Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Name: Divyesh Khunt SapID:60009210116 Batch: D12

Experiment 2

(Uninformed Search)

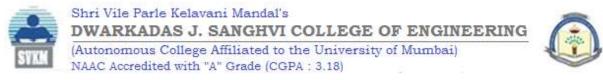
Aim: Implement Depth First Iterative Deepening to find the path for a given planning problem.

Theory:

Solving a problem by search is solving a problem by trial and error. Several real-life problems can be modelled as a state-space search problem.

- 1. Choose your problem and determine what constitutes a STATE (a symbolic representation of the state-of-existence).
- 2. Identify the START STATE and the GOAL STATE(S).
- 3. Identify the MOVES (single-step operations/actions/rules) that cause a STATE to change.
- 4. Write a function that takes a STATE and applies all possible MOVES to that STATE to produce a set of NEIGHBOURING STATES (exactly one move away from the input state). Such a function (state-transition function) is called MoveGen. MoveGen embodies all the single-step operations/actions/rules/moves possible in a given STATE. The output of MoveGen is a set of NEIGHBOURING STATES. MoveGen: STATE --> SET OF NEIGHBOURING STATES. From a graph theoretic perspective the state space is a graph, implicitly defined by a MoveGen function. Each state is a node in the graph, and each edge represents one move that leads to a neighbouring state. Generating the neighbours of a state and adding them as candidates for inspection is called "expanding a state". In state space search, a solution is found by exploring the state space with the help of a MoveGen function, i.e., expand the start state and expand every candidate until the goal state is found.

State spaces are used to represent two kinds of problems: configuration and planning problems.



- 1. In configuration problems the task is to find a goal state that satisfies some properties.
- 2. In planning problems the task is to find a path to a goal state. The sequence of moves in the path constitutes a plan.

Algorithm DFID

```
DFID-2(S)
1 count \leftarrow -1
   path ← empty list
3
  depthBound \leftarrow 0
4
   repeat
5
        previousCount ← count
6
        (count, path) \leftarrow DB-DFS-2(S, depthBound)
        depthBound \leftarrow depthBound + 1
7
   until (path is not empty) or (previousCount = count)
   return path
DB-DFS-2(S, depthBound)
    > Opens new nodes, i.e., nodes neither in OPEN nor in CLOSED,
    > and reopens nodes present in CLOSED and not present in OPEN.
10 count \leftarrow 0
11 OPEN \leftarrow (S, null, 0): []
12 CLOSED ← empty list
    while OPEN is not empty
13
         nodePair ← head OPEN
14
         (N, \underline{\hspace{1em}}, depth) \leftarrow nodePair
15
         if GOALTEST(N) = TRUE
16
              return (count, RECONSTRUCTPATH(nodePair, CLOSED))
17
         else CLOSED ← nodePair : CLOSED
18
19
              if depth < depthBound
                   neighbours \leftarrow MoveGen(N)
20
21
                   newNodes ← REMOVESEEN(neighbours, OPEN, [])
22
                   newPairs \leftarrow MAKEPAIRS(newNodes, N, depth + 1)
23
                   OPEN ← newPairs ++ tail OPEN
24
                   count \leftarrow count + length newPairs
25
              else OPEN ← tail OPEN
26
    return (count, empty list)
```

Auxiliary Functions for DFID

MAKEPAIRS(nodeList, parent, depth) if nodeList is empty 2 return empty list 3 **else** nodePair ← (**head** nodeList, parent, depth) return nodePair : MAKEPAIRS(tail nodeList, parent, depth) RECONSTRUCTPATH(nodePair, CLOSED) 1 SKIPTo(parent, nodePairs, depth) 2 $\textbf{if } (\mathsf{parent}, \, \underline{\hspace{1em}} \, \mathsf{, depth}) = \textbf{head } \mathsf{nodePairs}$ 3 return nodePairs 4 else return SkipTo(parent, tail nodePairs, depth) 5 (node, parent, depth) ← nodePair 6 path ← node: [] 7 while parent is not null path ← parent : path 9 $CLOSED \leftarrow SKIPTo(parent, CLOSED, depth - 1)$ 10 $(_, parent, depth) \leftarrow head CLOSED$ 11 return path

Lab Assignment to do:

Select any one problem from the following and implement DFID to find the path from start state to goal state. Analyse the Time and Space complexity. Comment on Optimality and completeness of the solution.

