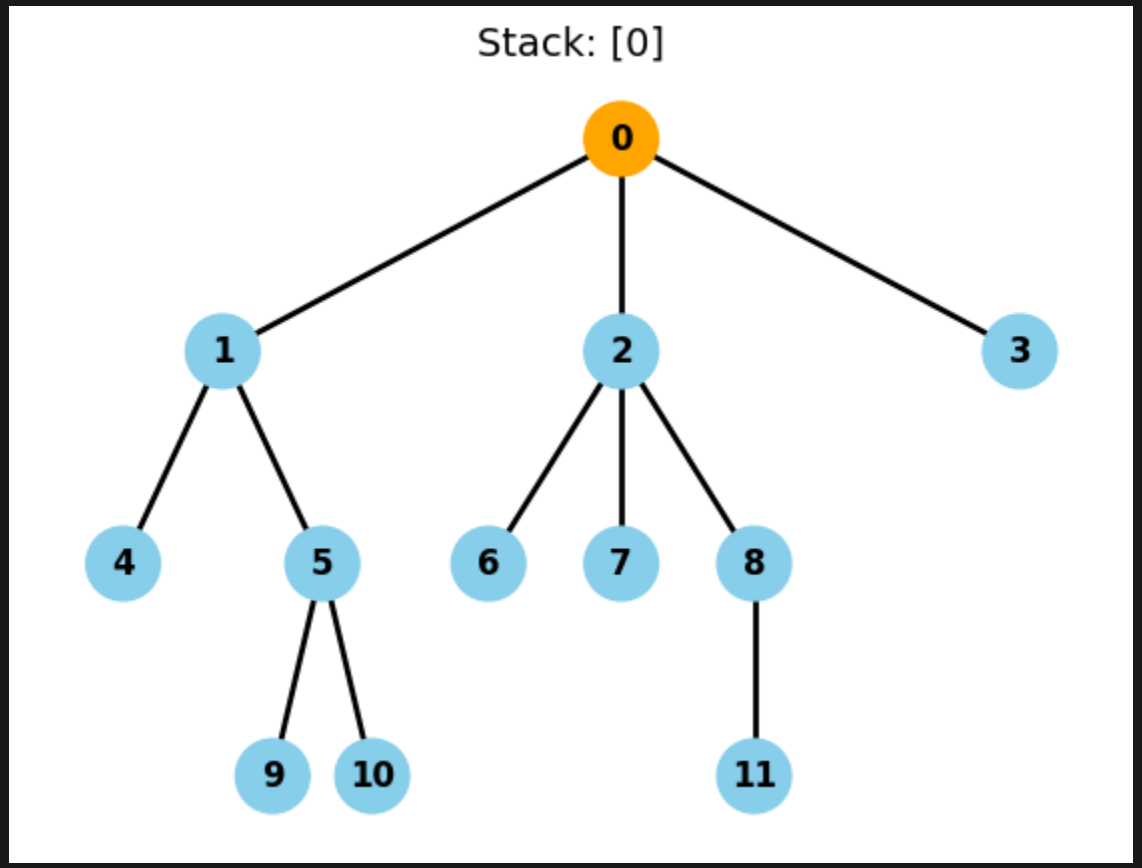
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



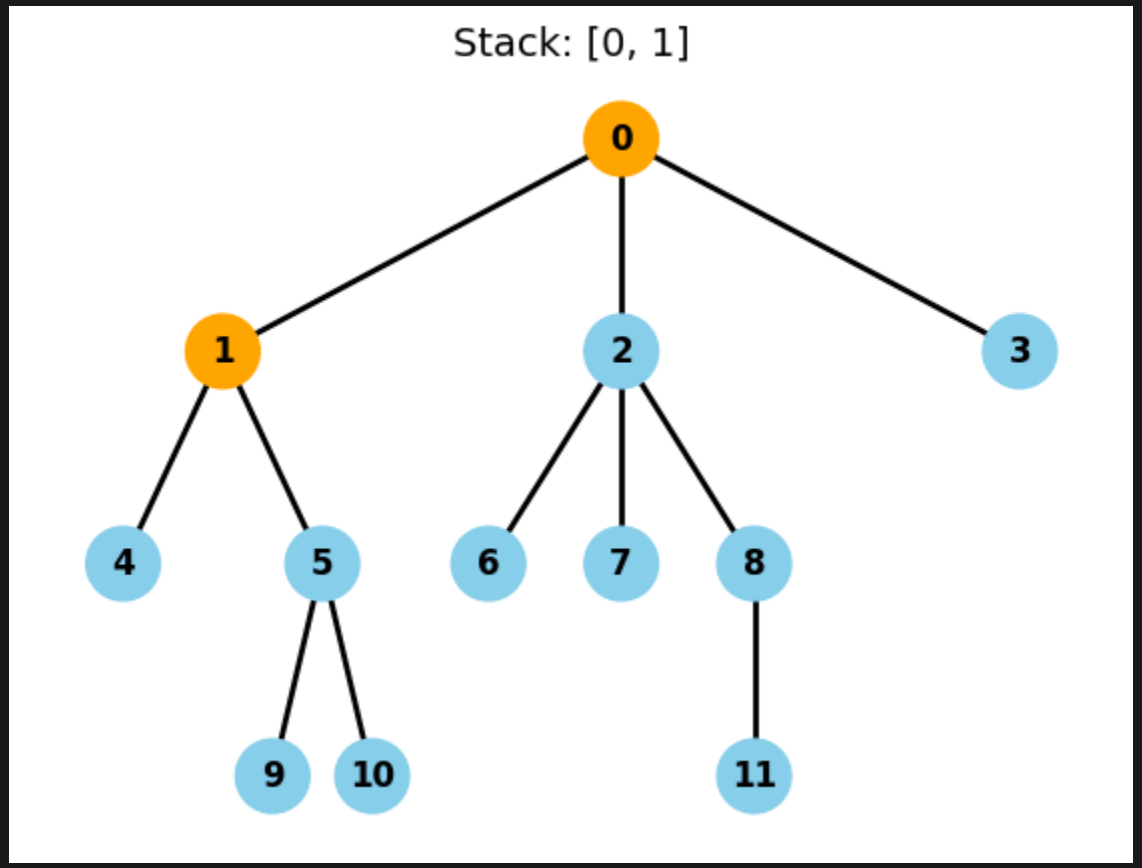
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



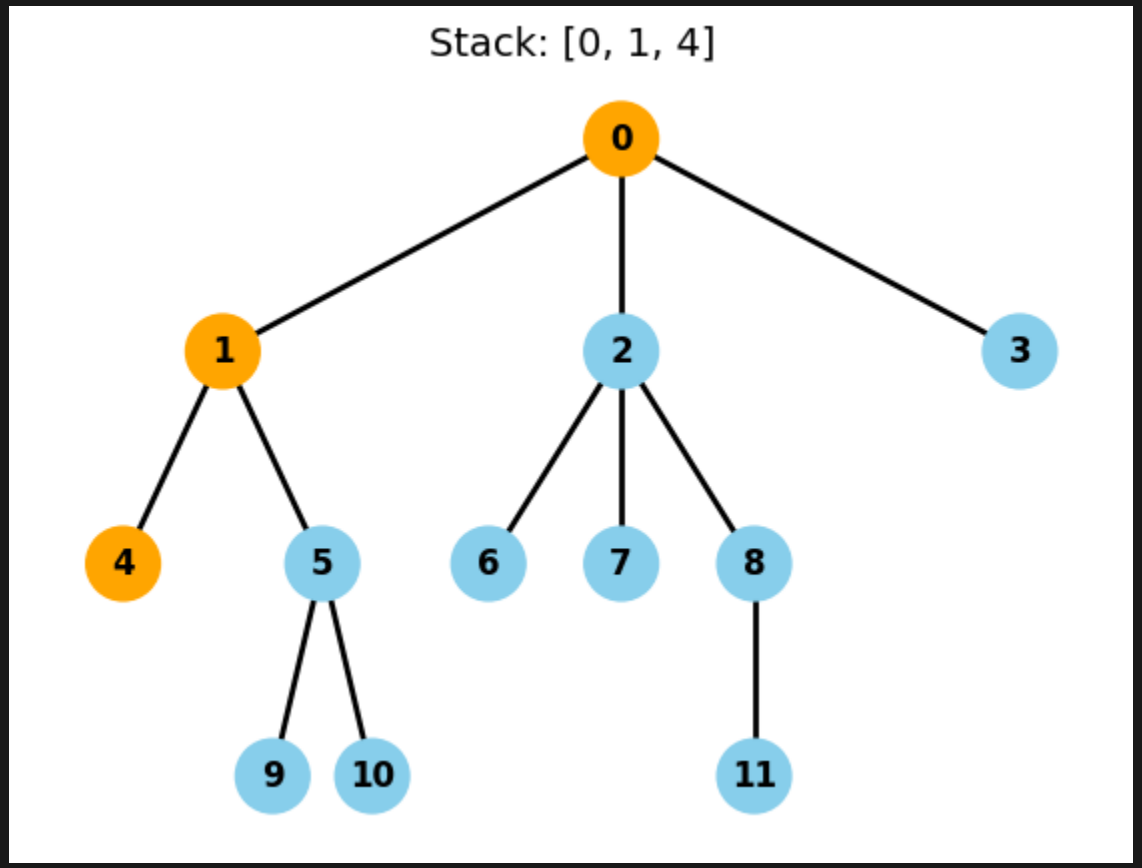
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



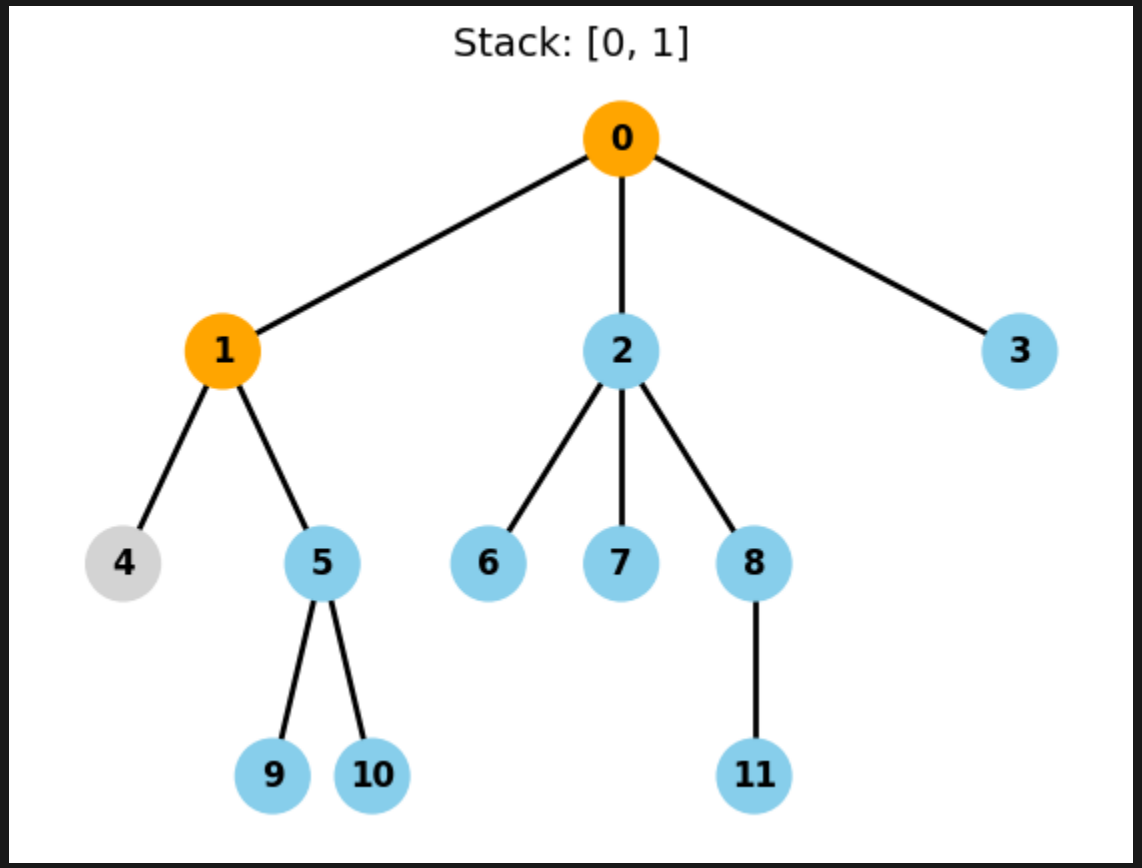
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



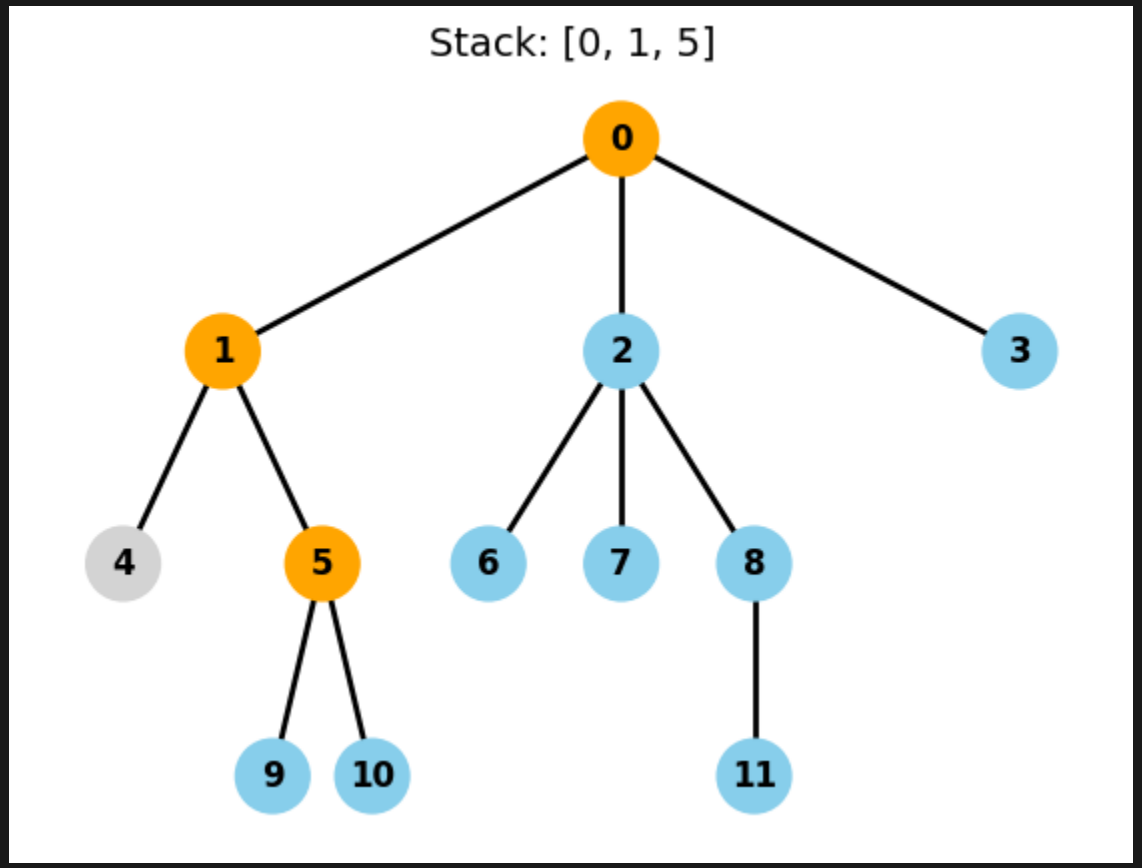
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



