AI Lab 1 Report

Om Patil (200010036)
Hrishikesh Pable (200010037)

December 2021

1 Psuedocode

1.1 $MoveGen(curPos)$

```
for each direction do
    newPos = shift curPos in direction
    if curPos inside maze and is not wall then
        if curPos not in Explored ∪ Frontier then
            Frontier.append(curPos)
        end if
    end if
end for
```

1.2 $GoalTest(curPos)$

```
if curPos is Goal then
    return True
else
    return False
end if
```

In the actual implementation both of these functions have been clubbed together to form the *explore*() function.

2 Implementations

Each of the following methods have been implemented as classes in python.

2.1 BFS (Breadth First Search)

In this method we implement a queue from which we dequeue nodes and explore them, until either queue is empty or the goal is found.

2.2 DFS (Depth First Search)

In this method we implement a stack from which we pop nodes and explore them, until either stack is empty or the goal is found.

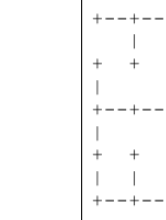
2.3 DLDF (Depth Limited Depth First Search)

This method is similar to DFS; however nodes which exceed the depth limit are not pushed into the stack.

2.4 DFID (Depth First Iterative Deepening Search)

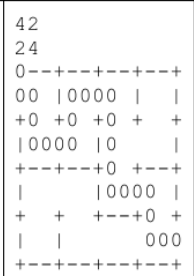
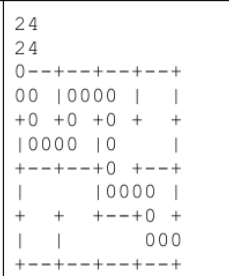
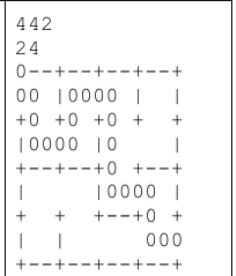
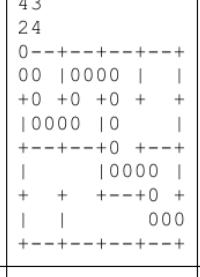
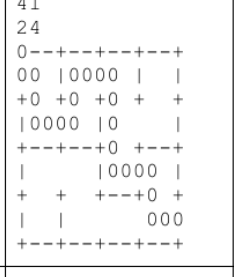
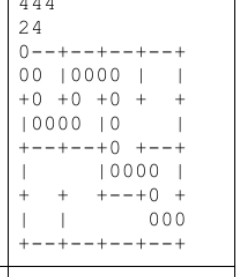
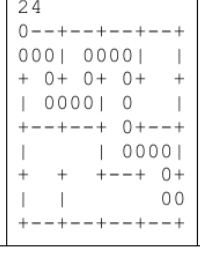
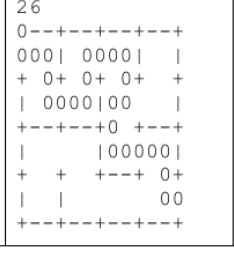
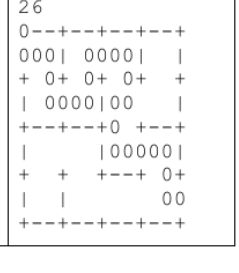
In this method we run DLDF while incrementing the depth limit until we find a valid path.

3 Comparison of Methods

Input	BFS	DFS	DFID
	42 24	24 24	442 24
	59 33	41 33	911 33
	97 55	71 55	2222 55

BFS is complete and gives the optimal solution in each case (as long as the branching factor is finite), but its space complexity is very high (exponential). DFS has a lower space complexity (linear polynomial) and is complete, but it may not be optimal. So, DFID resolves this issue, by setting a limit to the depth until which the nodes are explored. We sequentially go on increasing this limit until the goal is found, ensuring the shortest path. The same can be observed in the above given table.

4 Dependence of Result on Order of Neighbours Added

Order	BFS	DFS	DFID
Down > Up > Right > Left			
Up > Down > Left > Right			
Left > Right > Up > Down			

As seen in the above table we can conclude that the path as well as number of nodes explored are dependent on the order that neighbours are added.