EE555 Lab-2

Ragala Chethan

January 25, 2021

Objective

Implementation of Dijkstra algorithm

Problem

Suppose the floor is mapped in the form of a grid as shown in figure-1. The robot has to move from cell ‘S’ to cell ‘D’. Determine the best path to drive the robot. Note that ‘1’ represents that the cell is free to move through and ‘0’ represents that there is an obstacle in the cell.

	0	1	2	3	4
0	1	1	1	1	1
1	D	0	1	0	1
2	1	0	1	0	1
3	1	1	1	1	1
4	0	0	S	0	0

Figure 1: Floor Map

Dijkstra Algorithm

Algorithm: Pseudo code for Dijkstra's Algorithm

1. function Dijkstra(Graph, source):
2. dist[source] := 0 // Distance from source to source
3. for each vertex v in Graph: // Initializations
4. if v $\neq$ source
5. dist[v] := infinity // Unknown distance function from source to v
6. previous[v] := undefined // Previous node in optimal path
7. end if
8. add v to Q // All nodes initially in Q
9. end for
10. while Q is not empty:
11. u := vertex in Q with min dist[u]
12. remove u from Q
13. for each neighbor v of u: //where v has not yet been removed from Q
14. alt := dist[u] + length(u, v)
15. if alt ¡ dist[v]: // A shorter path to v has been found
16. dist[v] := alt
17. previous[v] := u
18. end if
19. end for
20. end while
21. return dist[], previous[]
22. end function

Python code for Dijkstra algorithm

```
import sys

def build_graph(arr):
        graph = {}
        max_row = len(arr)
        max_col = len(arr[0])
        for ind_row,row in enumerate(arr):
                for ind_col,col in enumerate(arr):
                        cell_name = str(ind_row) + str(ind_col)
                        cell_value = col
                        sub_graph = get_weights(arr,ind_row,ind_col,
                                        max_row,max_col,cell_value)
                        graph[cell_name] = sub_graph
        return graph

def get_weights(arr,ind_row,ind_col,max_row,max_col,cell_value):
        weights = {}
```

```
        behind_col_ind = max(0,ind_col-1)
        front_col_ind = min(max_col-1,ind_col+1)
        above_row_ind = max(0,ind_row-1)
        below_row_ind = min(max_row-1,ind_row+1)
        if(behind_col_ind != ind_col):
                weights[str(ind_row)+str(behind_col_ind)] =
                NAND(cell_value,arr[ind_row][behind_col_ind])
        if(front_col_ind != ind_col):
                weights[str(ind_row)+str(front_col_ind)] =
                NAND(cell_value,arr[ind_row][front_col_ind])
        if(above_row_ind != ind_row):
                weights[str(above_row_ind)+str(ind_col)] =
                NAND(cell_value,arr[above_row_ind][ind_col])
        if(below_row_ind != ind_row):
                weights[str(below_row_ind)+str(ind_col)] =
                NAND(cell_value,arr[below_row_ind][ind_col])
        return weights

def NAND(a,b):
        if int(not (a and b)) == 1:
                return float("inf")
        else:
                return 1

def Dijkstra(graph,src):
        Q = []
        dist = {}
        prev = {}
        dist[src] = 0
        u = src
        for v in graph:
                if v != src:
                        dist[v] = 1000
                        prev[v] = -1
                Q.append(v)
        #print(dist)
        #print(prev)
        #print(Q)
        while len(Q) != 0:
                min = 1001
                for i in Q:
                        if min > dist[i]:
                                min = dist[i]
                                u = i
                #print(u)
                Q.remove(u)
```

```
                for v in graph[u]:
                        if (v in Q) & (graph[u][v] == 1):
                                alt = dist[u]+1
                                if alt < dist[v]:
                                        dist[v] = alt
                                        prev[v] = u

        return dist,prev

def find_path(src,dest,dist,prev):
        path = []
        node = dest
        path.append(node)
        while node != src:
                node = prev[node]
                path.append(node)
        path.reverse()
        return path


map = [[1,1,1,1,1],[1,0,1,0,1],[1,0,1,0,1],[1,1,1,1,1],[0,0,1,0,0]]
graph = build_graph(map)
print("The graph is represented by the list:")
print(graph)
print("Enter the source:")
source = raw_input()
if map[int(source[0])][int(source[1])] == 0:
        print("Not a valid source node")
        sys.exit()
#source = '42'
print("Enter the destination:")
destination = raw_input()
if map[int(destination[0])][int(destination[1])] == 0:
        print("Not a valid destination node")
        sys.exit()
#destination = '10'
distance,previous = Dijkstra(graph,source)
print("Distance calculated by the Dijkstra algorithm from source node
to various nodes:")
print(distance)
print("Previous node to every node in the shortest path:")
print(previous)
path = find_path(source,destination,distance,previous)
print("The shortest path from source to node:")
print(path)
```

Output

The graph is represented by the list:
{'22': {'32': 1, '12': 1, '21': inf, '23': inf}, '02': {'03': 1, '12': 1, '01': 1}, '03': {'02': 1, '13': inf, '04': 1}, '00': {'10': 1, '01': 1}, '01': {'02': 1, '11': inf, '00': 1}, '20': {'10': 1, '30': 1, '21': inf}, '21': {'11': inf, '31': 1, '20': 1, '22': 1}, '04': {'03': 1, '14': 1}, '23': {'24': 1, '33': 1, '13': inf, '22': 1}, '44': {'43': inf, '34': 1}, '42': {'32': 1, '43': inf, '41': inf}, '43': {'33': 1, '44': inf, '42': 1}, '40': {'30': 1, '41': inf}, '41': {'31': 1, '42': 1, '40': inf}, '24': {'34': 1, '14': 1, '23': inf}, '11': {'10': 1, '12': 1, '01': 1, '21': inf}, '10': {'11': inf, '00': 1, '20': 1}, '13': {'03': 1, '12': 1, '14': 1, '23': inf}, '12': {'11': inf, '02': 1, '13': inf, '22': 1}, '14': {'24': 1, '13': inf, '04': 1}, '33': {'32': 1, '23': inf, '43': inf, '34': 1}, '32': {'33': 1, '31': 1, '42': 1, '22': 1}, '31': {'32': 1, '30': 1, '21': inf, '41': inf}, '30': {'31': 1, '20': 1, '40': inf}, '34': {'33': 1, '24': 1, '44': inf}}
Enter the source:
    42
Enter the destination:
    10
Distance calculated by the Dijkstra algorithm from source node to various nodes:
{'04': 6, '02': 4, '03': 5, '00': 6, '01': 5, '20': 4, '21': 1000, '22': 2, '23': 1000, '44': 1000, '42': 0, '43': 1000, '40': 1000, '41': 1000, '24': 4, '11': 1000, '10': 5, '13': 1000, '12': 3, '14': 5, '33': 2, '32': 1, '31': 2, '30': 3, '34': 3}
Previous node to every node in the shortest path:
{'04': '03', '02': '12', '03': '02', '00': '01', '01': '02', '20': '30', '21': -1, '22': '32', '23': -1, '44': -1, '43': -1, '40': -1, '41': -1, '24': '34', '11': -1, '10': '20', '13': -1, '12': '22', '14': '24', '33': '32', '32': '42', '31': '32', '30': '31', '34': '33'}
The shortest path from source to node:
['42', '32', '31', '30', '20', '10']