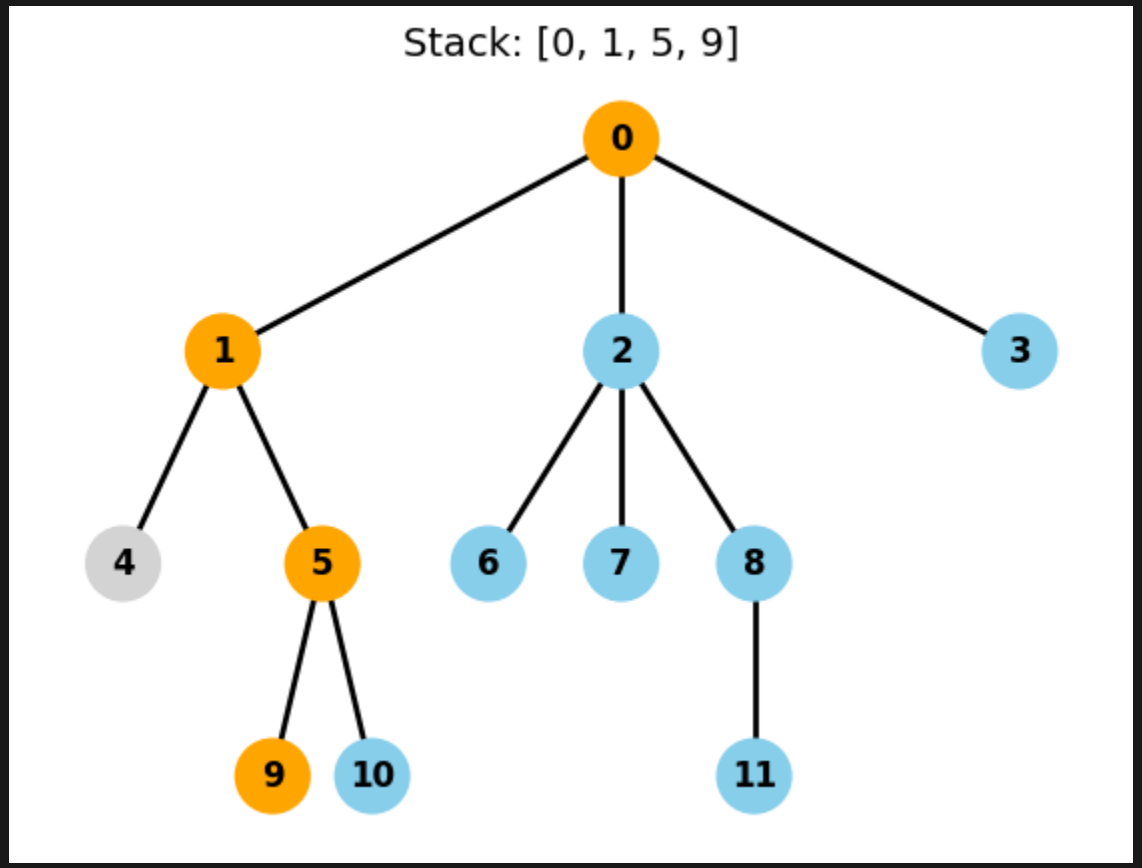
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



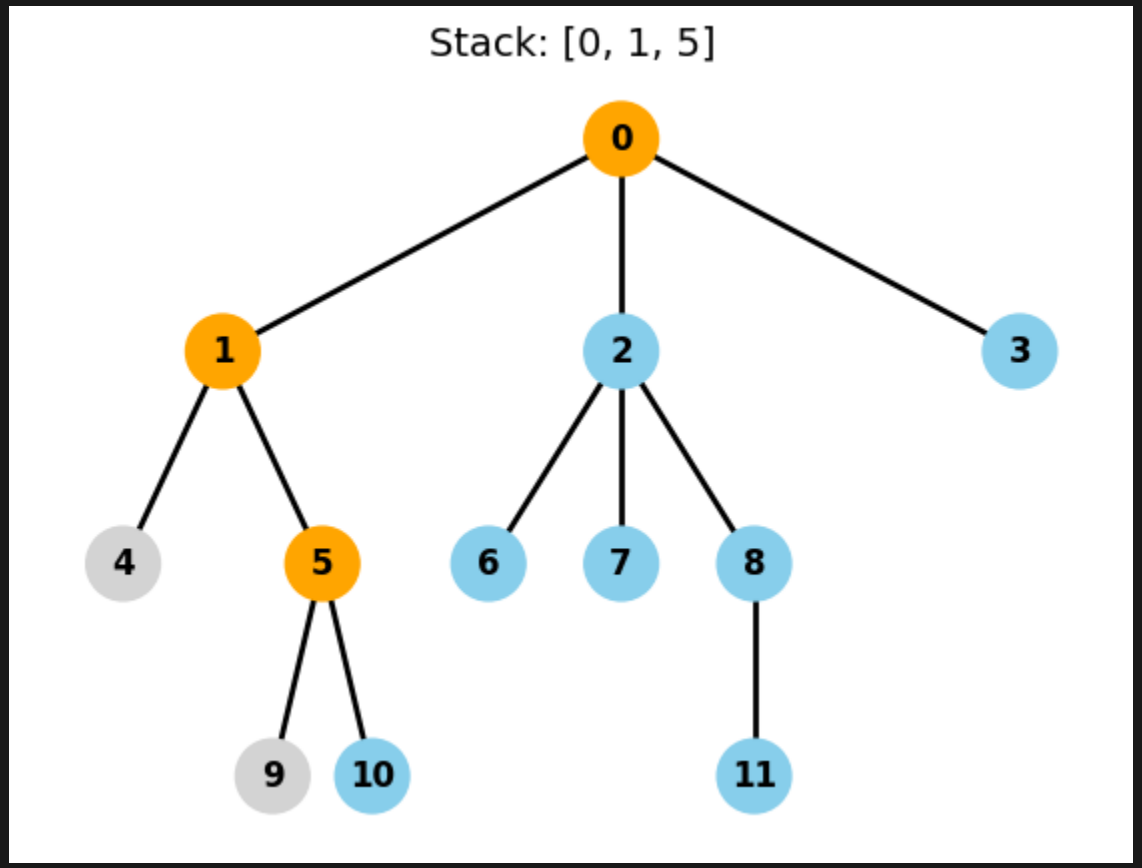
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



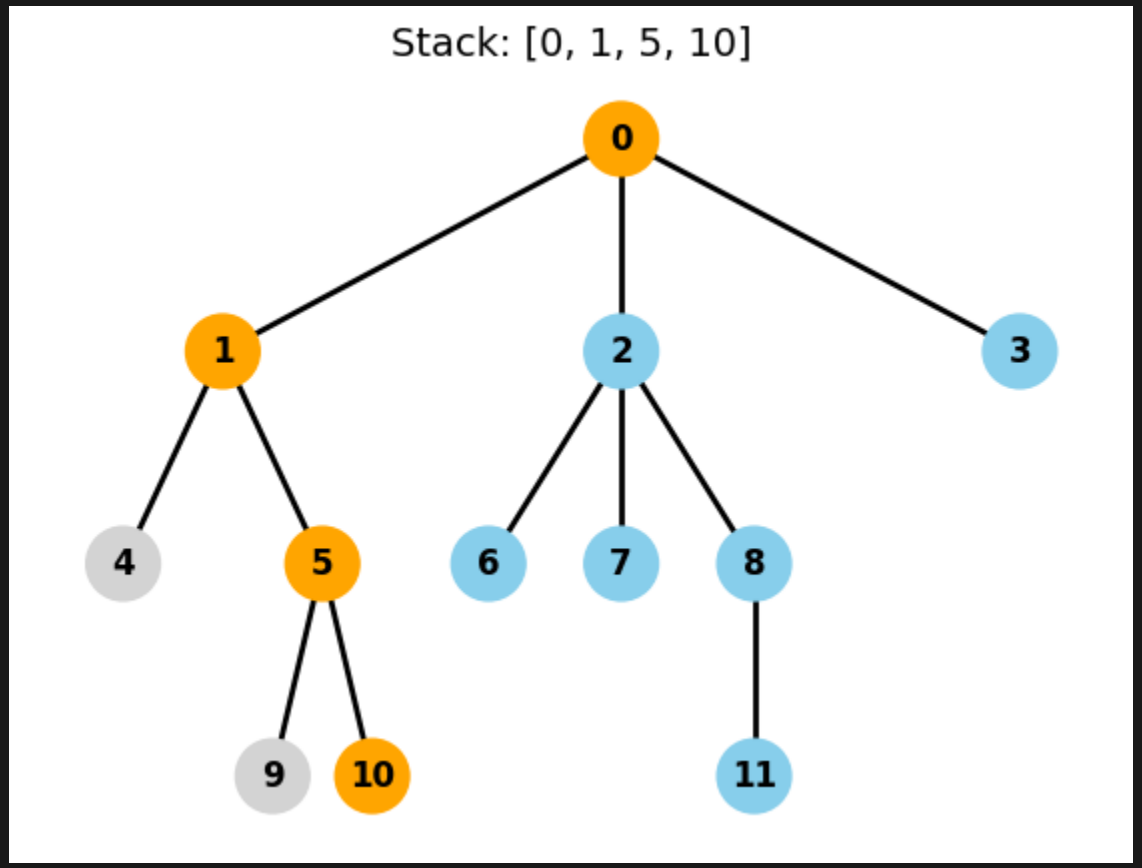
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



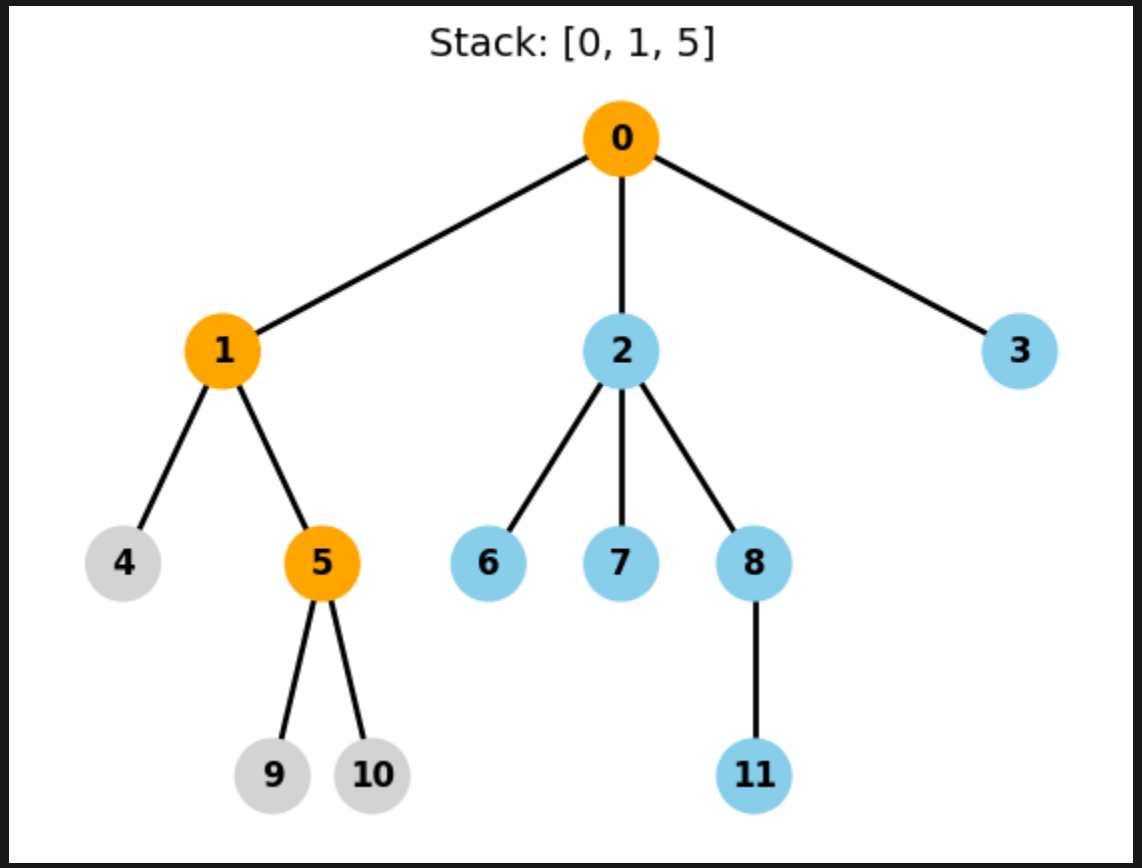
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



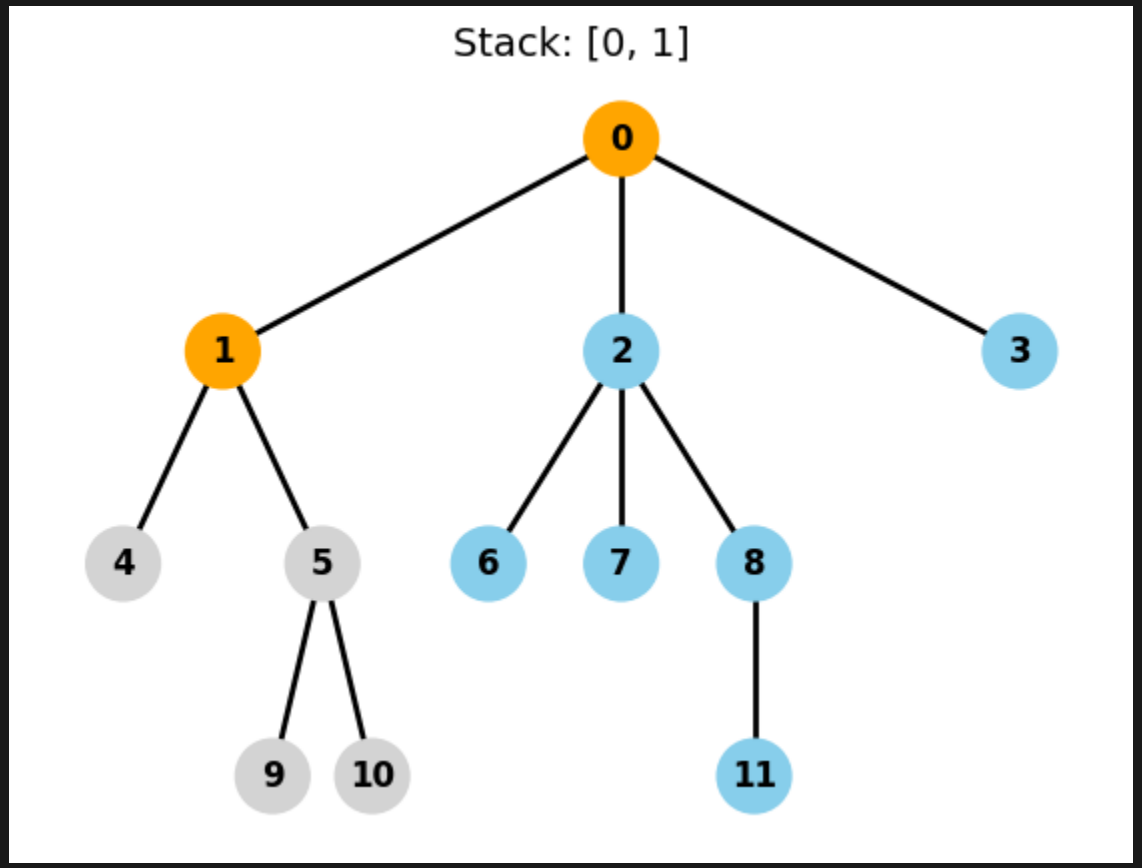
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



