SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

K. J. Somaiya College of Engineering, Mumbai-77

(A Constituent College of Somaiya Vidyavihar University)

Department of Electronics Engineering

Course Name: **Modern Artificial Intelligence**

Semester: **IV**

Experiment No: **THREE**

Title: **Implement Depth-First Search(DFS), Depth-limited Search(DLS), and Iterative deepening DFS (ID-DFS) for a search tree.**

Date of Performance: **23/02/2024**

Faculty Name:
Prof. Umang Patel

Name and Roll No.:

Vrishank Warrier, 16014022096

Contents

Experiment: THREE

Implement Depth-First Search(DFS), Depth-limited Search(DLS), and Iterative deepening DFS (ID-DFS) for a search tree.

Vrishank Warrier, 16014022096

23/02/2024

1 Aim and Objective of the Experiment

- Objective 1: For the given goal test the display path. from start state to goal state. Note: For ID-DFS take two variations. a) increasing ”l” linearly from 1,2,3,4 ..... b) increasing ”l” non-linearly from 1,2,4,8 ..... b)
- Objective 2: Calculate time and space complexity for all four variation
- Objective 3: Implement BFS for real-world problems:
  1. Vacuum cleaner robot
  2. 8-puzzle game
  3. Map of Romania

2 Theory

Depth-First Search (DFS) explores a search tree by traversing as deeply as possible along each branch before backtracking. Depth-limited Search (DLS) imposes a maximum depth to control exploration depth, preventing infinite search. Iterative Deepening Depth-First Search (ID-DFS) combines the benefits of DFS and DLS by incrementally increasing the depth limit in a series of DFS iterations, ensuring complete exploration without the memory overhead of traditional DFS or the potential cutoff of DLS. These algorithms are crucial for traversing and searching complex data structures efficiently.

3 Code

3.1 Tool Used/Language

1. VS CODE
2. PYTHON
3. OVERLEAF

3.2 Programs

1.

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def dfs(root, target):
    if root is None:
        return None

    if root.value == target:
        return [root.value]

    left_path = dfs(root.left, target)
    if left_path:
        return [root.value] + left_path

    right_path = dfs(root.right, target)
    if right_path:
        return [root.value] + right_path

    return None


root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

value = int(input("Enter the value of the node to find the path for: "))

path = dfs(root, value)
if path:
    print("DFS traversal:", path)
else:
    print("Node not found in the tree.")
```

Listing 1: Implementing BFS for search tree

2.

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def dls(root, target, depth):
    if root is None or depth < 0:
        return None

    if root.value == target:
        return [root.value]

    left_path = dls(root.left, target, depth - 1)
    if left_path:
        return [root.value] + left_path

    right_path = dls(root.right, target, depth - 1)
    if right_path:
        return [root.value] + right_path

    return None


root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

value = int(input("Enter the value of the node to find the path for: "))
depth_limit = int(input("Enter the depth limit: "))

path = dls(root, value, depth_limit)

if path:
    print("DLS traversal:", path)
else:
    print("Node not found in the tree within the specified depth limit.")
```

Listing 2: Implementing BFS for search tree

3.
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def dfs(root, target):
    if root is None:
        return None

    if root.value == target:
        return [root.value]

    left_path = dfs(root.left, target)
    if left_path:
        return [root.value] + left_path

    right_path = dfs(root.right, target)
    if right_path:
        return [root.value] + right_path

    return None

def id_dfs_l(root, target):
    depth = 0
    while True:
        path = dfs(root, target)
        if path:
            return path
        depth += 1

def id_dfs_nl(root, target):
    depth = 1
    while True:
        path = dfs(root, target)
        if path:
            return path

        depth *= 2


root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

```

```python
value = int(input("Enter the value of the node to find the path for: "))

path_l = id_dfs_l(root, value)
if path_l:
    print("ID-DFS traversal (increasing linearly):", path_l)
else:
    print("Node not found in the tree.")

path_nl = id_dfs_nl(root, value)
if path_nl:
    print("ID-DFS traversal (increasing non-linearly):", path_nl)
else:
    print("Node not found in the tree.")
```

Listing 3: Implementing BFS for search tree

4.
```
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []

def get_neighbors(state):
    neighbors = []
    zero_index = state.index(0)
    if zero_index >= 3:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index - 3] = new_state[zero_index - 3], new_state[zero_index]
        neighbors.append(new_state)
    if zero_index < 6:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index + 3] = new_state[zero_index + 3], new_state[zero_index]
        neighbors.append(new_state)
    if zero_index % 3 != 0:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index - 1] = new_state[zero_index - 1], new_state[zero_index]
        neighbors.append(new_state)
    if (zero_index + 1) % 3 != 0:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index + 1] = new_state[zero_index + 1], new_state[zero_index]
        neighbors.append(new_state)
    return neighbors

