DESIGN AND ANALYSIS OF ALGORITHM

CLA-P4 Assignment

By: Shravya Sharan (RA1911026010055)

1. **Aim:**

   To find the minimum cost path of a binary maze which has a given cost allotted to each position while travelling from bottom left position to top right position in the maze.

2. **Constraints:**
   - **a.** In binary maze, 1 represents wall and 0 represents traversable position.
   - **b.** Only immediate right, immediate left, immediate up and immediate down moves allowed.
   - **c.** No diagonal moves allowed.
   - **d.** Cost is not equal to -1.

3. **Theory:**
   - **a.** The User inputs the number of rows and columns for the binary maze which is followed by binary maze creation.
   - **b.** The binary maze inputs cost of position when the User input is zero and allots cost as -1 when user input is 1 (or any non-zero number).
   - **c.** The maze and the cost matrix thus formed are displayed.
   - **d.** The travel function uses a simple recursive algorithm. It begins traversing from the source cell (bottom left) and first prints all paths by going down and then prints all paths by going right then prints all paths by going up then prints all paths by going left.
   - **e.** This is done recursively for each cell encountered. Another matrix is used to prevent repetition of paths previously traversed. The matrix stores 1 for position visited and 0 for position to be visited.
   - **f.** Base condition is provided to check for invalid input or walls.
   - **g.** The cost of each path is stored and the minimum value is displayed as result.
   - **h.** In case there are no feasible paths possible between the Source (bottom left position) and Destination (top right position), then No path available is displayed.

4. Code Screenshot:

```
#include <bits/stdc++.h>
using namespace std;

#define INF -1
#define vi vector<vector<int>>

int ROW,COL,CTR=1;
vector<int> PATH_COST;

//Function to compute cost of all Paths from bottom left (Source) to top right (Destination)
void travel(vi grid, vi isValid, int i, int j, vector<int> path={})
{
    //Base Condition
    if (i<0 || j<0 || i>=grid.size() || j>=grid[0].size() || isValid[i][j]==1 || grid[i][j]==INF)
        return;

    //Printing path when destination is reached
    if(i==0&&j==COL-1)
    {
        path.push_back(grid[i][j]); //Inserting the Destination cost
        int total_cost=0;
        cout<<"PATH "<<CTR<<"-> ";
        for(int k=0;k<path.size();k++) //Printing the cost of individual positions in the Path
        {
            cout<<path[k];
            if(k!=path.size()-1) cout<<" + ";
            total_cost+=path[k]; //Computing the cost of Path
        }
        cout<<" = "<<total_cost<<endl;
        PATH_COST.push_back(total_cost); //Storing cost of path traversed
        CTR++;
        return;
    }
    isValid[i][j]=1; //Marking path as traversed
    path.push_back(grid[i][j]); //Storing the path

    //Recursive call
    travel(grid,isValid,i,j+1,path); //Moving right
    travel(grid,isValid,i+1,j,path); //Moving down
    travel(grid,isValid,i-1,j,path); //Moving up
    travel(grid,isValid,i,j-1,path); //Moving left

    path.pop_back();
    isValid[i][j]=0; //Marking path to traverse another path
}

//Main function
int main()
{
    cout<<"Enter the number of ROWS in the maze: ";
    cin>>ROW;
    cout<<"Enter the number of COLUMNS in the maze: ";
    cin>>COL;
    cout<<"Construct the maze. (1->Wall, 0->Path)"<<endl;

    int maze[ROW][COL],pos_cost;
    vector<int> temp;
    vi cost;
```

```
    //Accepting input for Binary Maze and Cost of each position
    for(int i=0;i<ROW;i++)
    {
        for(int j=0;j<COL;j++)
        {
            cout<<"Row: "<<i+1<<" Col: "<<j+1<<": ";
            cin>>maze[i][j];
            if(maze[i][j]==1) temp.push_back(INF);
            else if(maze[i][j]==0)
            {
                cout<<"Enter cost for this position: ";
                cin>>pos_cost;
                temp.push_back(pos_cost);
            }
            else
            {
                cout<<"Incorrect input. Replaced with wall.\n";
                maze[i][j]=1;
                temp.push_back(INF);
            }
        }
        cost.push_back(temp);
        temp.clear();
    }

    //Displaying Binary Maze
    cout<<"\nGiven Maze:\n";
    for(int i=0;i<ROW;i++)
    {
        for(int j=0;j<COL;j++)
        {
            cout<<maze[i][j]<<"\t";
        }
        cout<<endl;
    }

    //Displaying Cost alloted to each position
    cout<<"\nGiven Cost of each location:\n";
    for(int i=0;i<ROW;i++)
    {
        for(int j=0;j<COL;j++)
        {
            cout<<cost[i][j]<<"\t";
        }
        cout<<endl;
    }

    //Using fill constructor to initialize 2-D vector with 0
    vi isValid(ROW, vector<int>(COL,0));

    cout<<"\nComputing cost from Source to Destination... "<<endl;
    travel(cost,isValid,ROW-1,0); //Calling travel function

    if(PATH_COST.size()==0)
        cout<<"No path available.";
    else
        cout << "\nMinimum cost of path from Source to Destination is: "
                    <<*min_element(PATH_COST.begin(), PATH_COST.end());

    return 0;
}
```