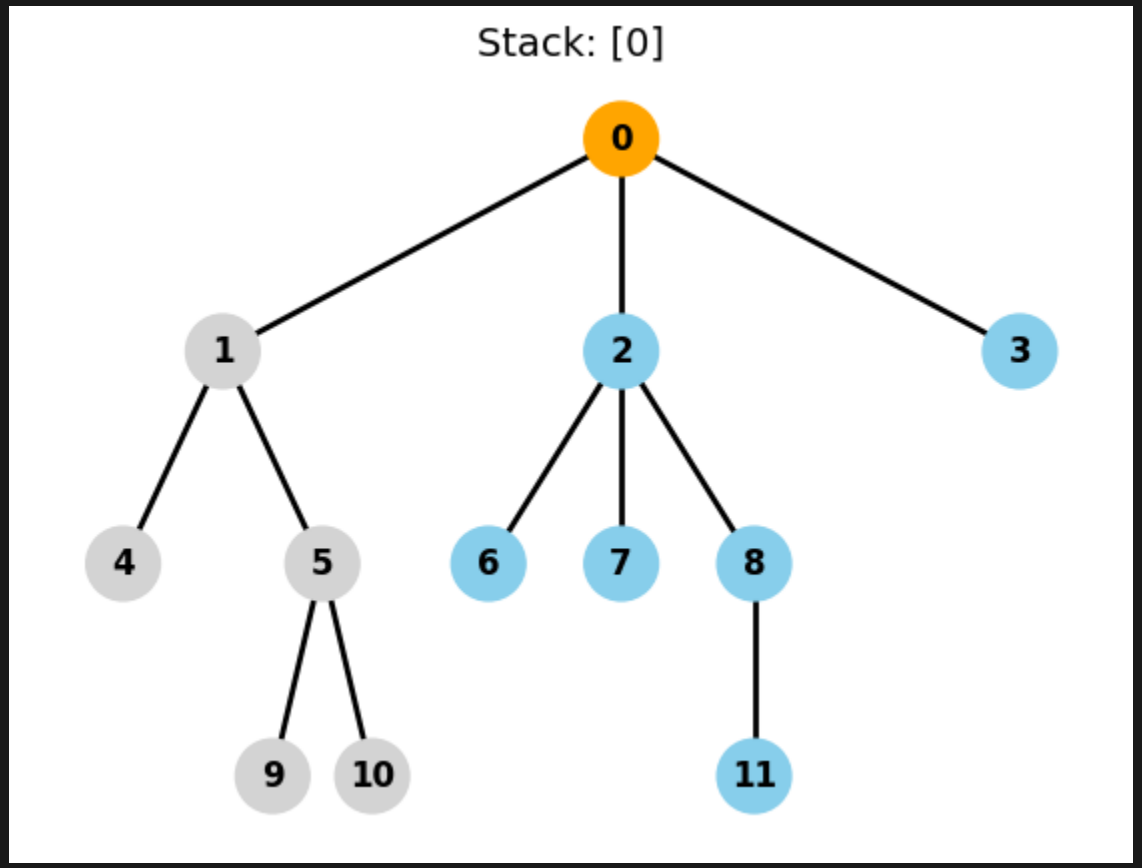
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



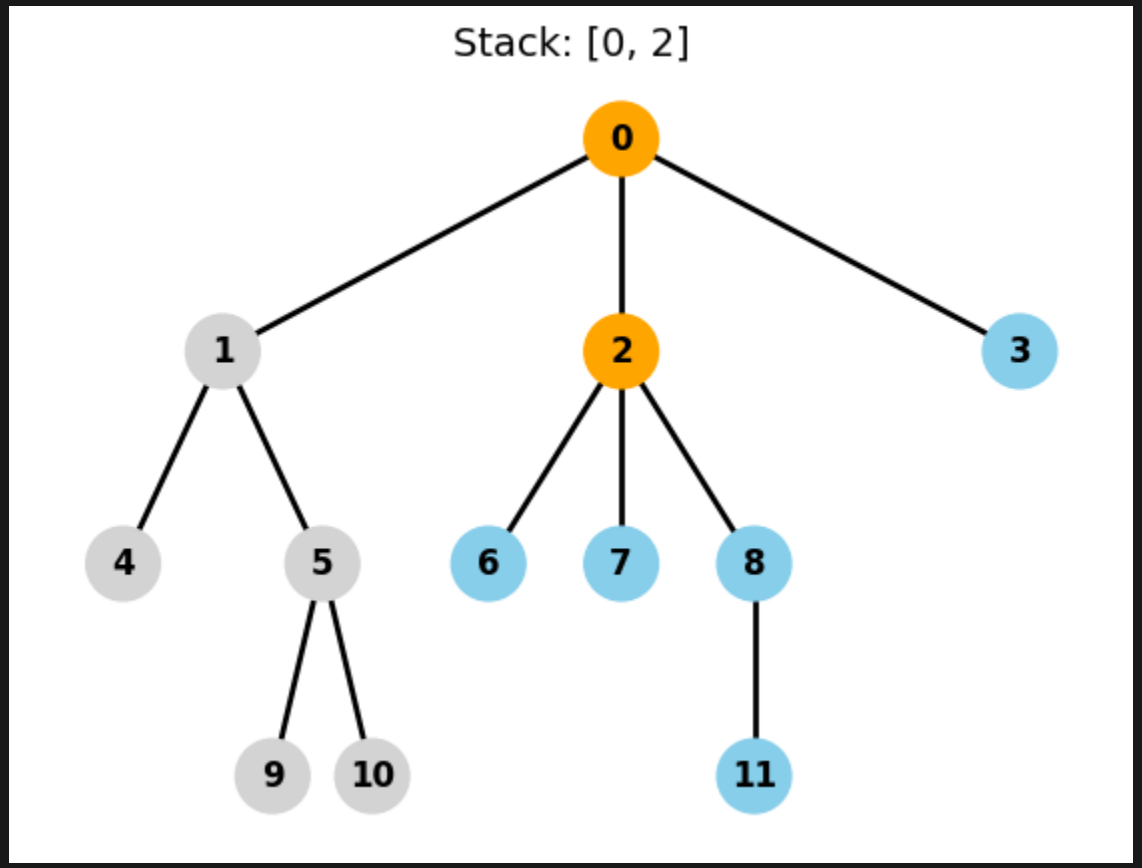
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



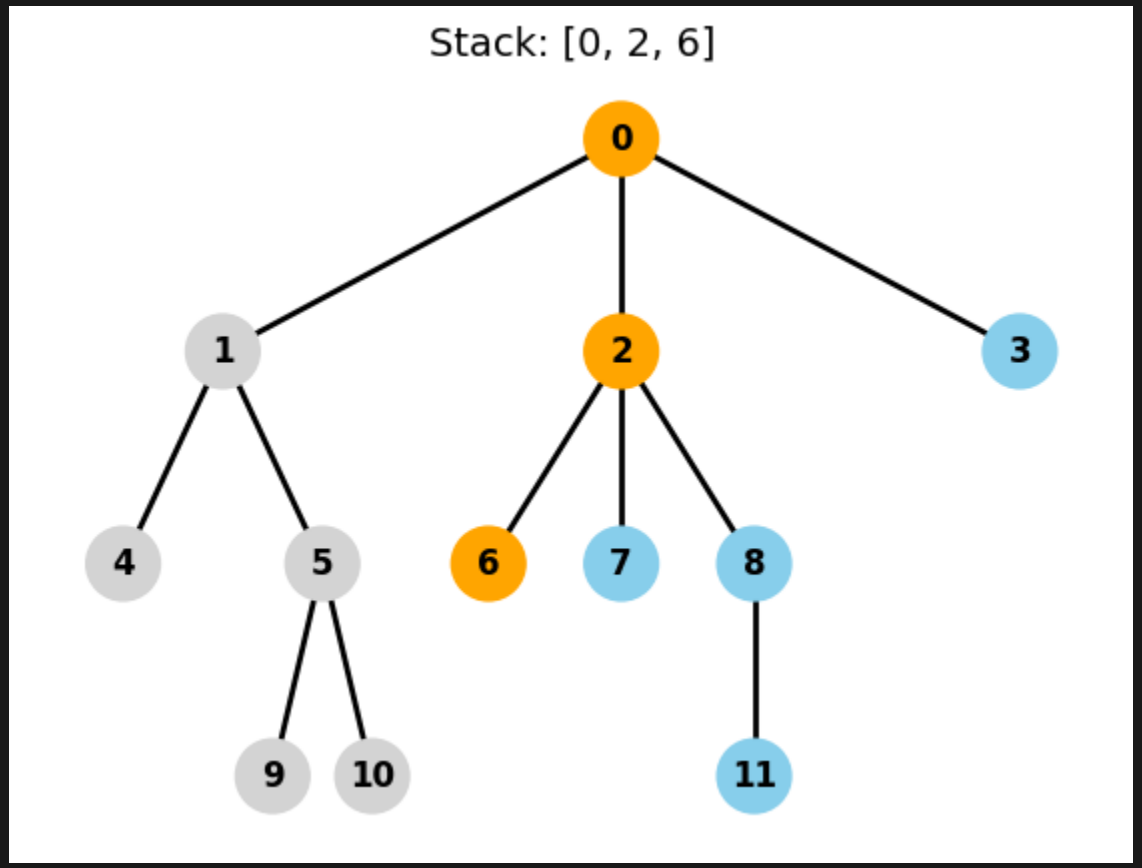
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



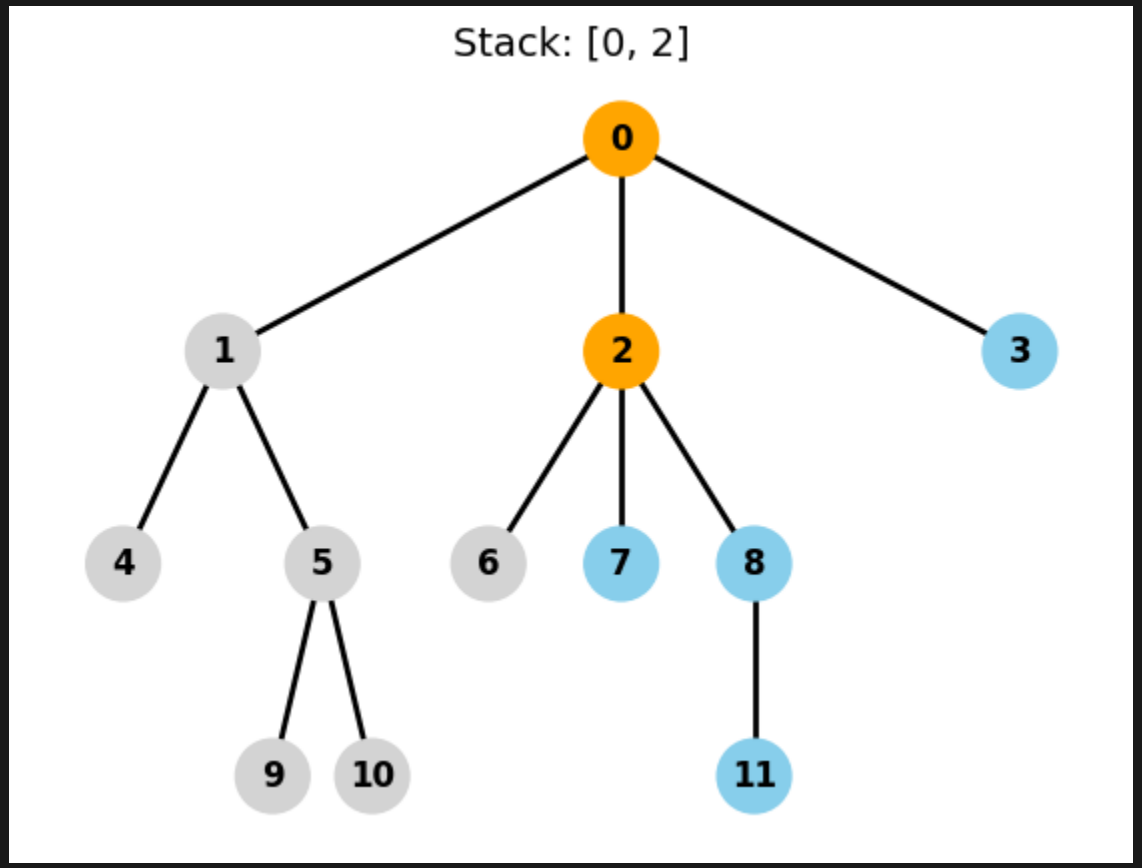
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



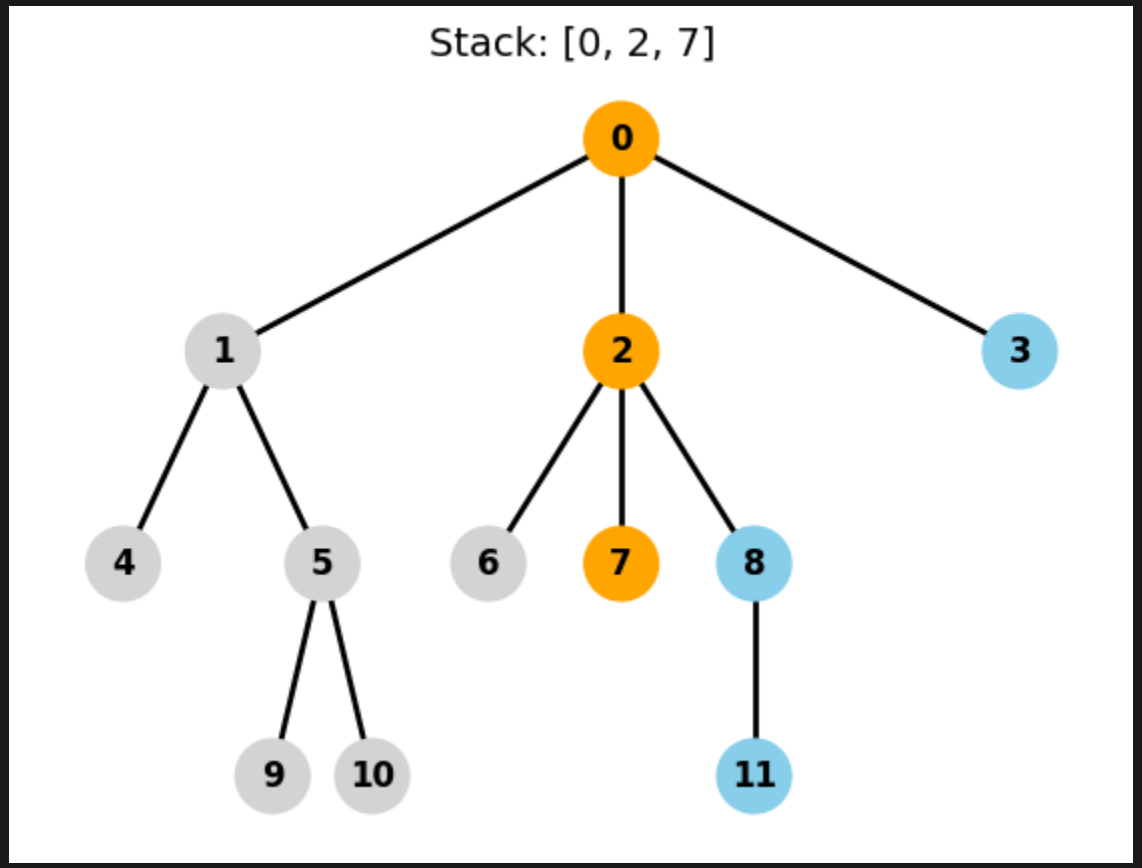
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



