Om Shele
TE C32
2103163

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

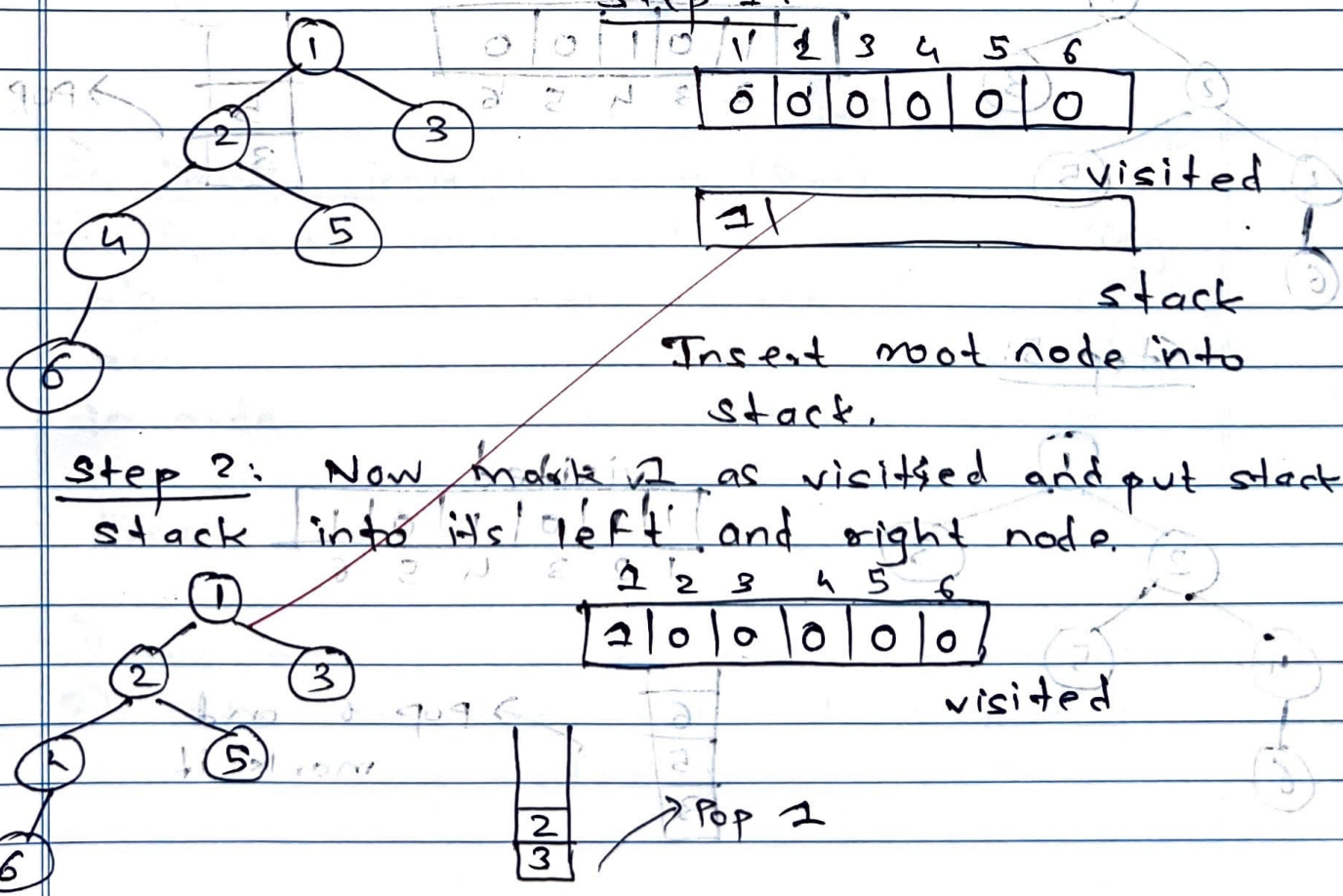
Experiment No : 2

Aim : Implement DFS / DFID / DLS search algorithm in python.

