

QUESTION:

Maze game is a well-known problem, where we are given a grid of 0's and 1's, 0's correspond to a place that can be traversed, and 1 corresponds to a place that cannot be traversed (i.e. a wall); the problem is to find a path from bottom left corner of grid to top right corner; immediate right, immediate left, immediate up and immediate down only are possible (no diagonal moves). We consider a variant of the maze problem where a cost is attached to visiting each location in the maze, and the problem is to find a path of least cost through the maze.

Solution

Consider a maze given by a cost c_i such that c_i takes a finite value for traversable locations and ∞ for walls (non-traversable locations), with n rows and m columns.

We convert the maze to a weighted graph $G(V, E)$ as follows: each location (i, j) in the maze corresponds to a node in the graph v_i , and $V = \bigcup v_i$ and $E = \{ (u, v) \mid u, v \in V \}$. Each node v_i is connected to other nodes with weights in the following manner:

$$\begin{aligned} d(v_i, v_{i+1}) &= c_{i+1} \\ d(v_i, v_{i-1}) &= c_{i-1} \\ d(v_i, v_{i+1}) &= c_{i+1} \\ d(v_i, v_{i-1}) &= c_{i-1} \\ d(v_i, v_j) &= \infty \text{ otherwise} \end{aligned}$$

This also means that the weight for an adjacent location in the maze (e.g. $d(v_i, v_{i+1})$) takes the value ∞ if the destination (v_{i+1}) represents a wall. Now, consider a path $t_0, t_1, t_2, \dots, t_k$ in the maze, where $t \in \mathbb{Z} \times \mathbb{Z}$ is a 2D tuple indicating the location (row, col) in the maze; this has corresponding nodes in $G(V, E)$ (say) $u_0, u_1, u_2, \dots, u_k$. Then, the cost of the path $t_0, t_1, t_2, \dots, t_k$ in the maze is

$$c_0 + \sum_{i=0}^{k-1} d(u_i, u_{i+1})$$

where c_0 remains the same for all paths and may be ignored. So, the optimal path minimizes $\sum_{i=0}^{k-1} d(u_i, u_{i+1})$ where $u_0 = v_{n,m}$ and $u_k = v_{1,1}$. This can be accomplished by computing the optimal path from $v_{n,m}$ to $v_{1,1}$ using Dijkstra's algorithm on the graph $G(V, E)$.

Hints: The problem can have multiple solutions. Students can use any design technique such as greedy method, backtracking, dynamic programming. Students can choose their own conditions, positive or negative costs for the graph.

AIM:-To find minimum cost and reach the destination (i.e. from bottom left to top right) in a maze of 0's and 1's with cost attached to each cell and moving in 4 possible directions.

Approach:-Backtracking

Given a grid of $n \times n$ filled with either 0(safe) or 1(un-safe) and a cost matrix with cost to move to respective square in the grid we use 4 recursive calls (up, down, left, right) with two base conditions

- 1-If the path go's out of bound or it tries to revisit the visited cell or if the cell has 1 which is a wall we simply return the function
- 2-If the path reaches the destination then we push the path and the cost into a data structure and simply return the function

Once the path is completed we make the visited vertices un-visited

This recursive approach helps us to find the minimum cost of the path from source to destination

ALGORITHM: DFS

- Create a recursive function that takes the index of node and a visited array.
- Mark the current cell as visited .
- Traverse all the adjacent and unmarked nodes and call the recursive function with index of adjacent node and append to the path and sum up the cost.