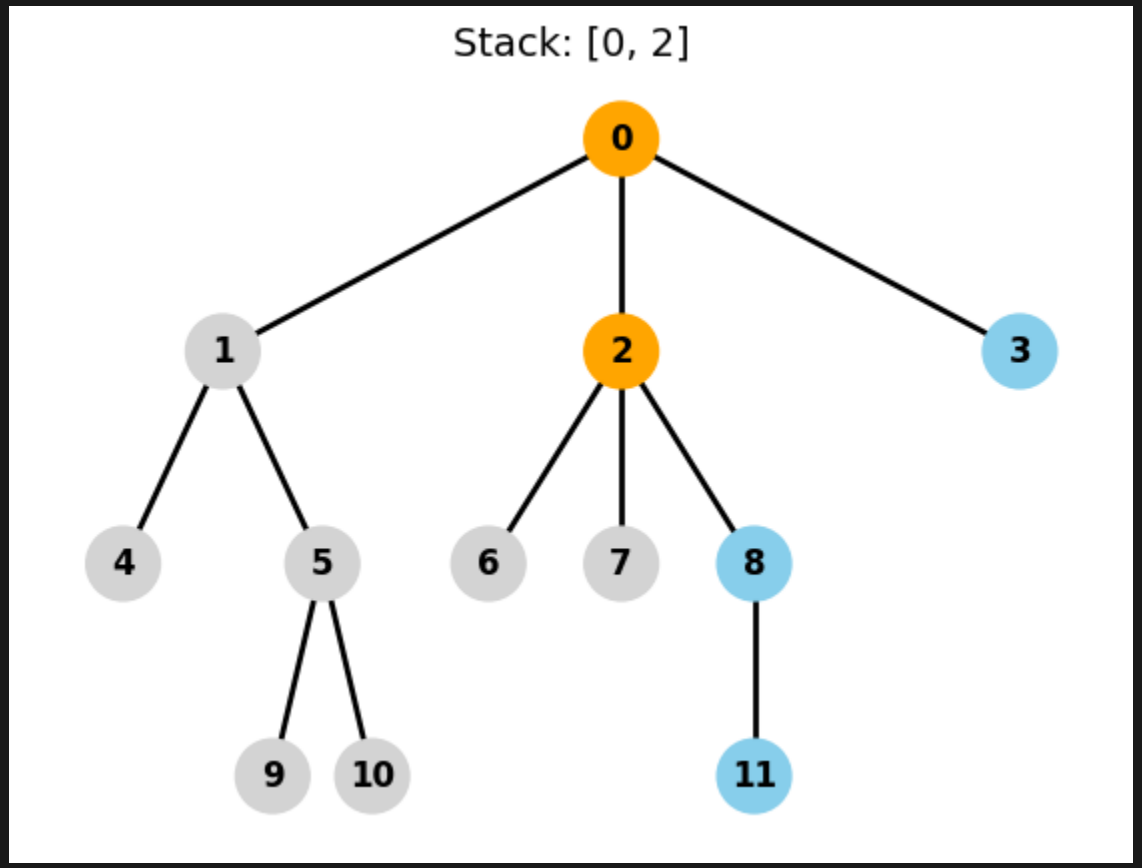
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



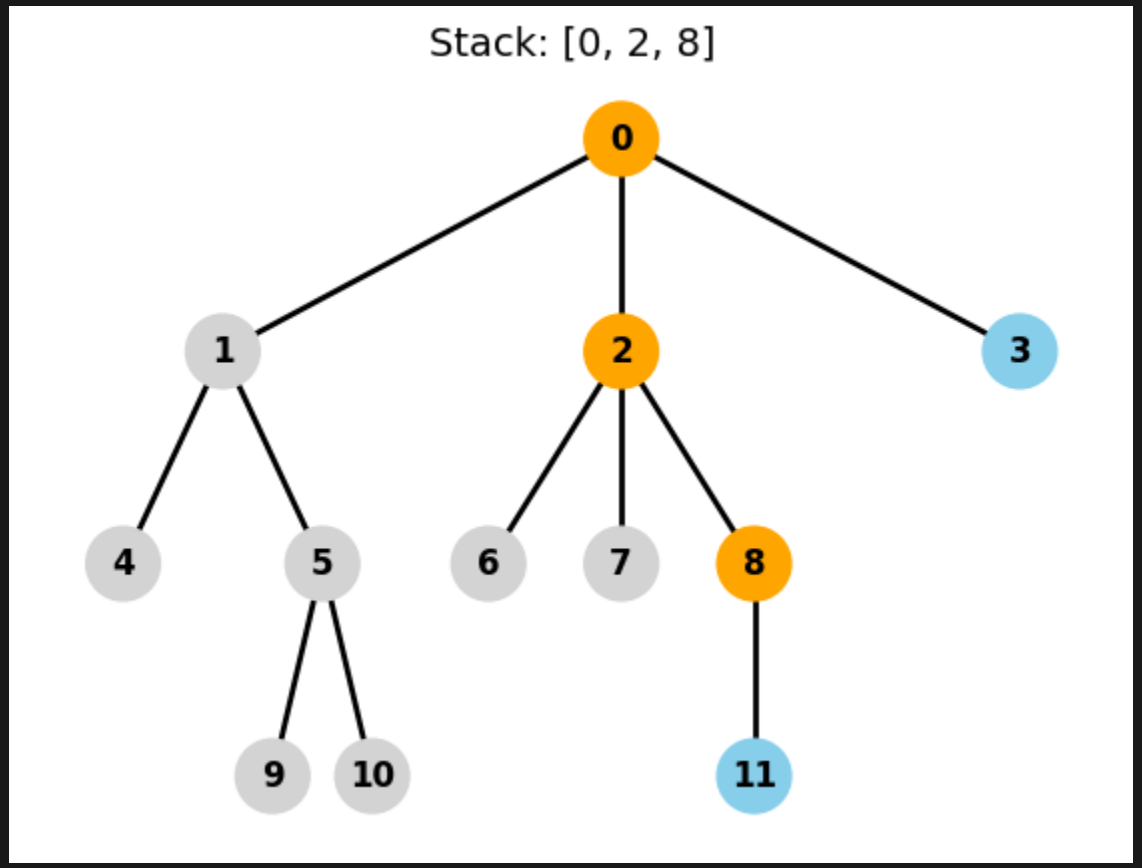
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



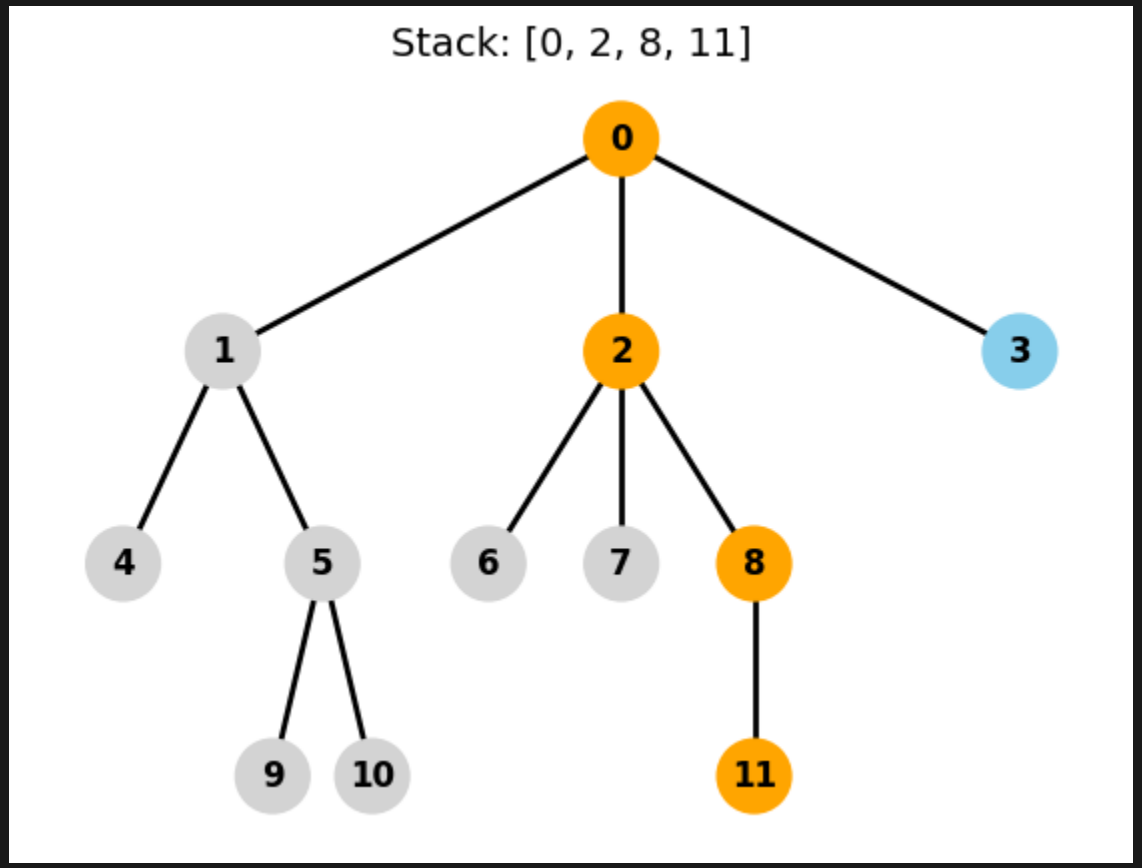
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



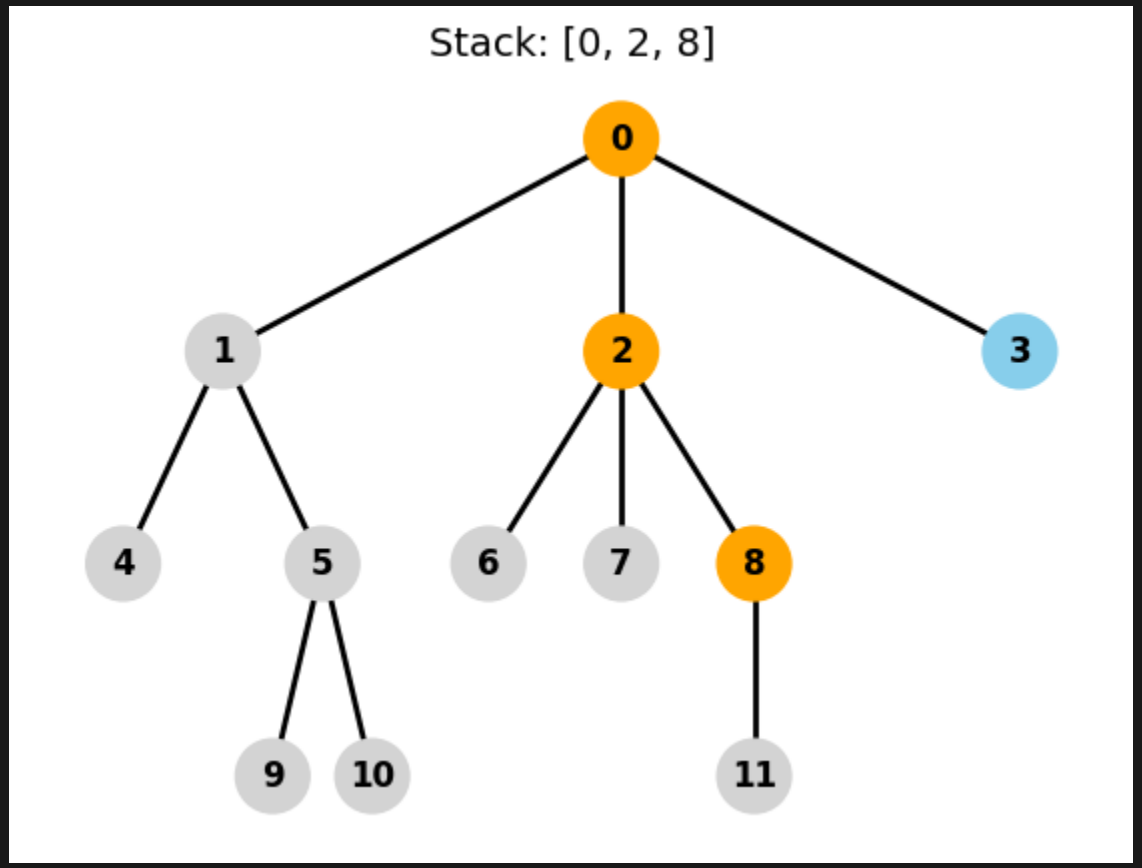
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



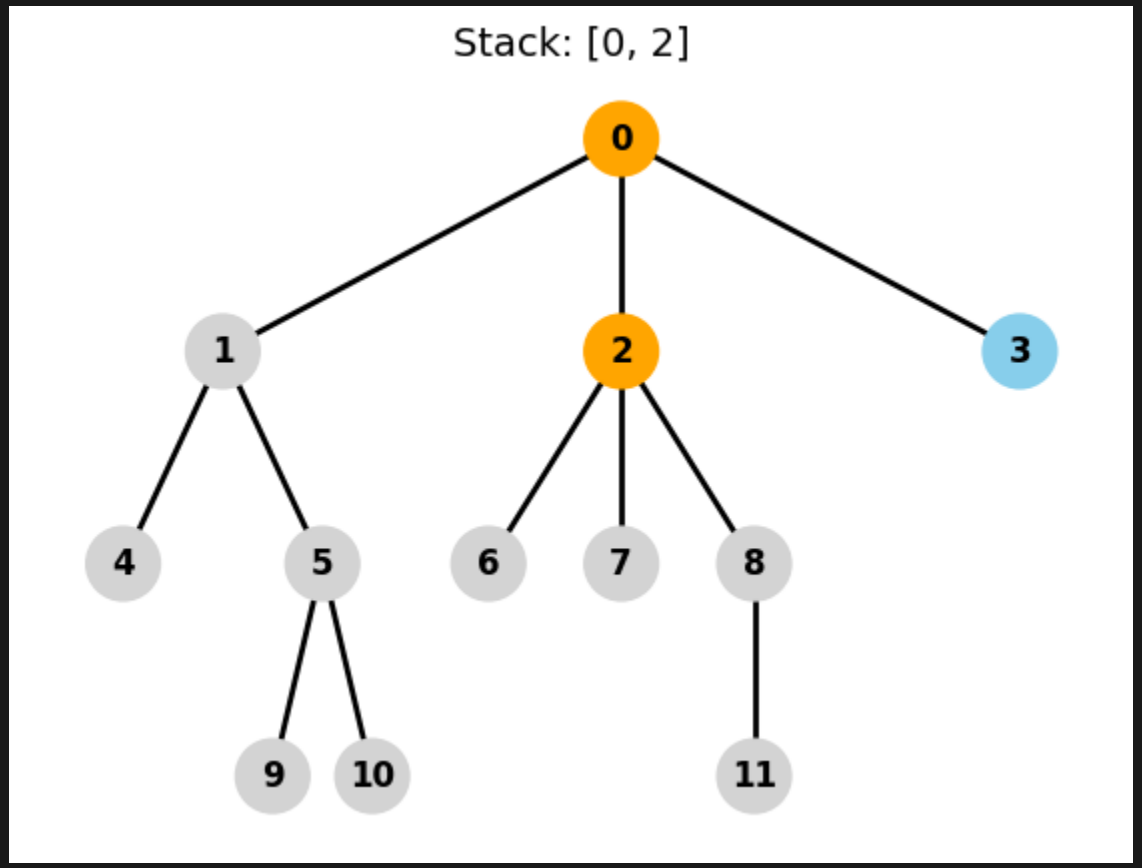
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



