Experiment No. 2

Implementing Depth-First Search (DFS), Depth-Limited Search (DLS), and Depth-First Iterative Deepening (DFID) algorithms in AI involves understanding their basic principles and then translating those into code. Here is a theoretical overview of each algorithm and how you might implement them:

1. **Depth-First Search (DFS):**

- DFS is a fundamental graph traversal algorithm.
- It explores as far as possible along each branch before backtracking.
- It uses a stack (either explicitly or via recursion) to keep track of nodes to visit.
- Pseudocode for DFS:

```plaintext
DFS(G, v):
    mark v as visited
    for each neighbor w of v:
        if w is not visited:
            DFS(G, w)
```

- Implementation steps:
  - Initialize a stack to store nodes to visit.
  - Start with the initial node, mark it as visited, and push it onto the stack.
  - While the stack is not empty:
    - Pop a node from the stack.
    - Mark it as visited.
    - Explore its unvisited neighbors, pushing them onto the stack.

2. **Depth-Limited Search (DLS):**

- DLS is similar to DFS but limits the depth of exploration.
- It's useful for preventing infinite loops in infinite-depth graphs.
- Pseudocode for DLS:

```plaintext
DLS(G, v, limit):
    if limit == 0:
        return
    mark v as visited
    for each neighbor w of v:
        if w is not visited:
            DLS(G, w, limit - 1)
```

- Implementation steps:
  - It's similar to DFS, but with an additional parameter to limit the depth of exploration.
  - Whenever the depth limit is reached, the algorithm backs up to the previous level.

3. **Depth-First Iterative Deepening (DFID):**

- DFID combines the benefits of both DFS and BFS (Breadth-First Search).
- It performs a series of depth-limited searches with increasing depth limits.
- It maintains the advantages of DFS while ensuring complete exploration of the tree.
- Pseudocode for DFID:

```plaintext
DFID(G, start):
    depth = 0
    loop:
        result = DLS(G, start, depth)
```

```
        if result != null:
            return result
        depth = depth + 1
```

- Implementation steps:
  - It repeatedly performs depth-limited searches with increasing depth limits until the goal is found.
  - If the goal is not found within the depth limit, it increases the limit and tries again.

Implementing these algorithms in code involves translating the pseudocode into a programming language of your choice (e.g., Python, Java, etc.). You'll need to represent the graph data structure, handle node visitation, manage the search stack (or recursion), and track the depth level if implementing DLS or DFID. Additionally, you may need to integrate these algorithms into a problem-solving framework depending on the specific application.

Code:

```
def dfs(tree,node,p,goal,closelist,parent):
    parent[node]=p
    print('\nCurrent Node:',node)
    print('ClosedList:',closelist)
    closelist.append(node)

    if node==goal:
        return True;

    for u in tree[node]:
        if u==p:
            continue
        if dfs(tree,u,node,goal,closelist,parent):
```

```
            return True
    return False


n=int(input('Enter the number of node in tree:'))
start=int(input('Enter start node:'))
goal=int(input('Enter goal node:'))

tree=[[] for i in range(n+1)]

print('Enter the edges:')
for i in range(n-1):
    e=input().split()
    u,v=int(e[0]),int(e[1])
    tree[u].append(v)
    tree[v].append(u)

closelist=[]
parent=(n+1)*[-1]
if not dfs(tree,start,-1,goal,closelist,parent):
    print('\nGoal node not found!!!')
else:
    print('\nGoal node found!!!')
    path=[]
    u=goal
    while u!=-1:
        path.append(u)
        u=parent[u]
    print('\nPath: ',end='')
```

```
    path=path[::-1]
    for i in range(len(path)):
        if i==len(path)-1:
            print(path[i])
        else:
            print(path[i],end='->')
```

Output:

Output

```
Enter the number of node in tree:7
Enter start node:1
Enter goal node:6
Enter the edges:
1 2
1 3
2 4
2 5
3 6
3 7

Current Node: 1
ClosedList: []

Current Node: 2
ClosedList: [1]

Current Node: 4
ClosedList: [1, 2]

Current Node: 5
ClosedList: [1, 2, 4]

Current Node: 3
ClosedList: [1, 2, 4, 5]

Current Node: 6
ClosedList: [1, 2, 4, 5, 3]

Goal node found!!!

Path: 1->3->6

=== Code Execution Successful ===
```