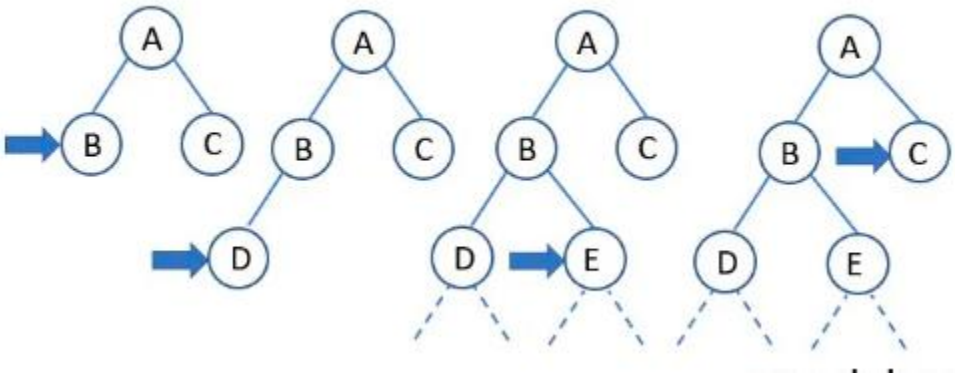
LAB 6

CLO: 01, 03

The depth-limited search:

The depth-limited search (DLS) method is almost equal to depth-first search (DFS), but DLS can work on the infinite state space problem because it bounds the depth of the search tree with a predetermined limit L. Nodes at this depth limit are treated as if they had no successors.

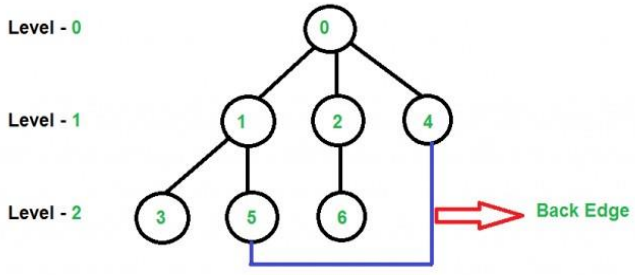
Now use the example in DFS to see what will happen if we use DLS with L=2.



Iterative Deepening Search/ Iterative Deepening Depth First Search

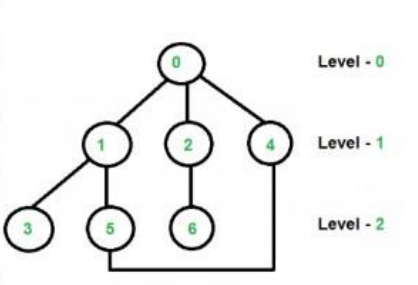
IDDFS combines depth-first search's space-efficiency and breadth-first search's fast search (for nodes closer to root).There can be two cases-

a) **When the graph has no cycle:** This case is simple. We can DFS multiple times with different height limits.
b) **When the graph has cycles:** This is interesting as there is no visited flag in IDDFS.



Although, at first sight, it may seem that since there are only 3 levels, so we might think that Iterative Deepening Depth First Search of level 3, 4, 5,...and so on will remain same. But, this is not the case. You can see that there is a cycle in the above graph, hence IDDFS will change for level-3,4,5..and so on.

Depth	Iterative Deepening Depth First Search
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4 5
3	0 1 3 5 4 2 6 4 5 1



Question 1:

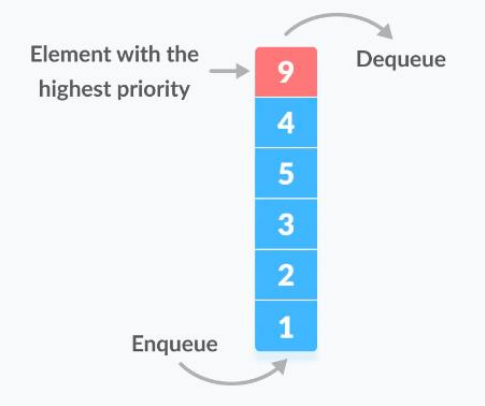
Implement IDDFS using Python:

Algorithm:

```
// Returns true if target is reachable from

// src within max_depth

bool IDDFS(src, target, max_depth)
    for limit from 0 to max_depth
       if DLS(src, target, limit) == true
           return true
    return false

bool DLS(src, target, limit)
    if (src == target)
        return true;

    // If reached the maximum depth,
    // stop recursing.
    if (limit <= 0)
        return false;

    foreach adjacent i of src
        if DLS(i, target, limit?1)
            return true

    return false
```

Priority Queue:

A priority queue is a special type of queue in which each element is associated with a priority value. And, elements are served on the basis of their priority. That is, higher priority elements are served first. However, if elements with the same priority occur, they are served according to their order in the queue.