CODE:

```
#include <bits/stdc++.h>
using namespace std;
//To check is it possible to make next move to the cell in maze
bool issafe_move(int row, int col, int **m, int r, int c, bool **is_visited){
if (row == -1 || row == r || col == -1 || col == c || is_visited[row][col] || m[row][col]
== 1)
return false;

return true;
```

```

}
//To check all possible combination of path from current position to reach top right
from bottom left
void check_it(int row, int col, int **m, int r,int c, string& path, vector<string>&
possible_paths,bool **is_visited,int **cost, vector<int>& possible_cost, int co)
{
//to check if cell is already visited
if (row == -1 || row == r || col == -1 || col == c || is_visited[row][col] || m[row][col]
== 1)
return;
//To check if we have reached the destination in maze
if (row == 0 && col == c-1)
{
possible_paths.push_back(path); //we store the final path we got in one vector to
print
possible_cost.push_back(co); //we store the cost of the final path we got
return;
}
is_visited[row][col] = true; //mark current position as visited

if (issafe_move(row + 1, col, m, r,c, is_visited)) //check if we can move down
{
path.push_back('D');
co+=cost[row][col];
check_it(row + 1, col, m, r,c,path, possible_paths, is_visited,cost,possible_cost,co);
//recursively call function to find path to reach destination
path.pop_back(); //we delete the last path and search for other possible path
co-=cost[row][col];
}
if (issafe_move(row, col - 1, m, r,c, is_visited)) //check if we can move left
{
path.push_back('L');
co+=cost[row][col];
check_it(row, col - 1, m, r,c,path, possible_paths,
is_visited,cost,possible_cost,co); //recursively call function to find path to reach
destination
path.pop_back();
co-=cost[row][col];
}
if (issafe_move(row, col + 1, m, r,c, is_visited)) //check if we can move right
{
path.push_back('R');
co+=cost[row][col];
check_it(row, col + 1, m, r,c,path, possible_paths, is_visited,cost,possible_cost,co);
//recursively call function to find path to reach destination
path.pop_back();
}

```

```

co-=cost[row][col];
}

if (issafe_move(row - 1, col, m, r,c, is_visited)) //check if we can move up
{
    path.push_back('U');
    co+=cost[row][col];
    check_it(row - 1, col, m, r,c,path, possible_paths,
    is_visited,cost,possible_cost,co);//recursively call function to find path to reach
    destination
    path.pop_back();
    co-=cost[row][col];
}

is_visited[row][col] = false;
}

void solve(int **m, int r,int c, int **cost)
{
    vector<string> possible_paths;
    vector<int> possible_cost;
    int co=0;
    string path;
    //bool is_visited[n][n];
    bool **is_visited=new bool *[r];
    for(int i=0;i<r;i++)
        is_visited[i]=new bool[c];
    //memset(is_visited, false, sizeof(is_visited));
    for(int i=0;i<r;i++)
        for(int j=0;j<c;j++)
            is_visited[i][j]=false; //we initialize the is_visited array to false initially as it is not
            visited still

    check_it(r-1, 0, m, r,c, path,possible_paths, is_visited,cost,possible_cost,co); //call
    function to find all possible path
    if(possible_paths.size()==0) //check if there is no path
    {
        cout<<"Oops Sorry,There is no path available to reach the destination !!(";
    }
    else //if multiple path exists
    {
        int minp=0;
        cout<<"\nPaths available: "<<possible_paths.size()<<" \tcost "<<endl;
        for (int i = 0; i < possible_paths.size(); i++)
        {
            cout << possible_paths[i] << "\t\t\t" <<possible_cost[i]+cost[0][c-1]<<endl;

```

```

if(possible_cost[i]<possible_cost[minp])
{
minp=i; //finding the index of path which has min cost
}
}
cout<<"\nPath with minimum cost is : ";
cout<<"("<<possible_paths[minp]<<") "<<possible_cost[minp]+cost[0][c-1];
string st=possible_paths[minp];
char s[r][c];
for(int i=0;i<r;i++)
for(int j=0;j<c;j++) // To display the path containing min cost in seperate matrix
s[i][j]='.';
s[r-1][0]='@';
int j=r-1,k=0;
for(int i=0;i<st.size();i++){
if(st[i]=='U')
s[--j][k]='@';
else if(st[i]=='R')
s[j][++k]='@';
else if(st[i]=='L')
s[j][--k]='@';
else
s[j++][k]='@';
}
cout<<"\n\nPATH HAVING MIN. COST:\n";
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
cout<<s[i][j]<<" ";
}
cout<<endl;
}
}
int main()
{
int i,j;
cout<<"Enter the number of rows and columns for maze :";
int r,c;
cin>>r>>c; //input no.of rows and columns
int **m=new int *[r];
for(int i=0;i<r;i++)
m[i]=new int[c];
int **cost=new int *[r];
for(int i=0;i<r;i++)
cost[i]=new int[c];
//int m[n][n],cost[n][n];
cout<<"\nEnter the values of each cell in matrix in 0's and 1's:\n";
for(i=0;i<r;i++)

```

```

{
for(j=0;j<c;j++)
{
cin>>m[i][j]; //input the 1's and 0's of maze
}
}
cout<<"\nEnter cost for the maze :\n";
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
cin>>cost[i][j]; //input the cost of each cell
}
}

solve(m, r,c, cost); //solve
cout<<endl;
return 0;
}