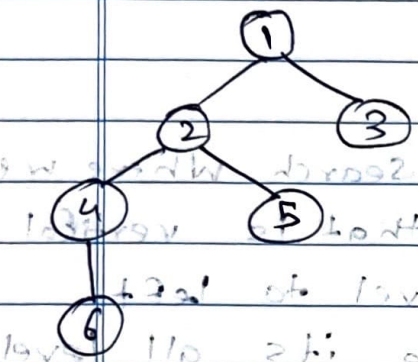
Theory :

1) **Depth First Search :**

- DFS stands for Depth First search where we first traversal in depth wise that is vertical search that is first we travel to left until we reach the end node its all level order traversal.
- And then we would backtrack to the previous ancestor node.
- It uses stack ferit's implementation.
- For e.g.,

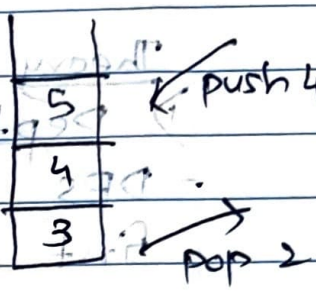


Step 3: Now will pop 2 and explore it and mark visited as 2

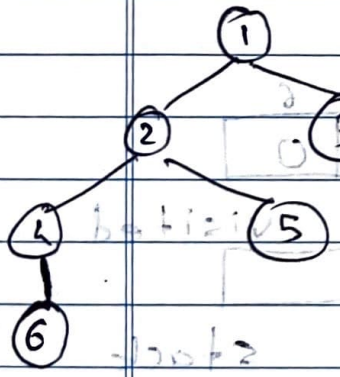


visited

1	2	3	4	5	6
1	1	0	0	0	0

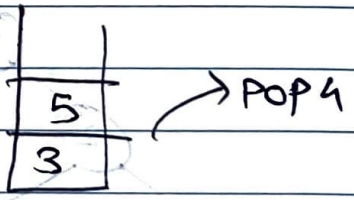


Step 4: Now explore 4 so make it as visited and push it corresponding left and right node

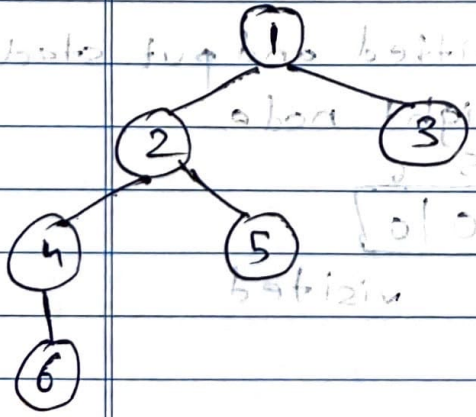


visited

1	2	3	4	5	6
1	1	0	1	0	0

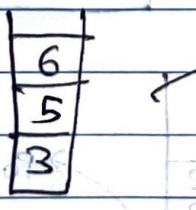


Step 5:



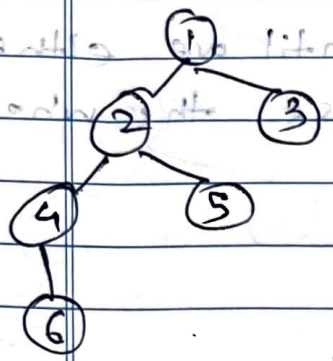
visited

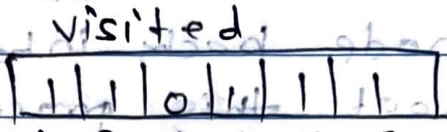
1	2	3	4	5	6
1	1	0	1	0	1

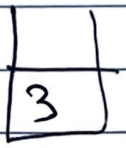


pop 6 and mark it

Step 6:

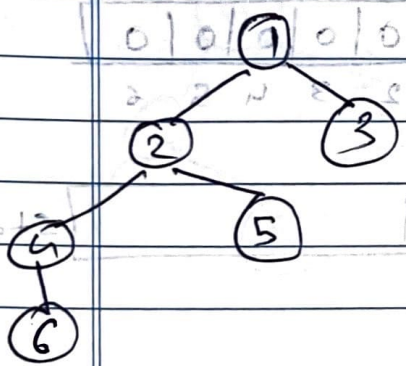


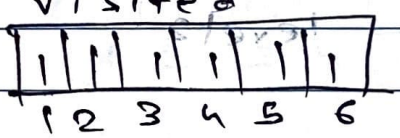




→ pop 5 and mark it.

Step 7:





Empty.