Question 2:

Implement **Priority Queue** using Python for weighted graph:

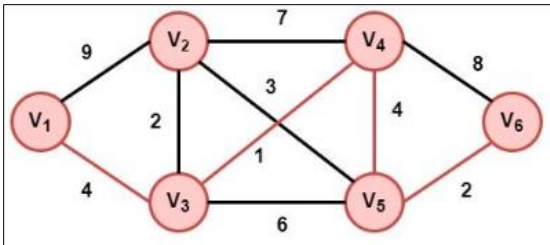
```
class WeightedGraph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, start, end, weight):
        if start not in self.graph:
            self.graph[start] = []
        self.graph[start].append((end, weight))

    def get_neighbors(self, node):
        return self.graph.get(node, [])

def priority_queue_for_graph(graph):
    pq = queue.PriorityQueue()
    pq.put(('A', 0))  # Starting node and its priority

    while not pq.empty():
        current_node, current_priority = pq.get()
        print(f"Processing {current_node} with priority {current_priority}")

        for neighbor, weight in graph.get_neighbors(current_node):
            next_priority = current_priority + weight
            pq.put((neighbor, next_priority))
```

```python
# Create the weighted graph
g = WeightedGraph()
g.add_edge('A', 'B', 5)
g.add_edge('A', 'C', 3)
g.add_edge('B', 'C', 2)
g.add_edge('B', 'D', 4)
g.add_edge('C', 'D', 6)

# Use a priority queue to process the nodes
priority_queue_for_graph(g)
```

```
Processing A with priority 0
Processing B with priority 5
Processing C with priority 3
Processing C with priority 7
Processing D with priority 9
Processing D with priority 9
Processing D with priority 13
```

Question 3:

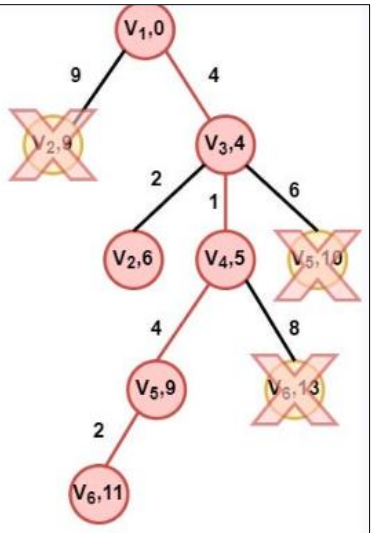
Uniform Cost Search (UCS) Algorithm: Uniform-cost search is an uninformed search algorithm that uses the lowest cumulative cost to find a path from the source to the destination. Nodes



Pace	Opened	Closed
0	{(V_1,0)}	{-}
1	{(V_2,9),(V_3,4)}	{(V_1,0)}
2	{(V_2,6),(V_4,5),(V_5,10)}	{(V_1,0),(V_3,4)}
3	{(V_2,6),(V_6,13),(V_5,9)}	{(V_1,0),(V_3,4),(V_4,5)}
4	{(V_6,13),(V_5,9)}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6)}
5	{(V_6,11)}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)}
6	{-}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)}

Path: V_1 - V_3 - V_4 - V_5 - V_6

Total Cost: 11



are expanded, starting from the root, according to the minimum cumulative cost. The uniform-cost search is then implemented using a Priority Queue.

Question 4:

Implement Uniform Cost Search (UCS) algorithm, using Python

```
# Example usage
g = WeightedGraph()
g.add_edge('A', 'B', 1)
g.add_edge('A', 'C', 5)
g.add_edge('B', 'C', 2)
g.add_edge('B', 'D', 3)
g.add_edge('C', 'D', 1)
g.add_edge('C', 'E', 7)
g.add_edge('D', 'E', 2)

start_node = 'A'
goal_node = 'E'
```