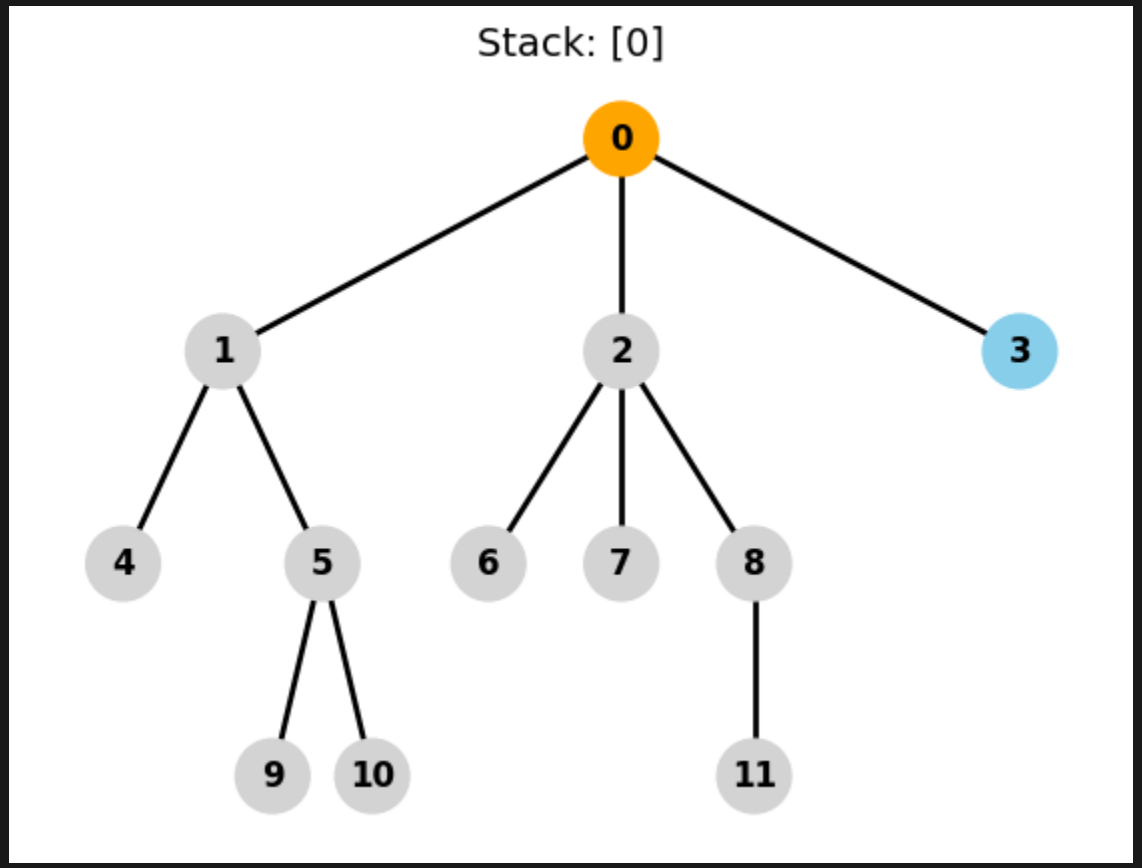
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



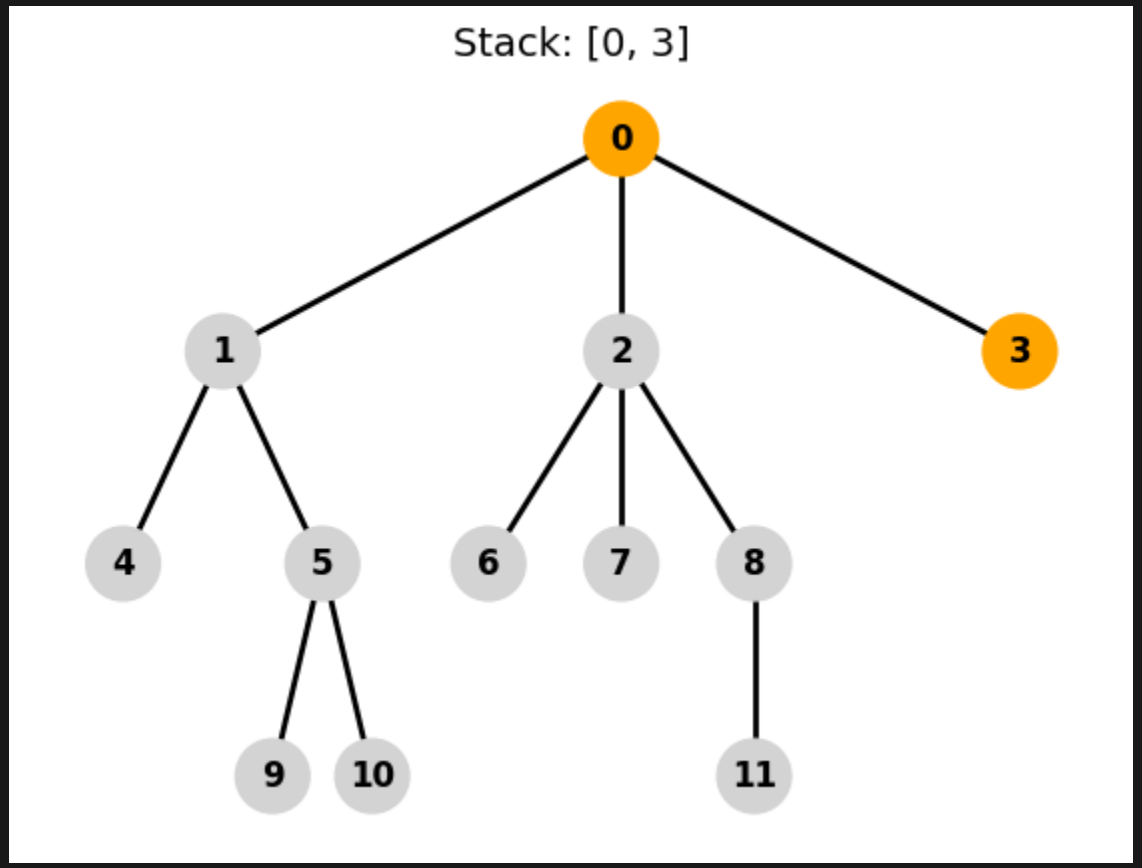
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



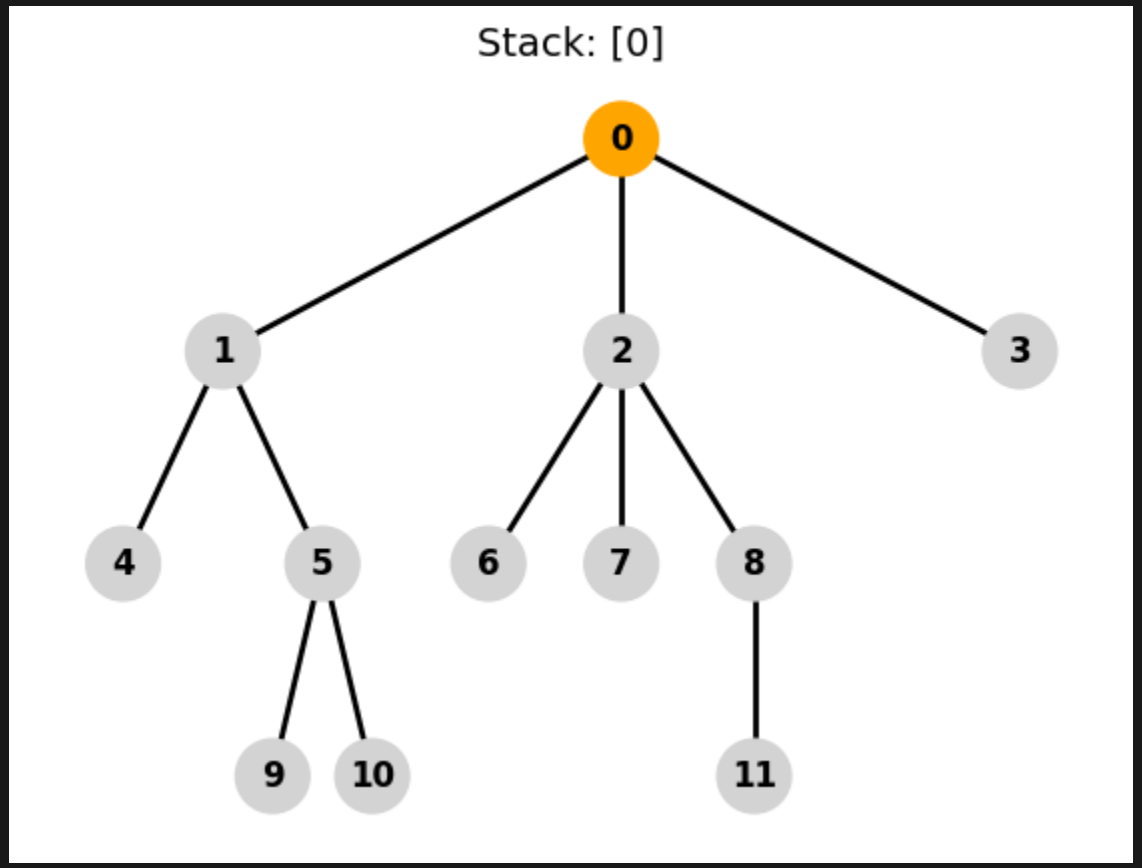
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



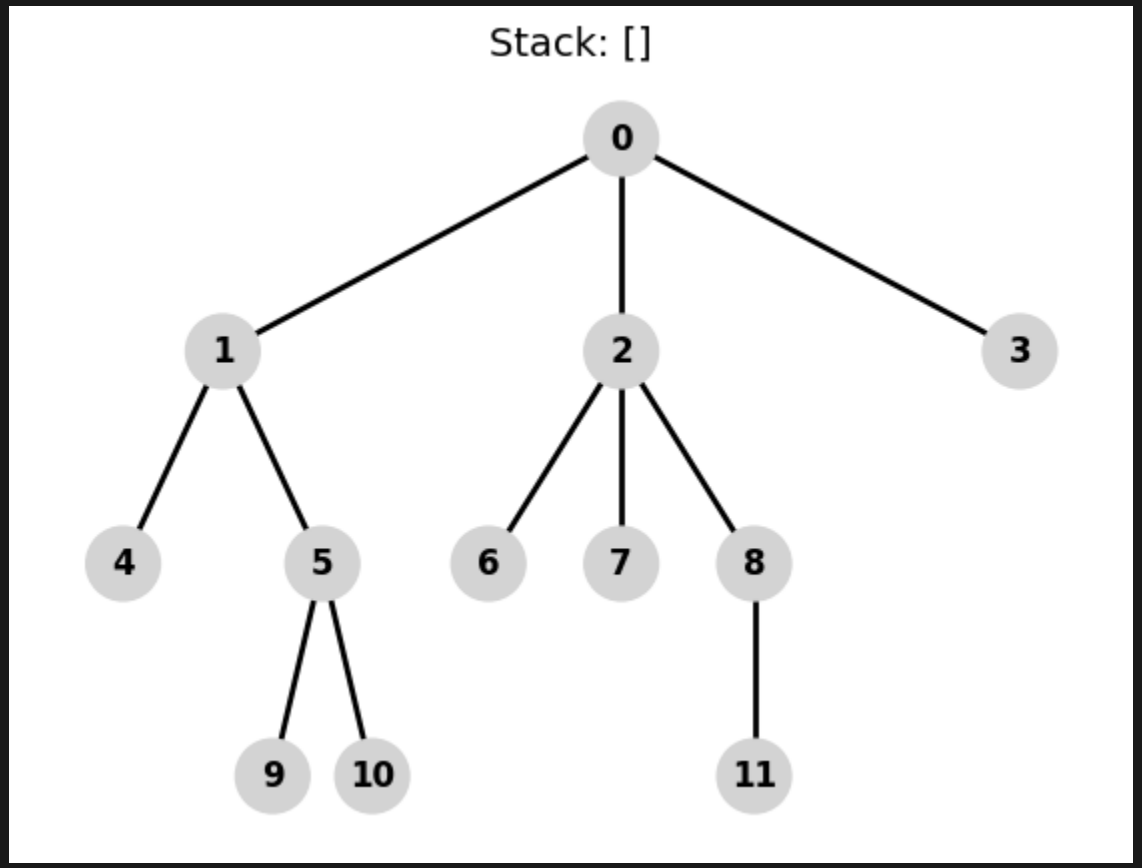
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



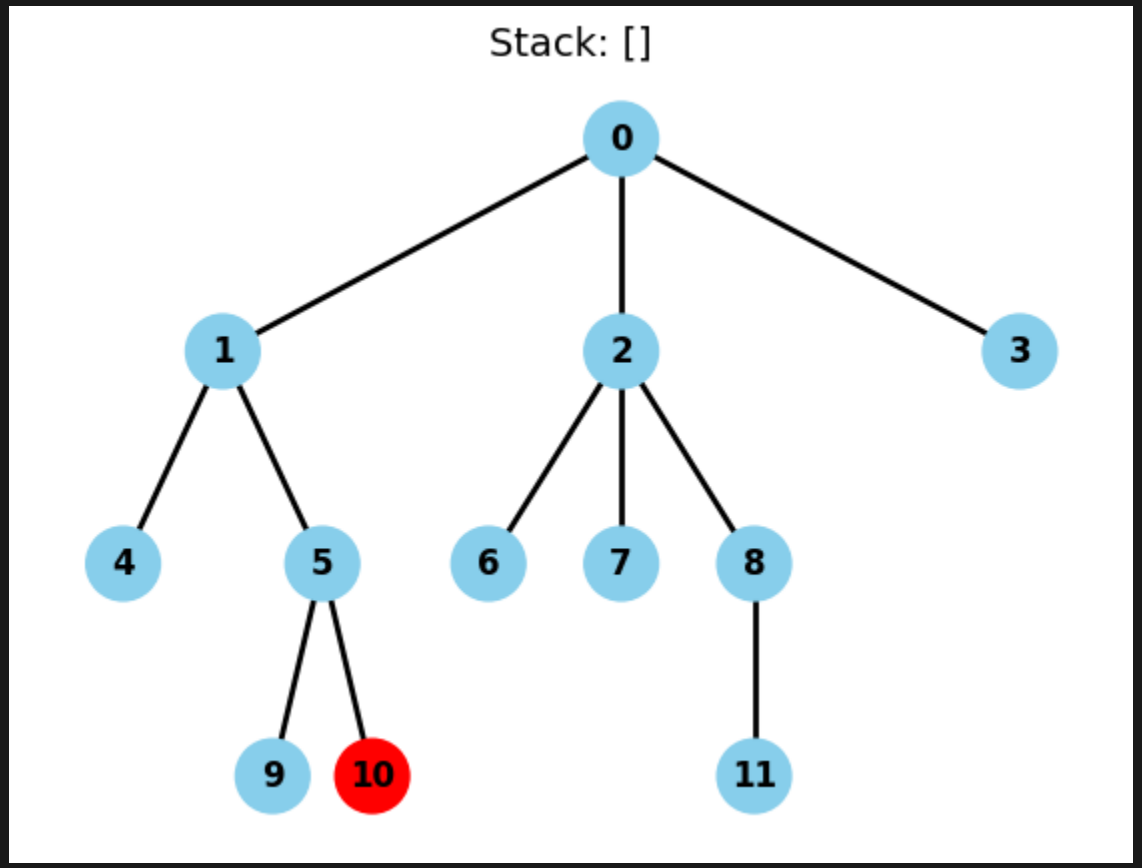
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