```

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  //To check is it possible to make next move to the cell in maze
4  bool issafe_move(int row, int col, int **m, int r,int c, bool **is_visited){
5      if (row == -1 || row == r || col == -1 || col == c || is_visited[row][col] || m[row][col] == 1)
6          return false;
7
8      return true;
9  }
10 //To check all possible combination of path from current position to reach top right from bottom left
11 void check_it(int row, int col, int **m, int r,int c, string& path, vector<string>& possible_paths,bool **is_visited,int **cost,
12 {
13     //to check if cell is already visited
14     if (row == -1 || row == r || col == -1 || col == c || is_visited[row][col] || m[row][col] == 1)
15         return;
16     //To check if we have reached the destination in maze
17     if (row == 0 && col == c-1)
18     {
19         possible_paths.push_back(path); //we store the final path we got in one vector to print
20         possible_cost.push_back(co); //we store the cost of the final path we got
21         return;
22     }
23     is_visited[row][col] = true; //mark current position as visited
24
25     if (issafe_move(row + 1, col, m, r,c, is_visited)) //check if we can move down
26     {
27         path.push_back('D');
28         co+=cost[row][col];
29         check_it(row + 1, col, m, r,c,path, possible_paths, is_visited,co,possible_cost,co); //recursively call function to find path to

```

Windows (CR+LF) WINDOWS-1252 Line 80, Column 10 Insert Read/Write default

```

31 co=cost[row][col];
32 }
33 if (issafe_move(row, col - 1, m, r,c, is_visited)) //check if we can move left
34 {
35     path.push_back('L');
36     co+=cost[row][col];
37     check_it(row, col - 1, m, r,c,path, possible_paths, is_visited,cost,possible_cost,co); //recursively call function to find path to
38     path.pop_back();
39     co-=cost[row][col];
40 }
41 if (issafe_move(row, col + 1, m, r,c, is_visited)) //check if we can move right
42 {
43     path.push_back('R');
44     co+=cost[row][col];
45     check_it(row, col + 1, m, r,c,path, possible_paths, is_visited,cost,possible_cost,co); //recursively call function to find path to
46     path.pop_back();
47     co-=cost[row][col];
48 }
49
50 if (issafe_move(row - 1, col, m, r,c, is_visited)) //check if we can move up
51 {
52     path.push_back('U');
53     co+=cost[row][col];
54     check_it(row - 1, col, m, r,c,path, possible_paths, is_visited,cost,possible_cost,co); //recursively call function to find path to
55     path.pop_back();
56     co-=cost[row][col];
57 }
58
59 is_visited[row][col] = false;

```

Logs & others

Code::Blocks Search results Build log Build messages

Windows (CR+LF) WINDOWS-1252 Line 80, Column 10 Insert Read/Write default

```

61
62 void solve(int **m, int r,int c, int **cost)
63 {
64     vector<string> possible_paths;
65     vector<int> possible_cost;
66     int co=0;
67     string path;
68     //bool is_visited[n][n];
69     bool **is_visited=new bool *[r];
70     for(int i=0;i<r;i++)
71         is_visited[i]=new bool[c];
72     //memset(is_visited, false, sizeof(is_visited));
73     for(int i=0;i<r;i++)
74         for(int j=0;j<c;j++)
75             is_visited[i][j]=false; //we initialize the is_visited array to false initially as it is not visited still
76
77     check_it(r-1, 0, m, r,c, path,possible_paths, is_visited,cost,possible_cost,co); //call function to find all possible path
78     if(possible_paths.size()==0) //check if there is no path
79     {
80         cout<<"\nOops Sorry,There is no path available to reach the destination !!((");
81     }
82     else //if multiple path exists
83     {
84         int minp=0;
85         cout<<"\nPaths available: "<<possible_paths.size()<<" \tcost "<<endl;
86         for (int i = 0; i < possible_paths.size(); i++)
87         {
88             cout << possible_paths[i] << "\t\t\t" <<possible_cost[i]+cost[0][c-1]<<endl;
89             if(possible_cost[i]<possible_cost[minp])

```

Logs & others

Code::Blocks Search results Build log Build messages

Windows (CR+LF) WINDOWS-1252 Line 80, Column 10 Insert Read/Write default