def dfs(initial_state, final_state):
    visited = set()
    stack = [(initial_state, [])]

    while stack:
        state, path = stack.pop()
        visited.add(tuple(state))

        if state == final_state:
            return path

        for neighbor in get_neighbors(state):
            if tuple(neighbor) not in visited:
                stack.append((neighbor, path + [neighbor]))

    return None
```

```
print("Enter initial state of the puzzle (0 represents the empty tile):")
initial_state = [int(input()) for _ in range(9)]
print("Enter final state of the puzzle:")
final_state = [int(input()) for _ in range(9)]

path = dfs(initial_state, final_state)

if path:
    print("Shortest path to reach the final state:")
    step = 1
    for state in path:
        print("Step", step)
        print(state[:3])
        print(state[3:6])
        print(state[6:])
        print()
        step += 1
else:
    print("No solution exists for the given states.")
```

Listing 4: Implementing DFS for 8 puzzle game

5.
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []

def get_neighbors(state):
    neighbors = []
    zero_index = state.index(0)
    if zero_index >= 3:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index - 3] = new_state[zero_index - 3], new_state[zero_index]
        neighbors.append(new_state)
    if zero_index < 6:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index + 3] = new_state[zero_index + 3], new_state[zero_index]
        neighbors.append(new_state)
    if zero_index % 3 != 0:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index - 1] = new_state[zero_index - 1], new_state[zero_index]
        neighbors.append(new_state)
    if (zero_index + 1) % 3 != 0:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index + 1] = new_state[zero_index + 1], new_state[zero_index]
        neighbors.append(new_state)
    return neighbors

def dls(state, final_state, depth_limit):
    stack = [(state, [])]

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

        if len(path) > depth_limit:
            continue

        if current_state == final_state:
            return path

        for neighbor in get_neighbors(current_state):
            stack.append((neighbor, path + [neighbor]))

    return None
```

```python
print("Enter initial state of the puzzle (0 represents the empty tile):")
initial_state = [int(input()) for _ in range(9)]
print("Enter final state of the puzzle:")
final_state = [int(input()) for _ in range(9)]
depth_limit = int(input("Enter depth limit: "))

path = dls(initial_state, final_state, depth_limit)

if path:
    print("Shortest path to reach the final state:")
    step = 1
    for state in path:
        print("Step", step)
        print(state[:3])
        print(state[3:6])
        print(state[6:])
        print()
        step += 1
else:
    print("No solution exists for the given states within the depth limit."
)
```

Listing 5: Implementing DLS for 8 puzzle game

6.

```
from collections import deque

class Node:
    def __init__(self, value):
        self.value = value
        self.children = []

def get_neighbors(state):
    neighbors = []
    zero_index = state.index(0)
    if zero_index >= 3:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index - 3] = new_state[zero_index - 3], new_state[zero_index]
        neighbors.append(new_state)
    if zero_index < 6:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index + 3] = new_state[zero_index + 3], new_state[zero_index]
        neighbors.append(new_state)
    if zero_index % 3 != 0:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index - 1] = new_state[zero_index - 1], new_state[zero_index]
        neighbors.append(new_state)
    if (zero_index + 1) % 3 != 0:
        new_state = state.copy()
        new_state[zero_index], new_state[zero_index + 1] = new_state[zero_index + 1], new_state[zero_index]
        neighbors.append(new_state)
    return neighbors


def id_bfs(initial_state, final_state):
    depth_limit = 0

    while True:
        visited = set()
        queue = deque([(initial_state, [])])

        while queue:
            state, path = queue.popleft()
            visited.add(tuple(state))

            if state == final_state:
                return path

            if len(path) < depth_limit:
                for neighbor in get_neighbors(state):
```

```
                    if tuple(neighbor) not in visited:
                        queue.append((neighbor, path + [neighbor]))

        depth_limit += 1

        if depth_limit > 30:  # You can adjust the maximum depth limit
            return None

def main():
    print("Enter initial state of the puzzle (0 represents the empty tile):
")
    initial_state = [int(input()) for _ in range(9)]
    print("Enter final state of the puzzle:")
    final_state = [int(input()) for _ in range(9)]

    path = id_bfs(initial_state, final_state)

    if path:
        print("Shortest path to reach the final state:")
        step = 1
        for state in path:
            print("Step", step)
            print(state[:3])
            print(state[3:6])
            print(state[6:])
            print()
            step += 1
    else:
        print("No solution exists for the given states.")


main()
```