2) Depth Limiting Search (DLS)

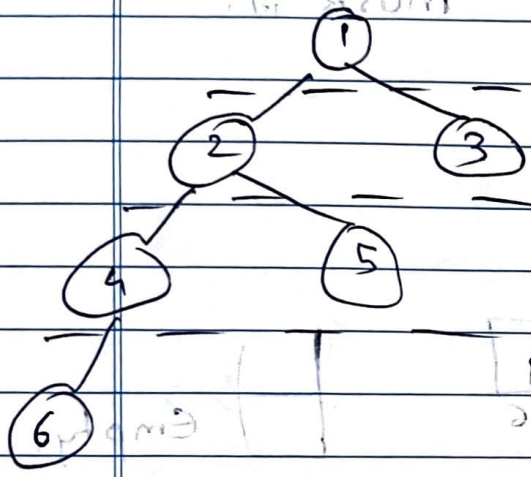
⇒ It's an variation of DFS travelling algorithm. It takes care of an edge case problem with DFS by implementing limiting in order to remove infinity problem.

- Basic DFS algo :-

1) We push the root node into the stack
2). We pop the root node before pushing all its child nodes into the stack.

3) We pop a child node before pushing all of its node back into the stack.
4) We repeat this process until we either find our result or traverse the whole tree.

For eg,



let say limit = 2

Step 1:

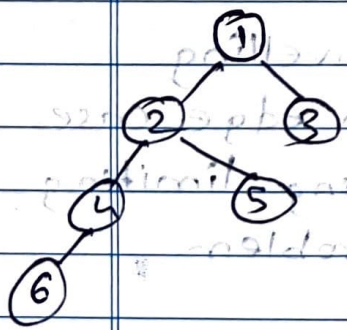
0	0	0	0	0	0
1	2	3	4	5	6

1

stack

Step 2: level counter = 0

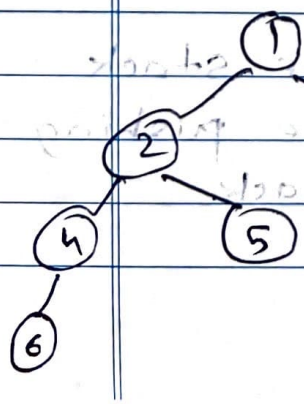
1	0	0	0	0	0
1	2	3	4	5	6



3
2

Step 3: level counter = 1

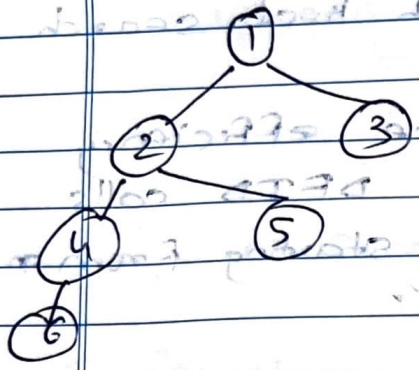
1	1	0	0	0	0
1	2	3	4	5	6



4
5
3

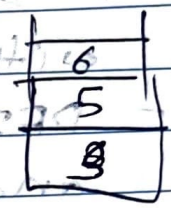
stack

Step 4: level counter = 2

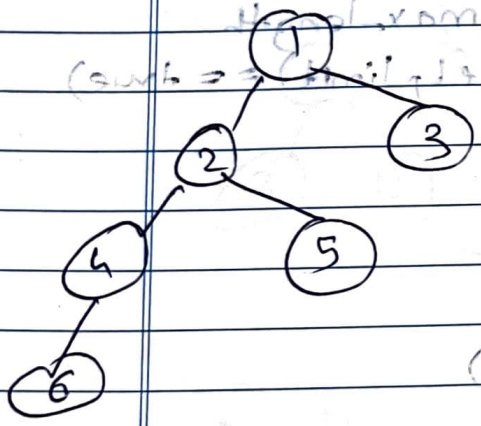


visited :-

1	2	3	4	5	6
1	1	0	1	0	0



Step 5: level counter = 2

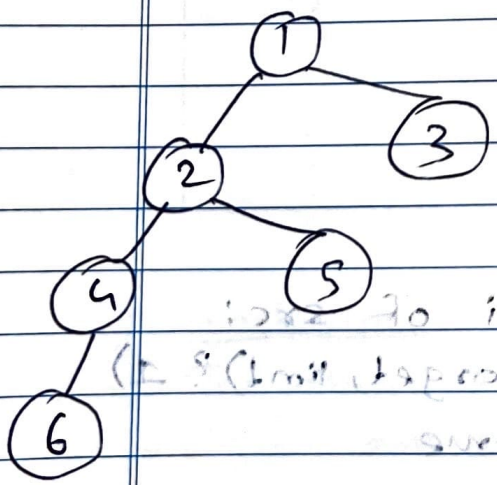


visited :-

1	2	3	4	5	6
1	1	0	1	1	0

can travers to 6 as level 3.