5. Output:

a. Case 1: Path available.

```
Enter the number of ROWS in the maze: 4
Enter the number of COLUMNS in the maze: 4
Construct the maze. (1->Wall, 0->Path)
Row: 1 Col: 1: 1
Row: 1 Col: 2: 1
Row: 1 Col: 3: 1
Row: 1 Col: 4: 0
Enter cost for this position: 25
Row: 2 Col: 1: 0
Enter cost for this position: 55
Row: 2 Col: 2: 0
Enter cost for this position: 46
Row: 2 Col: 3: 0
Enter cost for this position: 58
Row: 2 Col: 4: 0
Enter cost for this position: 14
Row: 3 Col: 1: 1
Row: 3 Col: 2: 0
Enter cost for this position: 41
Row: 3 Col: 3: 0
Enter cost for this position: 52
Row: 3 Col: 4: 1
Row: 4 Col: 1: 0
Enter cost for this position: 14
Row: 4 Col: 2: 0
Enter cost for this position: 61
Row: 4 Col: 3: 1
Row: 4 Col: 4: 0
Enter cost for this position: 53
```

```
Given Maze:
1       1       1       0
0       0       0       0
1       0       0       1
0       0       1       0

Given Cost of each location:
-1      -1      -1      25
55      46      58      14
-1      41      52      -1
14      61      -1      53

Computing cost from Source to Destination...
PATH 1-> 14 + 61 + 41 + 52 + 58 + 14 + 25 = 265
PATH 2-> 14 + 61 + 41 + 46 + 58 + 14 + 25 = 259

Minimum cost of path from Source to Destination is: 259
```

b. Case 2: Path Unavailable.

```
Enter the number of ROWS in the maze: 4
Enter the number of COLUMNS in the maze: 4
Construct the maze. (1->Wall, 0->Path)
Row: 1 Col: 1: 1
Row: 1 Col: 2: 1
Row: 1 Col: 3: 0
Enter cost for this position: 10
Row: 1 Col: 4: 0
Enter cost for this position: 19
Row: 2 Col: 1: 0
Enter cost for this position: 37
Row: 2 Col: 2: 0
Enter cost for this position: 46
Row: 2 Col: 3: 1
Row: 2 Col: 4: 0
Enter cost for this position: 72
Row: 3 Col: 1: 0
Enter cost for this position: 55
Row: 3 Col: 2: 1
Row: 3 Col: 3: 0
Enter cost for this position: 58
Row: 3 Col: 4: 0
Enter cost for this position: 37
Row: 4 Col: 1: 0
Enter cost for this position: 91
Row: 4 Col: 2: 0
Enter cost for this position: 73
Row: 4 Col: 3: 1
Row: 4 Col: 4: 0
Enter cost for this position: 25
```

```
Given Maze:
1       1       0       0
0       0       1       0
0       1       0       0
0       0       1       0

Given Cost of each location:
-1      -1      10      19
37      46      -1      72
55      -1      58      37
91      73      -1      25

Computing cost from Source to Destination...
No path available.
```