Problem 1: 8-Puzzle Problem 2: Water Jug Problem 3: Graph

DIFD ON TSP:

```
DFID ON TSP

[1] from collections import defaultdict

[6] def dfs(graph, node, goal, depth, visited):
    if depth == 0:
        return False
    if node == goal:
        return True

mark_visited(visited, node)

for neighbor in movegen(graph, node):
    if not visited[neighbor]:
    if dfs(graph, neighbor, goal, depth - 1, visited):
        return True

return True
```

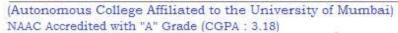
```
def dfid(graph, start, goal):
    max_depth = 0

while True:
    visited = defaultdict(bool)
    if dfs(graph, start, goal, max_depth, visited):
        return reconstruct_path(graph, start, goal, visited)
    max_depth += 1
```



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING





Department of Computer Science and Engineering (Data Science)

```
[8] def movegen(graph, node):
    return graph[node]

def set_goal(goal_node):
    return goal_node
```

```
def reconstruct_path(graph, start, goal, visited):
    path = []
    current = goal

while current != start:
    path.append(current)
    for neighbor in graph[current]:
        if visited[neighbor]:
            current = neighbor
            break

path.append(start)
    path.reverse()
    return path
```

```
[10] def mark_visited(visited, node):
    visited[node] = True

def remove_seen(visited, node):
    visited[node] = False
```



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING



(Autonomous College Affiliated to the University of Mumbai) NAAC Accredited with "A" Grade (CGPA: 3.18)

Department of Computer Science and Engineering (Data Science)

```
graph = {}
while True:
    key = input("Enter a key (or '-1' to quit): ")
    if key.lower() == '-1':
        break
    value = input(f"Enter a value for '{key}': ")
    if key in graph:
        graph[key].append(value)
    else:
        graph[key] = [value]
print("User Dictionary:")
for key, values in graph.items():
    print(f"{key}: {', '.join(values)}")
Enter a key (or '-1' to quit): S
Enter a value for 'S': 'A', 'B', 'D'
Enter a key (or '-1' to quit): A
Enter a value for 'A': 'C', 'B', 'S'
Enter a key (or '-1' to quit): B
Enter a value for 'B': 'A', 'S', 'C'
Enter a key (or '-1' to quit): C
Enter a value for 'C': 'B', 'A'
Enter a key (or '-1' to quit): D
Enter a value for 'D': 'S', 'G'
Enter a key (or '-1' to quit): G
Enter a value for 'G': 'D'
Enter a key (or '-1' to quit): -1
User Dictionary:
S: 'A', 'B', 'D'
A: 'C', 'B', 'S'
B: 'A','S', 'C'
C: 'B','A'
D: 'S', 'G'
G: 'D'
```



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING



(Autonomous College Affiliated to the University of Mumbai) NAAC Accredited with "A" Grade (CGPA: 3.18)

Department of Computer Science and Engineering (Data Science)

```
[23] graph = {
         'S': ['A', 'B', 'D'],
         'A': ['C', 'B', 'S'],
         'B': ['A','S', 'C'],
         'C': ['B','A'],
         'D': ['S', 'G'],
         'G': ['D']
     }
     start node = input("Enter the start node: ")
     goal_node = input("Enter the goal node: ")
     goal = set_goal(goal_node)
     path = dfid(graph, start_node, goal)
     if path:
         print(f"Path from {start_node} to {goal_node}: {path}")
         for node in path:
             remove_seen(graph, node)
     else:
         print(f"No path found from {start_node} to {goal_node}.")
```

OUTPUT:

```
Enter the start node: A
Enter the goal node: G
Path from A to G: ['A', 'S', 'D', 'G']
```

```
Enter the start node: B
Enter the goal node: G
Path from B to G: ['B', 'C', 'A', 'S', 'D', 'G']
```