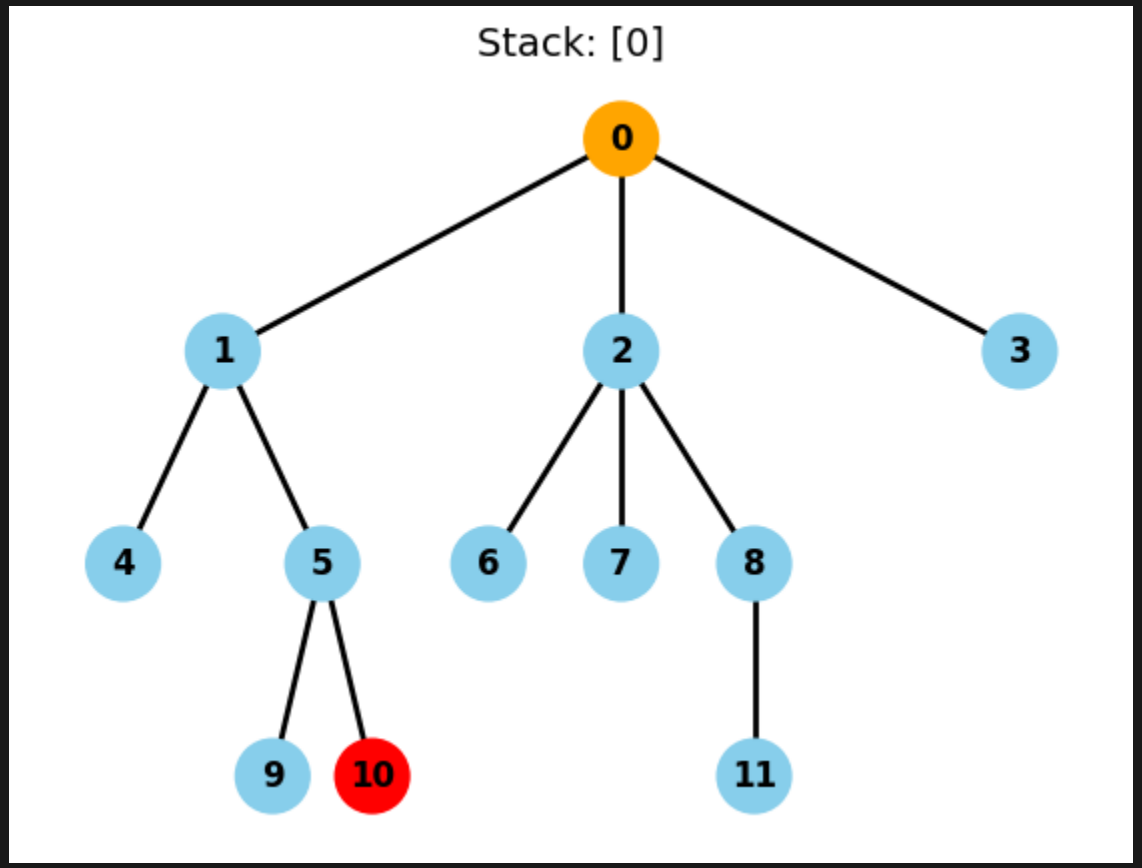
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



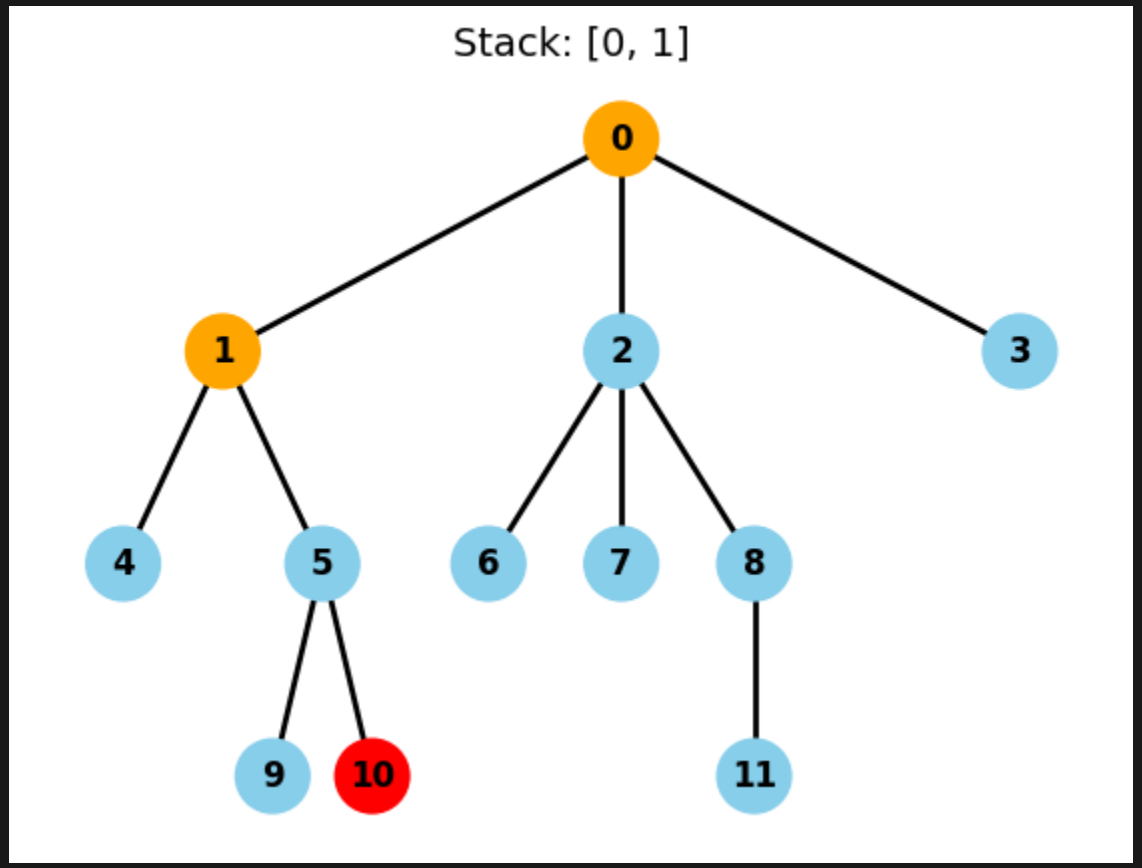
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



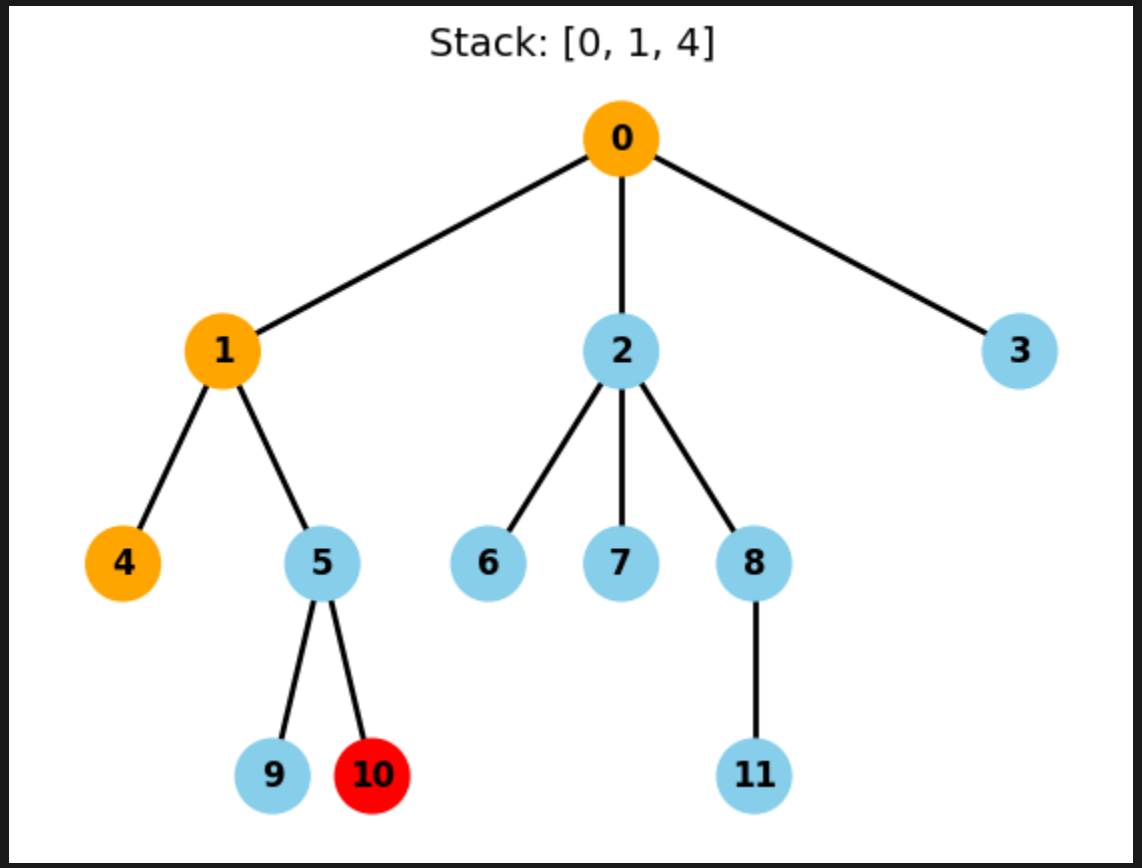
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



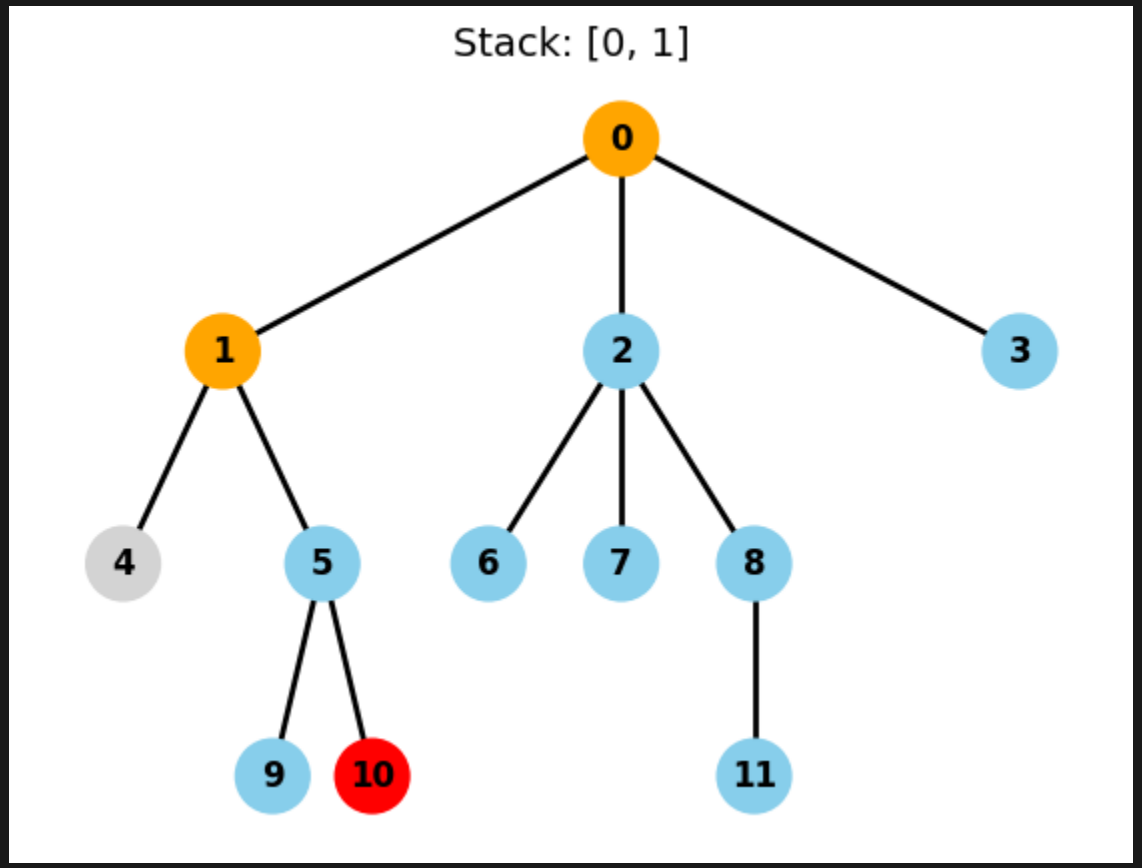
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