Step 6: level counter = 3;



visited :-

1	2	3	4	5	6
1	1	0	1	1	0

Stack

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

3) DFID

⇒ DFID stands for the Depth first search with Iterative Deepning.

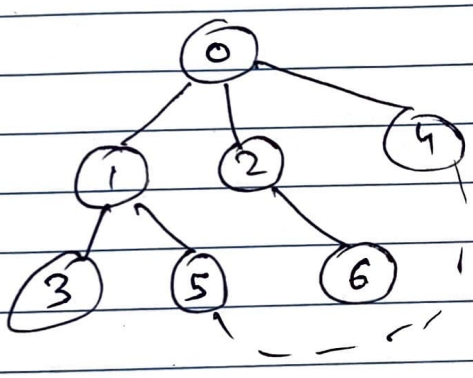
- DFID combines DFS space efficiency and BFS fast search DFID calls DFS for different depth starting from an initial value.
- pseudo code :-

```
bool DFID (src, target, max-depth)
    for limit from 0 to max-length
        if (DLS(src, target, limit) == true)
            return true
    return False.

bool DLS (src, target, limit)

    if (src == target)
        return true

    if (limit <= 0)
        return false

    for each adjacent i of src:
        if (DLS(src, target, limit) ? 1)
            return true
    return false.
```

- It's time complexity is $-O(b^d)$,
  where
  b = branching factor,
  d = depth limit.

- Space complexity - $O(d)$

- Examples :-



back edge
(for higher level > depth of tree)

Depth	DFID
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4
3	0 1 3 5 4 2 6 4 5 7

- Conclusion :
In this experiment, we have learnd DFS, DLS, and DFID algorithms.

Code: (DFS)

```
graph = {
    '5': {
        '3': 2,
        '7': 1
    },
    '3': {
        '2': 3,
        '4': 4
    },
    '7': {
        '8': 5
    },
    '2': {},
    '4': {
        '8': 6
    },
    '8': {}
}

visited = set()
partial_explored = set()


def dfs_with_cost(visited, partial_explored, graph, node, path_cost,
goal_node):
  if node not in visited:
    print(f"{node} (Cost: {path_cost})")
    visited.add(node)

    if node == goal_node:
      print(f"Goal node {goal_node} reached!")
      return

    partial_explored.add(node)
    print(
        f"Nodes with partially explored children in current iteration:
{partial_explored}"
    )

    for neighbour, edge_cost in graph[node].items():
      if neighbour not in visited:
```

```
        dfs_with_cost(visited, partial_explored, graph, neighbour,
                      path_cost + edge_cost, goal_node)

goal_node = '8'
print(
    f"Following is the Depth-First Search with Path Cost to reach goal
node {goal_node}"
)
dfs_with_cost(visited, partial_explored, graph, '5', 0, goal_node)

print("\nNodes with partially explored children in each iteration:")
print(partial_explored)
```

Output:

```
Run
Following is the Depth-First Search with Path Cost to reach goal node 8
5 (Cost: 0)
Nodes with partially explored children in current iteration: {'5'}
3 (Cost: 2)
Nodes with partially explored children in current iteration: {'5', '3'}
2 (Cost: 5)
Nodes with partially explored children in current iteration: {'5', '2', '3'}
4 (Cost: 6)
Nodes with partially explored children in current iteration: {'5', '4', '2', '3'}
8 (Cost: 12)
Goal node 8 reached!
7 (Cost: 1)
Nodes with partially explored children in current iteration: {'5', '4', '2', '3', '7'}

Nodes with partially explored children in each iteration:
{'5', '4', '2', '3', '7'}
```