Listing 6: Implementing ID DFS for 8 puzzle game

3.3 Output

1. Output of dfs

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\DFS.py"
Enter the value of the node to find the path for: 6
DFS traversal: [1, 3, 6]
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03>
```

2. Output of dls

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\DLS.py"
Enter the value of the node to find the path for: 6
Enter the depth limit: 1
Node not found in the tree within the specified depth limit.
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03>
```

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\DLS.py"
Enter the value of the node to find the path for: 6
Enter the depth limit: 5
DLS traversal: [1, 3, 6]
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03>
```

3. Output of id dfs

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\ID_DFS.py
Enter the value of the node to find the path for: 6
ID-DFS traversal (increasing linearly): [1, 3, 6]
ID-DFS traversal (increasing non-linearly): [1, 3, 6]
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03>
```

4. Output of dfs 8 puzzle game

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\8 puzzle game using dfs.py"
Enter initial state of the puzzle (0 represents the empty tile):
2
8
3
1
0
4
7
6
5
Enter final state of the puzzle:
1
2
3
8
0
4
7
6
5
```

```
[1, 2, 3]
[6, 4, 5]
[8, 0, 7]

Step 2566
[1, 2, 3]
[6, 4, 5]
[8, 7, 0]

Step 2567
[1, 2, 3]
[6, 4, 0]
[8, 7, 5]

Step 2568
[1, 2, 3]
[6, 0, 4]
[8, 7, 5]

Step 2569
[1, 2, 3]
[0, 6, 4]
[8, 7, 5]

Step 2570
[1, 2, 3]
[8, 6, 4]
[0, 7, 5]

Step 2571
[1, 2, 3]
[8, 6, 4]
[7, 0, 5]

Step 2572
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
```

5. Output of dls 8 puzzle game

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\8 puzzle game using dls.py"
Enter initial state of the puzzle (0 represents the empty tile):
2
8
3
1
0
4
7
6
5
Enter final state of the puzzle:
1
2
3
8
0
4
7
6
5
Enter depth limit: 10
Shortest path to reach the final state:
Step 1
[2, 8, 3]
[1, 4, 0]
[7, 6, 5]

Step 2
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]

Step 3
[2, 8, 3]
[1, 4, 0]
[7, 6, 5]

Step 4
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
```

```
Step 5
[2, 8, 3]
[1, 4, 0]
[7, 6, 5]

Step 6
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]

Step 7
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

Step 8
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

Step 9
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

Step 10
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
```

6. Output of id dfs 8 puzzle game

```
PS C:\Users\91993\Desktop\MAI\EXP\EXP-03> python -u "c:\Users\91993\Desktop\MAI\EXP\EXP-03\8 puzzle game using id dfs.py"
Enter initial state of the puzzle (0 represents the empty tile):
2
8
3
1
0
4
7
6
5
Enter final state of the puzzle:
1
2
3
8
0
4
7
6
5
Shortest path to reach the final state:
Step 1
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

Step 2
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

Step 3
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

Step 4
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
```

3.4 Time and Space Complexity

- Depth-First Search
  - Time Complexity: $O(b^m)$
  - Space Complexity: $O(bm)$
- Depth-Limited Search
  - Time Complexity: $O(b^l)$
  - Space Complexity: $O(bl)$
- Iterative Deepening Depth-First Search - linear
  - Time Complexity: $O(b^d)$
  - Space Complexity: $O(bd)$
- Iterative Deepening Depth-First Search - non linear
  - Time Complexity: $O(b^d)$
  - Space Complexity: $O(b^d)$

4 Post Lab Question

1. Do the comparative analysis of DFS, DLS, and ID-DFS.
ANS:
1. Depth-First Search (DFS):
-Completeness: It might get stuck in infinite loops or miss solutions in certain cases.
-Optimality: It doesn't always find the best solution.
-Efficiency: It's memory-efficient but not always time-efficient.

2. Depth-Limited Search (DLS):
-Completeness: If the limit is too low, it might miss the solution.
-Optimality: Similar to DFS, it doesn't guarantee the best solution.
-Efficiency: It's memory-efficient like DFS.

3. Iterative Deepening Depth-First Search (ID-DFS):
-Completeness: It's guaranteed to find a solution if one exists.
-Optimality: It's optimal if the solution's depth is known.
-Efficiency: It's more memory-efficient than breadth-first search but can take more time compared to DFS.

5 Conclusion

We have successfully learned about DFS Search algorithms and implemented the algorithm and were able to understand its working. Using DFS, DLS and ID-DFS, we were also able to implement a real-life application, i.e. Solving 8 puzzle game.

References

[1] https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/.

[2] https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/.

[3] chat.openai.com.