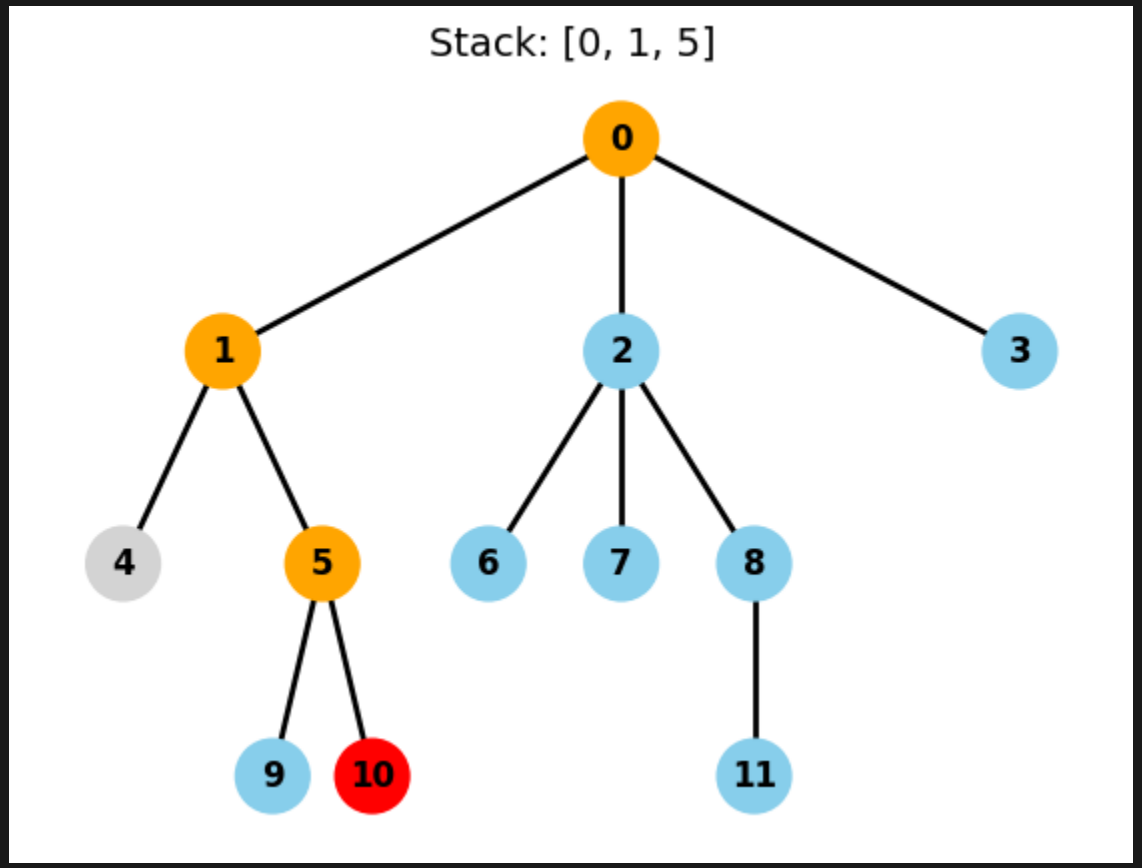
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



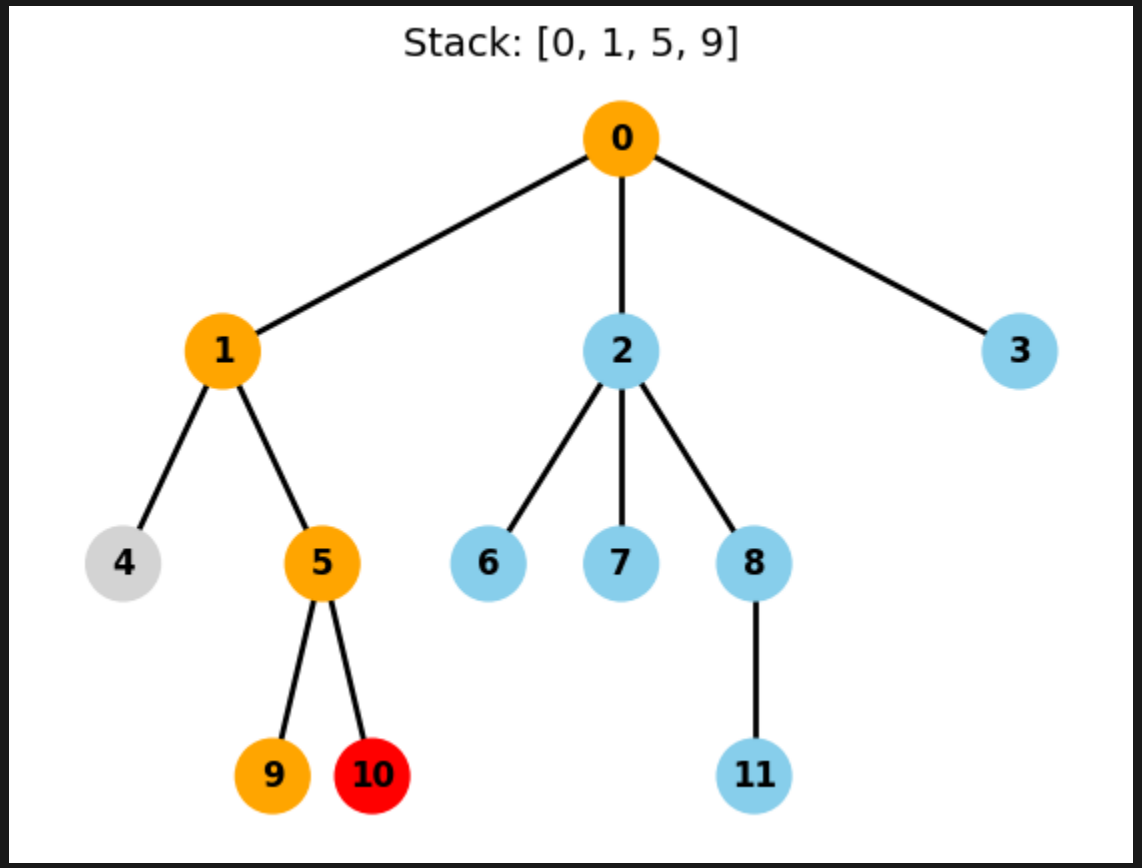
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



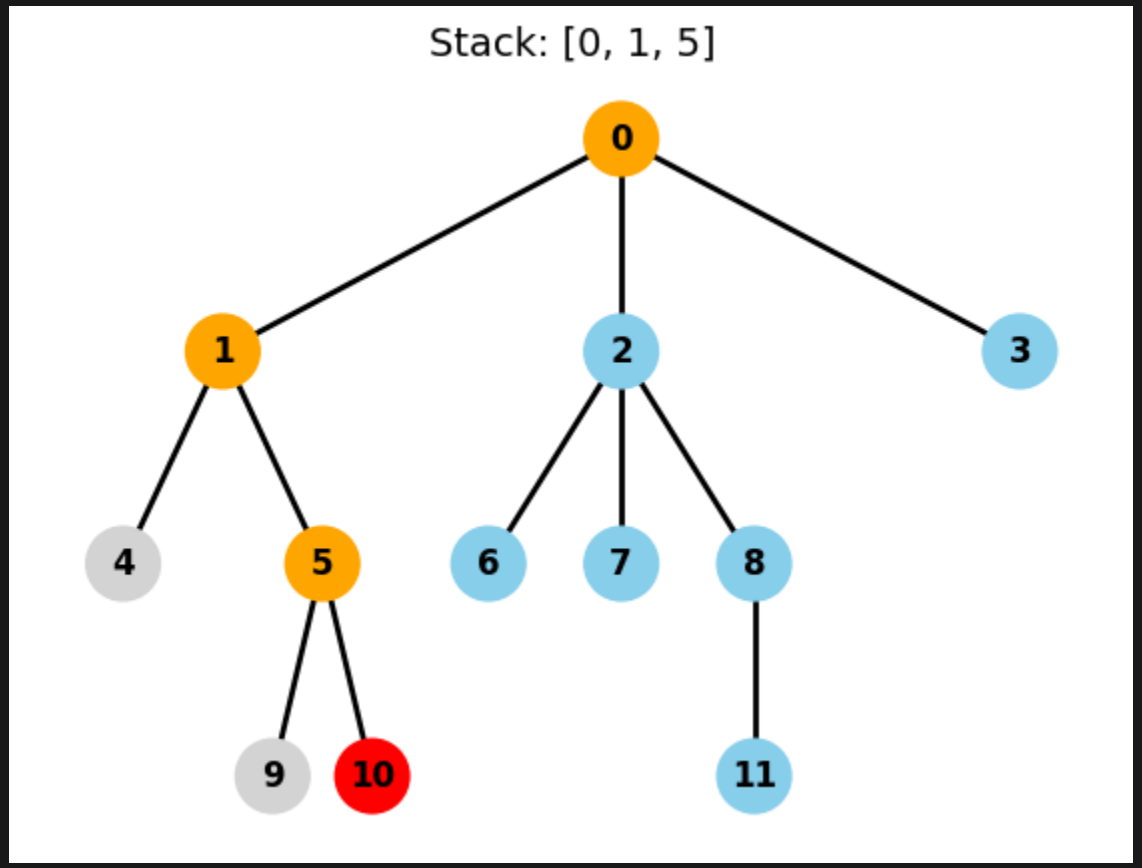
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



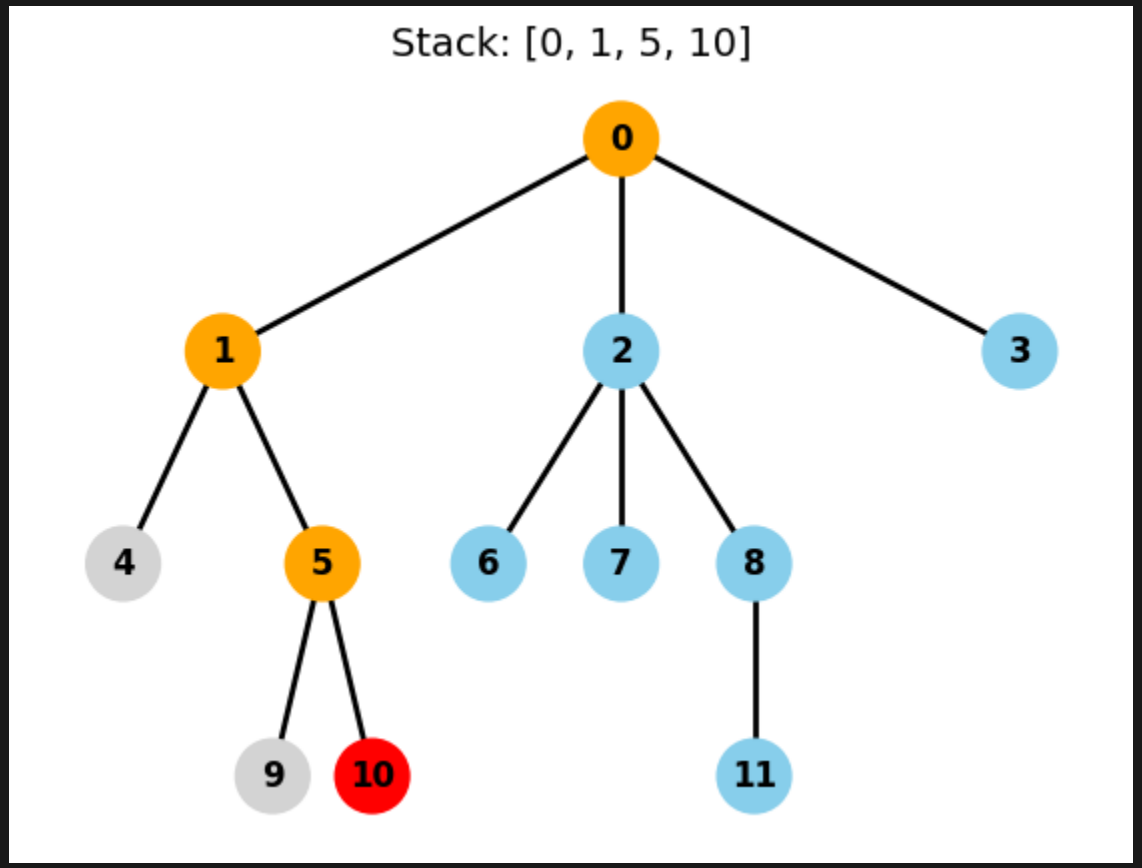
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



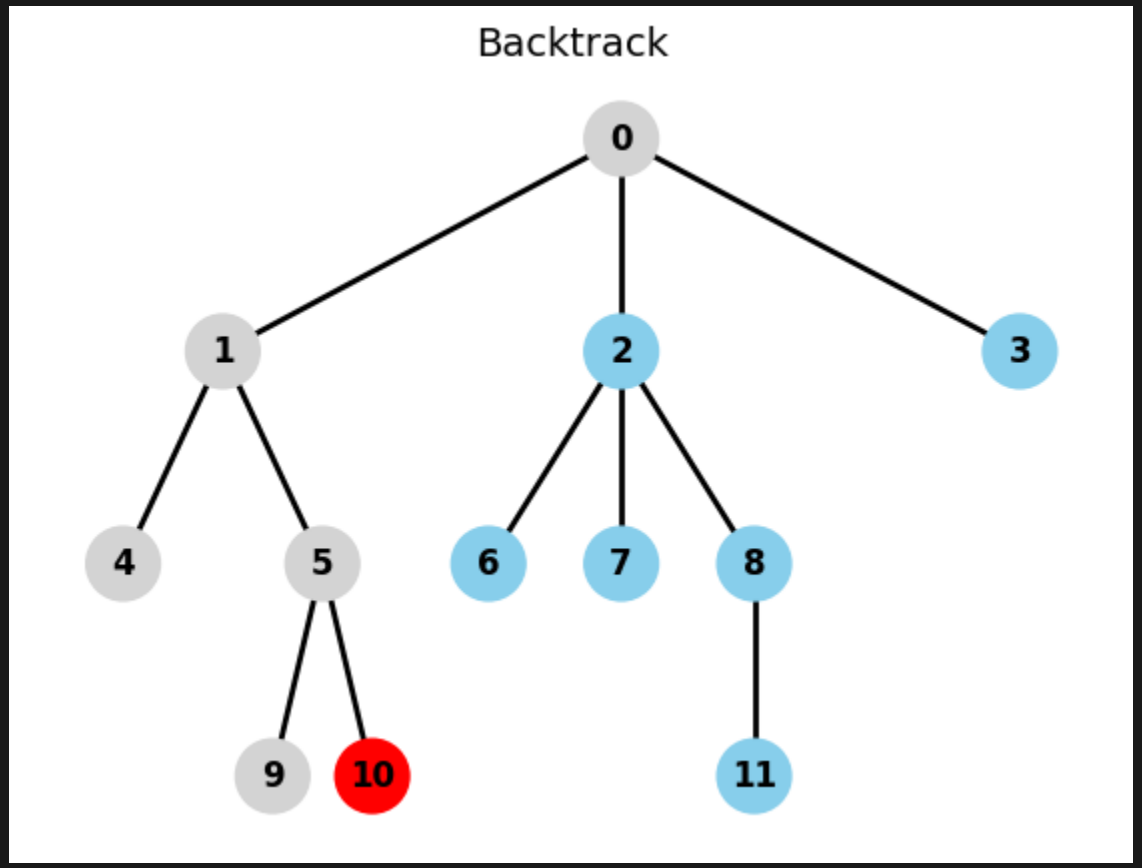
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



