THE MAZE

(BFS)

Dipali Gajera (19645)
San Francisco Bay University
Summer, 2022

Contents

INTRODUCTION

- The Breadth First Search algorithm is a common way to solve node-based path executions. When given a graph of nodes, BFS will essentially gather all the paths that could be taken and visit each one until the target node is reached.
- A graph is a type of data structure used to describe related data. Nodes and edges make up graphs. Edges as pathways connecting various nodes.
- Another widely used data structure is a queue. It is as a line that we always add to the back of by taking from the front.

DESIGN

(*) BFS :-

1) visited : 0
Queue : 0

1) Add 0 to the queue
2) Mark 0 as visited

2) V : 0
1

Q :

1) Remove 0 from the queue
2) Print : 0

3) V : 0 C K
1 1 1

Q : C K

1) Add C & K to the queue
2) Mark C & K visited.

4) V : 0 C K
1 1 1

Q : K

1) Remove C from the queue
2) print 0 C

5) V : 0 C K G1
1 1 1 1

Q : K G1

1) Add G1
2) Mark G1 visited.

6) V : 0 C K G1
1 1 1 1

Q : G1

2) Remove K
3) print 0 C K

8) V : O C K G
Q :
1) Remove G
2) print O C K G.

9) V : O C K G D
Q : D
1) Add D
2) Mark D visited.

9) V : O C K G D
Q :
1) Remove D
2) Print : O C K G D

10) V : O C K G D A I
Q : A I
1) Add A
2) Mark A visited.

11) V : O C K G D A I
Q : I
1) Remove A
2) Print :
O C K G D A

12) V : O C K G D A I B
Q : I B
1) Add B
2) Mark B visited

13) V = O C K G D A I B

Q : B

1) Remove I
2) print :
O C K G D A I

14) V = O C K G D A I B R

Q = B R

1) Add R
2) Mark R visited

15) V = O C K G D A I B R

Q : R

1) Remove R
2) print :-
O C K G D A I B

16) V = O C K G D A I B R

Q :

1) Remove R
2) print :- O C K G D A I B R.

IMPLEMENTATION

Week-14 > MazeBFS.py > findPathBfs

```python
import collections

def findPathBfs(maze, start, destination):
    dirs = [(0,1), (0,-1), (1,0), (-1,0)]
    queue = collections.deque([(start [0], start [1])])
    while queue:
        x, y = queue.popleft()

        if x == destination[0] and y == destination[1]:
            return True

        for dx, dy in dirs:
            nx = x
            ny = y

        while 0 <= nx + dx < len (maze) and 0 <= ny + dy < len(maze [0]) and maze[nx + dx] [ny + dy] != 1:
            nx += dx
            ny += dy

        if maze [nx] [ny] != 0:
            continue

        maze [nx] [ny] = 2
        queue.append( (nx, ny))
    return False
```

Week-14 > MazeBFS.py > findPathBfs

```python
def main():
    maze = []
    val = int(input("How many array do you want to enter : "))

    for i in range(0,val):
        arr = []
        n = int(input("How many elements do you want to enter : "))
        for j in range(0, n):
            ele = int(input("Enter array values : "))
            arr.append(ele)
        maze.append(arr)

    print()
    val = 2
    destination = []
    starting_position = []
    for k in range(0,val):
        pos = int(input("Enter the start position : "))
        destination.append(pos)

    print()
    for d in range(0,val):
        pos = int(input("Enter the position for Destination : "))
        starting_position.append(pos)

    print()
    result = findPathBfs(maze, destination, starting_position)
    print("-----------------------------------")
    print("The Maze output will be :-", '"', result, '"')
    print("-----------------------------------")

if __name__ == '__main__':
    main()
```

TEST

```
dipaligajera@Dipalis-MacBook-Pro Algoritham % /usr/bin/python3 "/Users/dipaligajera/Deskt
op/untitled folder 2/Algoritham/Week-14/MazeBFS.py"
How many array do you want to enter : 5
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 1
Enter array values : 0
Enter array values : 0
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 1
Enter array values : 0
How many elements do you want to enter : 5
Enter array values : 1
Enter array values : 1
Enter array values : 0
Enter array values : 1
Enter array values : 1
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0

Enter the start position : 0
Enter the start position : 4

Enter the position for Destination : 3
Enter the position for Destination : 2

-----------------------------------
The Maze output will be :- " False "
-----------------------------------
dipaligajera@Dipalis-MacBook-Pro Algoritham %
```

```
dipaligajera@Dipalis-MacBook-Pro Algoritham % /usr/bin/python3 "/Users/dipaligajera/Desktop/untitled folder 2/Algoritham/Week-14/MazeBFS.py"
How many array do you want to enter : 5
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 1
Enter array values : 0
Enter array values : 0
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 1
Enter array values : 0
How many elements do you want to enter : 5
Enter array values : 1
Enter array values : 1
Enter array values : 0
Enter array values : 1
Enter array values : 1
How many elements do you want to enter : 5
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0
Enter array values : 0

Enter the start position : 4
Enter the start position : 3

Enter the position for Destination : 0
Enter the position for Destination : 1

-----------------------------------
The Maze output will be :- " False "
-----------------------------------
```

ENHANCEMENT IDEAS

- This maze problem, we can also solve this using shortest path algorithm. Like, Dijkstra's Algorithm and Bellman Ford's Algorithm.
- Another way for this problem using shortest path, Minimum Spanning Tree(MST).
- Converting the maze into X and Y squares is the either way and convert into tree using nodes.

CONCLUSION

- From this project, i learned a lot about Breadth First Search(BFS) Algorithm and also compare with other algorithm.
- The queue method makes it easier to move ahead using First In Last Out concept in order to arrive to the target.

REFERENCE

- https://www.youtube.com/watch?v=D14YK-0MtcQ
- https://github.com/Areesha-Tahir/BFS-DFS-Maze-Solver-In-Python
- https://makeschool.org/mediabook/oa/tutorials/trees-and-mazes/solving-the-maze/