Code: (DLS)

```
graph = {
    '5': {
        '3': 2,
        '7': 1
    },
    '3': {
        '2': 3,
        '4': 4
    },
    '7': {
        '8': 5
    },
```

```
    '2': {},
    '4': {
        '8': 6
    },
    '8': {}
}

def dls_with_cost(graph, node, goal_node, depth_limit, path_cost,
visited,
                  partial_explored):
  if node not in visited:
    print(f"{node} (Cost: {path_cost})")
    visited.add(node)

  if node == goal_node:
    print(f"Goal node {goal_node} reached!")
    return True

  if depth_limit == 0:
    return False

  partial_explored.add(node)
  print(
      f"Nodes with partially explored children in current iteration:
{partial_explored}"
  )

  for neighbour, edge_cost in graph[node].items():
    if dls_with_cost(graph, neighbour, goal_node, depth_limit - 1,
                     path_cost + edge_cost, visited, partial_explored):
      print(f"{node} (Cost: {path_cost})")
      return True

  return False

goal_node_dls = '8'
depth_limit_dls = 3
visited_dls = set()
partial_explored_dls = set()
print(
    f"Following is the Depth-Limited Search with Path Cost (Depth
Limit: {depth_limit_dls})"
)
```

```
if not dls_with_cost(graph, '5', goal_node_dls, depth_limit_dls, 0,
                     visited_dls, partial_explored_dls):
  print(
      f"Goal node {goal_node_dls} not found within depth limit
{depth_limit_dls}"
  )
```

Output:

```
Following is the Depth-Limited Search with Path Cost (Depth Limit: 3)
5 (Cost: 0)
Nodes with partially explored children in current iteration: {'5'}
3 (Cost: 2)
Nodes with partially explored children in current iteration: {'5', '3'}
2 (Cost: 5)
Nodes with partially explored children in current iteration: {'2', '5', '3'}
4 (Cost: 6)
Nodes with partially explored children in current iteration: {'2', '5', '3', '4'}
8 (Cost: 12)
Goal node 8 reached!
4 (Cost: 6)
3 (Cost: 2)
5 (Cost: 0)
```

Code: (DFID)

```
graph = {
    '5': {
        '3': 2,
        '7': 1
    },
    '3': {
        '2': 3,
        '4': 4
    },
    '7': {
        '8': 5
    },
    '2': {},
    '4': {
        '8': 6
    },
    '8': {}
}
```

```
def dls_with_cost(graph, node, goal_node, depth_limit, path_cost,
visited, partial_explored):
  if node not in visited:
    print(f"{node} (Cost: {path_cost})")
    visited.add(node)

  if node == goal_node:
    print(f"Goal node {goal_node} reached!")
    return True

  if depth_limit == 0:
    return False

  partial_explored.add(node)
  print(
      f"Nodes with partially explored children in current iteration:
{partial_explored}"
  )

  for neighbour, edge_cost in graph[node].items():
    if dls_with_cost(graph, neighbour, goal_node, depth_limit - 1,
path_cost + edge_cost, visited, partial_explored):
      print(f"{node} (Cost: {path_cost})")
      return True

  return False


def dfid_with_cost(graph, start_node, goal_node, max_depth, visited,
partial_explored):
  for depth_limit in range(max_depth + 1):
    print(f"Depth-Limited Search with Depth Limit: {depth_limit}")
    if dls_with_cost(graph, start_node, goal_node, depth_limit, 0,
visited, partial_explored):
      return

goal_node_dfid = '8'
max_depth_dfid = 3
visited_dfid = set()
partial_explored_dfid = set()
print(
    f"Following is the Depth-First Iterative Deepening with Path Cost
(Max Depth: {max_depth_dfid})"
```

```
)
dfid_with_cost(graph, '5', goal_node_dfid, max_depth_dfid,
visited_dfid,
               partial_explored_dfid)
```

Output:

```
Following is the Depth-First Iterative Deepening with Path Cost (Max Depth: 3)
Depth-Limited Search with Depth Limit: 0
5 (Cost: 0)
Depth-Limited Search with Depth Limit: 1
Nodes with partially explored children in current iteration: {'5'}
3 (Cost: 2)
7 (Cost: 1)
Depth-Limited Search with Depth Limit: 2
Nodes with partially explored children in current iteration: {'5'}
Nodes with partially explored children in current iteration: {'3', '5'}
2 (Cost: 5)
4 (Cost: 6)
Nodes with partially explored children in current iteration: {'3', '7', '5'}
8 (Cost: 6)
Goal node 8 reached!
7 (Cost: 1)
5 (Cost: 0)
```