

## Title: Analysis of DFS and A\* Algorithm

Ex. No.:05

Reg. No.: RA2011003011334

Date: 13/02/23

Name: RISHABH SINGH SAHIL

### Aim:

Implementation and Analysis of DFS and A\* Algorithm.

### DFS Program:

class Graph:

    # Constructor

    def \_\_init\_\_(self):

        # default dictionary to store graph

        self.graph = defaultdict(list)

    # function to add an edge to graph

    def addEdge(self, u, v):

        self.graph[u].append(v)

    # A function used by DFS

    def DFSUtil(self, v, visited):

        # Mark the current node as visited

        # and print it

        visited.add(v)

        print(v, end=' ')

        # Recur for all the vertices

        # adjacent to this vertex

        for neighbour in self.graph[v]:

            if neighbour not in visited:

                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses

    # recursive DFSUtil()

    def DFS(self, v):

        # Create a set to store visited vertices

        visited = set()

        # Call the recursive helper function

        # to print DFS traversal

        self.DFSUtil(v, visited)

if \_\_name\_\_ == "\_\_main\_\_":

    g = Graph()

    g.addEdge(0, 1)

    g.addEdge(0, 2)

    g.addEdge(1, 2)

```

g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2)")
# Function call
g.DFS(2)

```

### Output:

```

main.py Run Shell Clear
1
2 def dfs(graph, start, visited=None):
3     if visited is None:
4         visited = set()
5         visited.add(start)
6
7     print(start)
8
9     for next in graph[start] - visited:
10        dfs(graph, next, visited)
11    return visited
12
13
14 graph = {'0': set(['1', '2']),
15         '1': set(['0', '3', '4']),
16         '2': set(['0']),
17         '3': set(['1']),
18         '4': set(['2', '3'])}
19
20 dfs(graph, '0')

```

0  
2  
1  
4  
3  
3  
>

### A\* Program:

```
from collections import defaultdict
```

```
class Graph:
```

```

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)

```

```
visited[s] = True
```

```
while queue:
```

```
    # Dequeue a vertex from  
    # queue and print it  
    s = queue.pop(0)  
    print(s, end=" ")
```

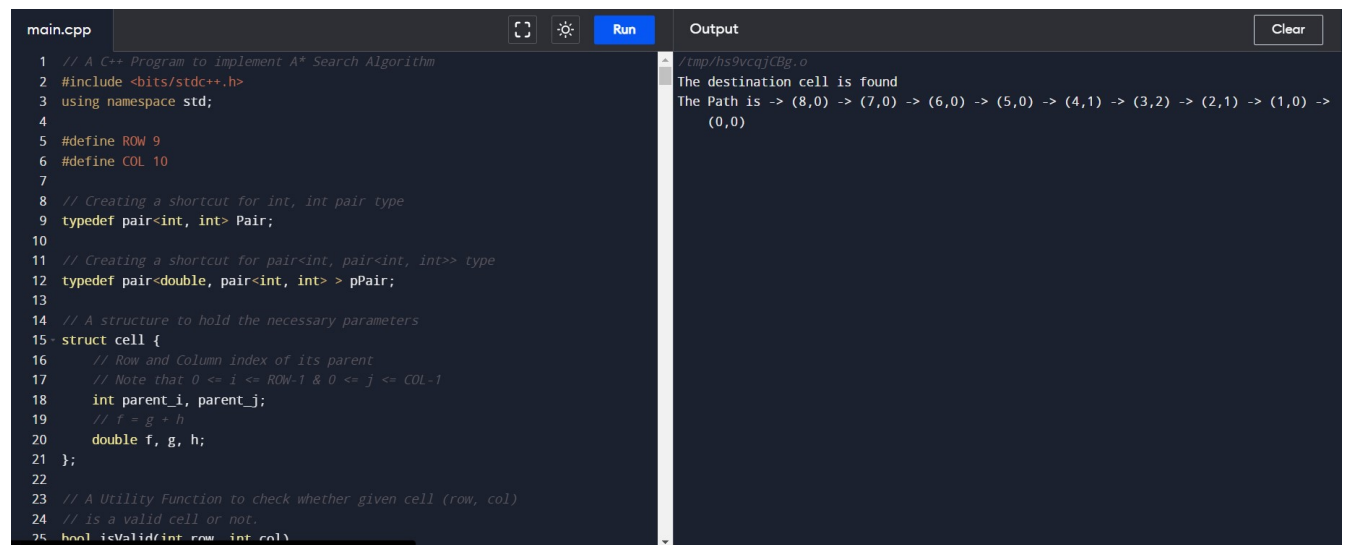
```
# Create a graph given in
```

```
# the above diagram
```

```
g = Graph()  
g.addEdge(0, 1)  
g.addEdge(0, 2)  
g.addEdge(1, 2)  
g.addEdge(2, 0)  
g.addEdge(2, 3)  
g.addEdge(3, 3)
```

```
print("Following is Breadth First Traversal"  
      " (starting from vertex 2)")  
g.BFS(2)
```

### Output:



```
main.cpp Run Output Clear
1 // A C++ Program to implement A* Search Algorithm
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #define ROW 9
6 #define COL 10
7
8 // Creating a shortcut for int, int pair type
9 typedef pair<int, int> Pair;
10
11 // Creating a shortcut for pair<int, pair<int, int>> type
12 typedef pair<double, pair<int, int> > pPair;
13
14 // A structure to hold the necessary parameters
15 struct cell {
16     // Row and Column index of its parent
17     // Note that 0 <= i <= ROW-1 & 0 <= j <= COL-1
18     int parent_i, parent_j;
19     // f = g + h
20     double f, g, h;
21 };
22
23 // A Utility Function to check whether given cell (row, col)
24 // is a valid cell or not.
25 bool isValid(int row, int col)
```

```
/tmp/hs9vcqjCBg.o
The destination cell is found
The Path is -> (8,0) -> (7,0) -> (6,0) -> (5,0) -> (4,1) -> (3,2) -> (2,1) -> (1,0) -> (0,0)
```

### Result:

Successfully DFS and A\* Algorithm .