6. Code:

```
#include <bits/stdc++.h>
using namespace std;

#define INF -1
#define vi vector<vector<int>>

int ROW,COL,CTR=1;
vector<int> PATH_COST;

//Function to compute cost of all Paths from bottom left (Source) to top right (Destination)
void travel(vi grid, vi isValid, int i, int j, vector<int> path={})
{
  //Base Condition
  if (i<0 || j<0 || i>=grid.size() || j>=grid[0].size() || isValid[i][j]==1 || grid[i][j]==INF)
    return;

  //Printing path when destination is reached
  if(i==0&&j==COL-1)
  {
    path.push_back(grid[i][j]); //Inserting the Destination cost
    int total_cost=0;

    cout<<"PATH "<<CTR<<"-> ";
    for(int k=0;k<path.size();k++) //Printing the cost of individual positions in the Path
    {
      cout<<path[k];
      if(k!=path.size()-1) cout<<" + ";
      total_cost+=path[k]; //Computing the cost of Path
    }
    cout<<" = "<<total_cost<<endl;
    PATH_COST.push_back(total_cost); //Storing cost of path traversed
    CTR++;
    return;
  }

  isValid[i][j]=1; //Marking path as traversed
  path.push_back(grid[i][j]); //Storing the path

  //Recursive call
  travel(grid,isValid,i,j+1,path); //Moving right
  travel(grid,isValid,i+1,j,path); //Moving down
  travel(grid,isValid,i-1,j,path); //Moving up
  travel(grid,isValid,i,j-1,path); //Moving left

  path.pop_back();
  isValid[i][j]=0; //Marking path to traverse another path
}
//Main function
```

```
int main()
{
   cout<<"Enter the number of ROWS in the maze: ";
   cin>>ROW;
   cout<<"Enter the number of COLUMNS in the maze: ";
   cin>>COL;
   cout<<"Construct the maze. (1->Wall, 0->Path)"<<endl;

   int maze[ROW][COL],pos_cost;
   vector<int> temp;
   vi cost;

   //Accepting input for Binary Maze and Cost of each position
   for(int i=0;i<ROW;i++)
   {
      for(int j=0;j<COL;j++)
      {
         cout<<"Row: "<<i+1<<" Col: "<<j+1<<": ";
         cin>>maze[i][j];
         if(maze[i][j]==1) temp.push_back(INF);
         else if(maze[i][j]==0)
         {
            cout<<"Enter cost for this position: ";
            cin>>pos_cost;
            temp.push_back(pos_cost);
         }
         else
         {
            cout<<"Incorrect input. Replaced with wall.\n";
            maze[i][j]=1;
            temp.push_back(INF);
         }
      }
      cost.push_back(temp);
      temp.clear();
   }

   //Displaying Binary Maze
   cout<<"\nGiven Maze:\n";
   for(int i=0;i<ROW;i++)
   {
      for(int j=0;j<COL;j++)
      {
         cout<<maze[i][j]<<"\t";
      }
      cout<<endl;
   }



   //Displaying Cost alloted to each position
```

```
    cout<<"\nGiven Cost of each location:\n";
    for(int i=0;i<ROW;i++)
    {
       for(int j=0;j<COL;j++)
       {
          cout<<cost[i][j]<<"\t";
       }
       cout<<endl;
    }

    //Using fill constructor to initialize 2-D vector with 0
    vi isValid(ROW, vector<int>(COL,0));

    cout<<"\nComputing cost from Source to Destination... "<<endl;
    travel(cost,isValid,ROW-1,0); //Calling travel function

    if(PATH_COST.size()==0)
       cout<<"No path available.";
    else
       cout << "\nMinimum cost of path from Source to Destination is:
"<<*min_element(PATH_COST.begin(), PATH_COST.end());

    return 0;
}
```

7. Result:

The code was created and successfully executed.