```

91 minp=i; //finding the index of path which has min cost
92 -}
93 -}
94 cout<<"\nPath with minimum cost is : ";
95 cout<<"("<<possible_paths[minp]<<" " "<<possible_cost[minp]+cost[0][c-1];
96 string st=possible_paths[minp];
97 char s[r][c];
98 for(int i=0;i<r;i++)
99 for(int j=0;j<c;j++) // To display the path containing min cost in separate matrix
100 s[i][j]='.';
101 s[r-1][0]='@';
102 int j=r-1, k=0;
103 for(int i=0;i<st.size();i++){
104 if(st[i]=='U')
105 s[--j][k]='@';
106 else if(st[i]=='R')
107 s[j][++k]='@';
108 else if(st[i]=='L')
109 s[j][--k]='@';
110 else
111 s[j++][k]='@';
112 }
113 cout<<"\n\nPATH HAVING MIN. COST:\n";
114 for(int i=0;i<r;i++){
115 for(int j=0;j<c;j++){
116 cout<<s[i][j]<<" ";
117 cout<<endl;
118 }
119 }

```

Logs & others

Code::Blocks Search results Build log Build messages

Windows (CR+LF) WINDOWS-1252 Line 80, Column 10 Insert Read/Write default

```

120 int main()
121 {
122 int i,j;
123 cout<<"Enter the number of rows and columns for maze :";
124 int r,c;
125 cin>>r>>c; //input no. of rows and columns
126 int **m=new int *[r];
127 for(int i=0;i<r;i++)
128 m[i]=new int[c];
129 int **cost=new int *[r];
130 for(int i=0;i<r;i++)
131 cost[i]=new int[c];
132 //int m[n][n],cost[n][n];
133 cout<<"\nEnter the values of each cell in matrix in 0's and 1's:\n";
134 for(i=0;i<r;i++)
135 {
136 for(j=0;j<c;j++)
137 {
138 cin>>m[i][j]; //input the 1's and 0's of maze
139 }
140 }
141 cout<<"\nEnter cost for the maze :\n";
142 for(i=0;i<r;i++)
143 {
144 for(j=0;j<c;j++)
145 {
146 cin>>cost[i][j]; //input the cost of each cell
147 }
148 }
149
150 solve(m, r, c, cost); //solve
151 cout<<endl;
152 return 0;
153 }
154

```

Logs & others

Code::Blocks Search results Build log Build messages

Windows (CR+LF) WINDOWS-1252 Line 80, Column 10 Insert Read/Write default

TEST CASE-1

Input & Output:

```
Enter the number of rows and columns for maze : 5 5

Enter the values of each cell in matrix in 0's and 1's:
1 1 0 0 0
1 0 0 0 1
0 0 1 0 0
0 1 1 0 0
0 0 0 0 1

Enter cost for the maze :
-1 5 4 -2 7
3 8 -1 -5 4
5 9 4 5 3
10 -7 -8 3 4
9 4 -7 3 1

Paths available: 6      cost
RRRURULULURR          27
RRRURULUUR             24
RRRUUULURR             20
RRRUUUUR               17
UURURRUR               40
UURURURR               49

Path with minimum cost is : (RRRUUUUR) 17

PATH HAVING MIN. COST:
. . . @ @
. . . @ .
. . . @ .
. . . @ .
@ @ @ @ .

Process returned 0 (0x0)   execution time : 219.725 s
Press any key to continue.
```

TEST CASE-2

Input & Output:

```
Enter the number of rows and columns for maze :5 5
Enter the values of each cell in matrix in 0's and 1's:
0 0 0 0 1
1 0 0 0 0
0 0 1 0 0
1 1 0 0 1
0 1 0 0 1

Enter cost for the maze :
1 2 3 4 5
-5 -4 -3 -2 -1
3 2 4 3 5
1 1 1 1 1
5 -1 3 1 6
Oops Sorry,There is no path available to reach the destination !!(
Process returned 0 (0x0)   execution time : 68.181 s
Press any key to continue.
```

COMPLEXITY ANALYSIS:

TIME COMPLEXITY:

- 1.Number of total cells=SIZE²
2. Each cell has a maximum of 3 unvisited neighbouring cells
3. Therefore, time complexity= $O(3^{(SIZE^2)})=O(3^{(N^2)})$

SPACE COMPLEXITY:

- 1.As there can only be a maximum of 3 unvisited cells for each cell
2. The space complexity= $O(3^{(N^2)})$

RESULT:

Using the backtracking method, we determined the optimized solution for the maze problem with the cost that will be incurred for the whole traversal.