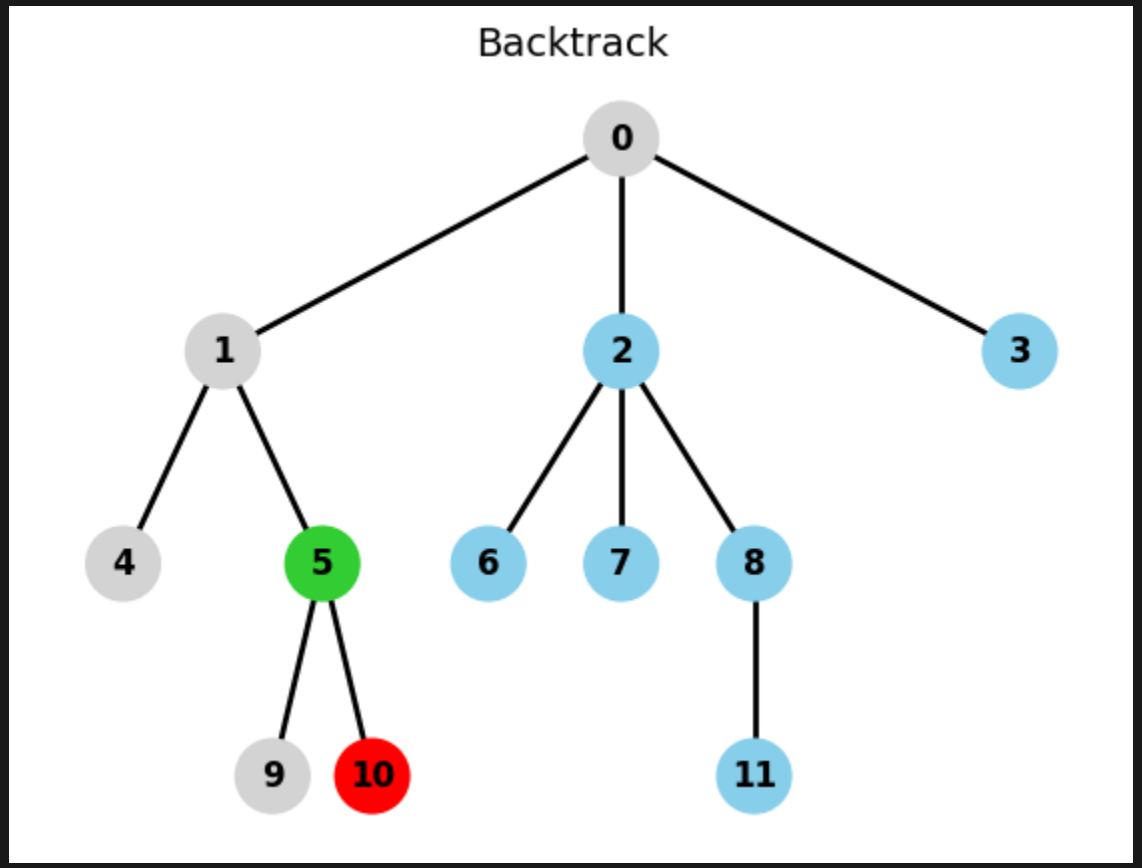
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



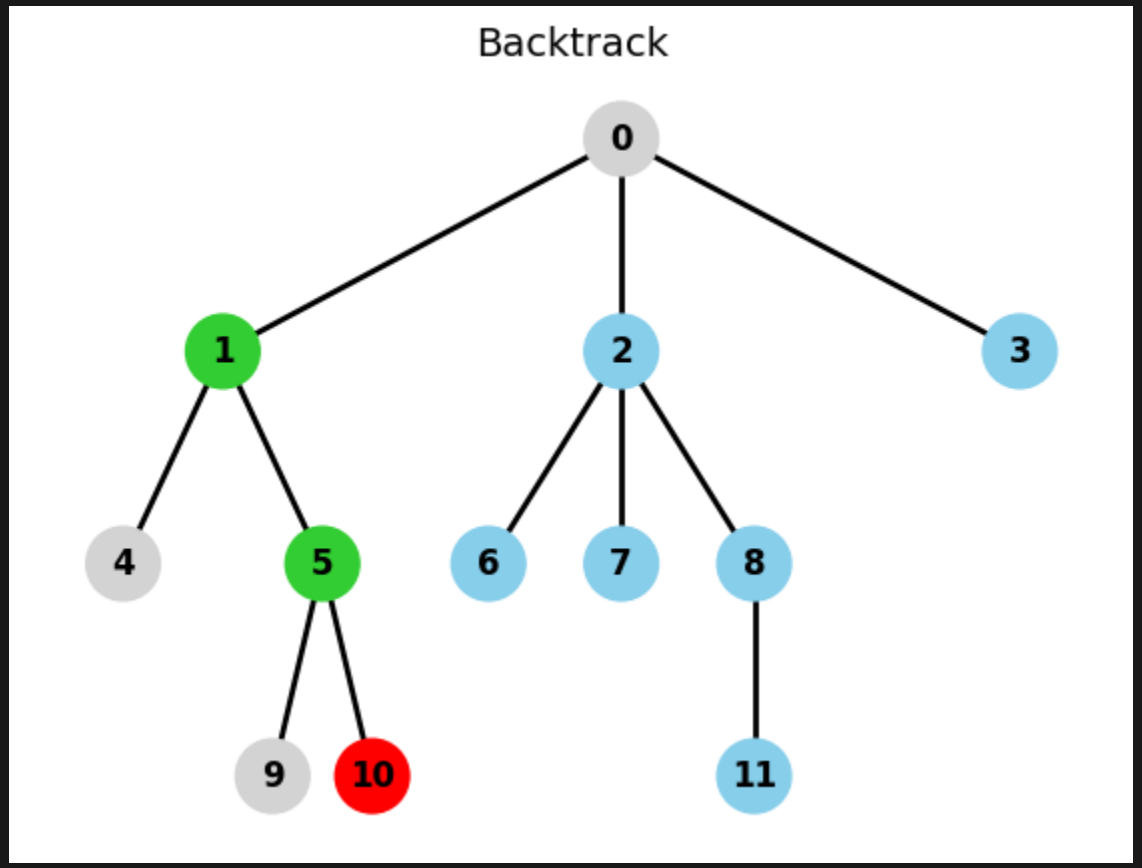
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



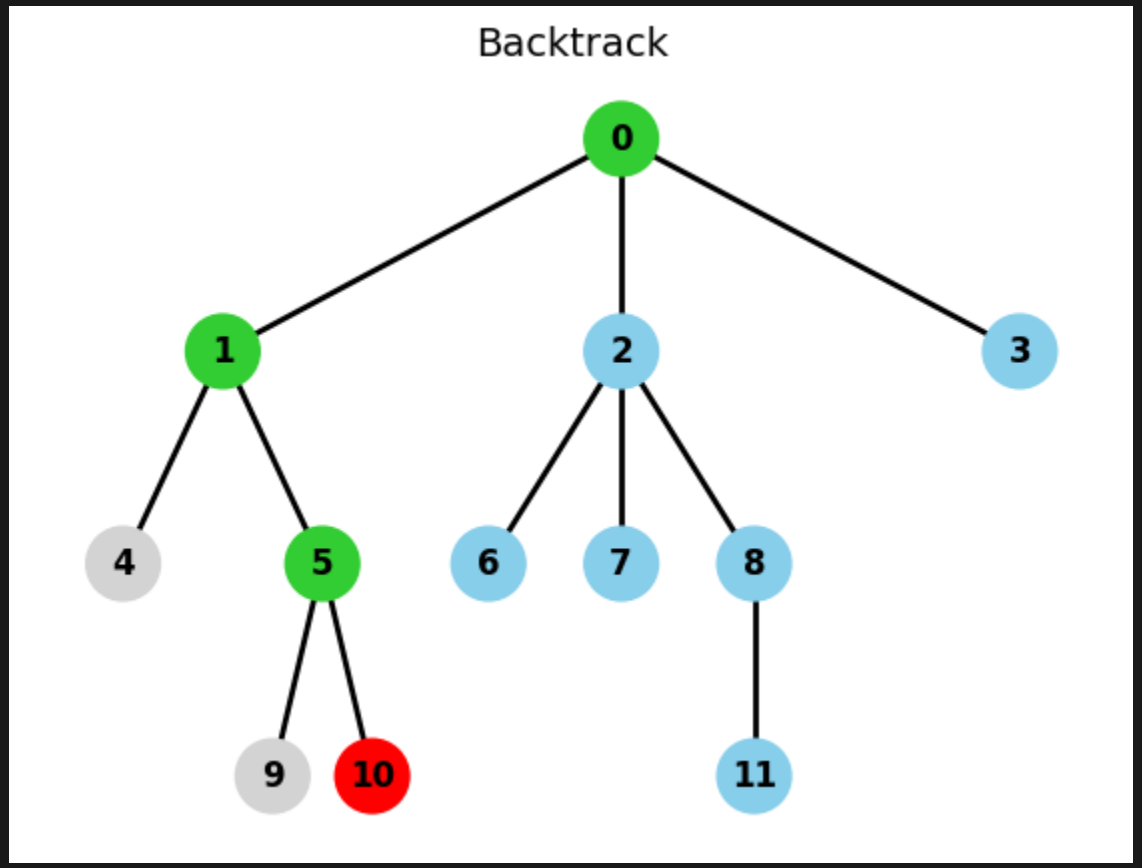
Depth First Search (DFS)

Stack: Last In First Out (LIFO)



Depth First Search (DFS)

Stack: Last In First Out (LIFO)

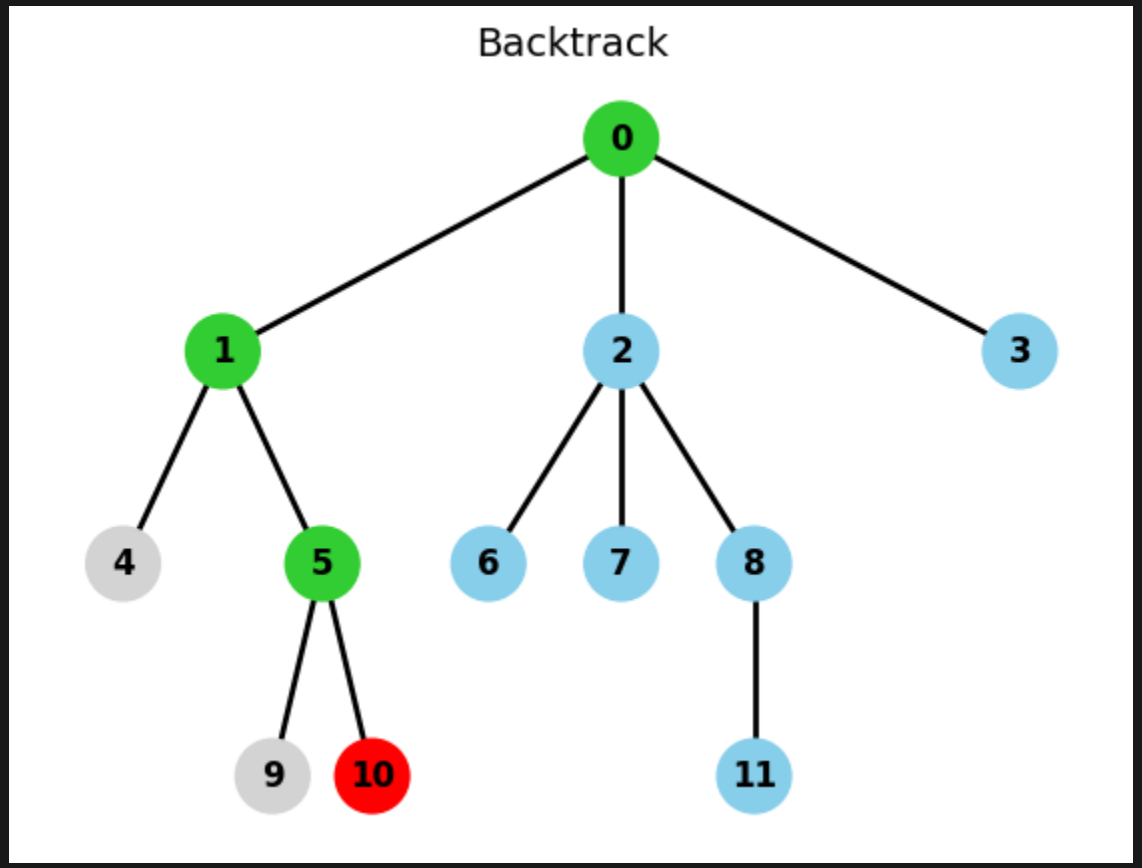


Depth First Search (DFS)

Stack: Last In First Out (LIFO)

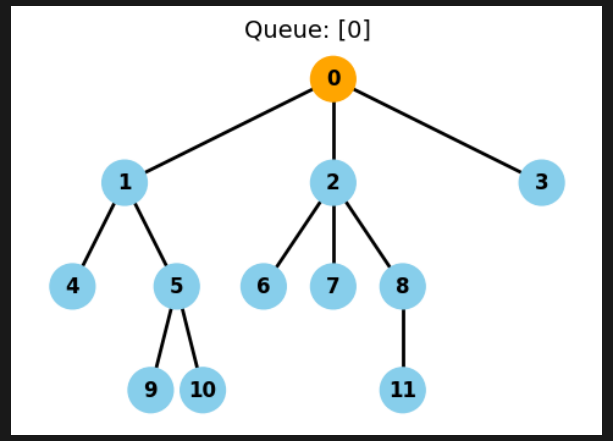
특징

- 시간 복잡도: O(V+E)
- 공간 복잡도: O(V)
  일반적으로 BFS보다
  메모리 사용량이 낮음
- 일반적으로 최단 경로를
  보장하지 않음



Depth First Search (DFS)

수도 코드 (pseudo code)

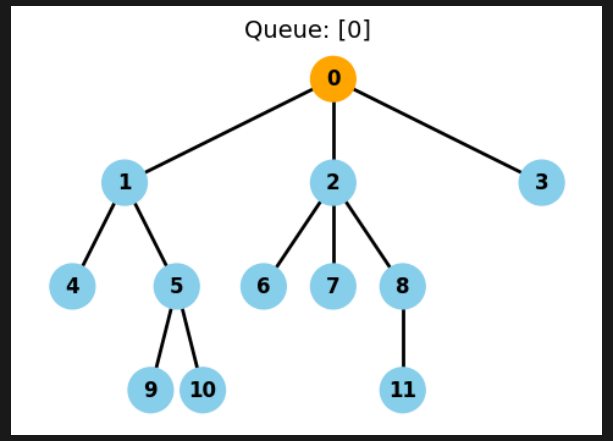



```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

path = dfs(graph, 'A', 'F')
```

```
def dfs(graph, start, goal):
    visited = set()
    stack = [start]

    while stack:
        current = stack.pop()

        if current == goal:
            return True

        if current not in visited:
            visited.add(current)
            for neighbor in graph.get(current, []):
                if neighbor not in visited:
                    stack.append(neighbor)

    return False
```

Depth First Search (DFS)

수도 코드 (pseudo code)




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

path = dfs(graph, 'A', 'F')
```

```
def dfs(graph, start, goal):
    visited = set()
    stack = [start]

    while stack:
        current = stack.pop()

        if current == goal:
            return True

        if current not in visited:
            visited.add(current)
            for neighbor in graph.get(current, []):
                if neighbor not in visited:
                    stack.append(neighbor)

    return False
```

```
def dfs(graph, start, goal):
    stack = [(start, [start])]
    visited = set()

    while stack:
        current, path = stack.pop()

        if current == goal:
            return path

        if current not in visited:
            visited.add(current)
            for neighbor in graph.get(current, []):
                if neighbor not in visited:
                    stack.append((neighbor, path + [neighbor]))

    return None
```

BFS vs. DFS

BFS

- Queue (FIFO) 사용
- 시간 복잡도: O(V+E)
- 공간 복잡도: O(V)
- 목표 노드가 시작점 근처에 있을 때 빠르게 발견 가능
- 가중치가 모두 동일할 경우 최단 경로 탐색 가능

서치 알고리즘

DFS

- Stack (LIFO) 사용
- 시간 복잡도: O(V+E)
- 공간 복잡도: O(V)
  일반적으로 BFS보다 메모리 사용량이 낮음
- 일반적으로 최단 경로를 보장하지 않음

강의 요약

01

Queue (큐)

FIFO

02

BFS

03

Stack (스택)

LIFO

04

DFS