Avinash reddy Vasipalli

MAZE Problem

Aim:

To create a program which helps us determines the most optimal way to move through a maze problem and the cost attached to it, where we have assumed the cost of each step to be 1.

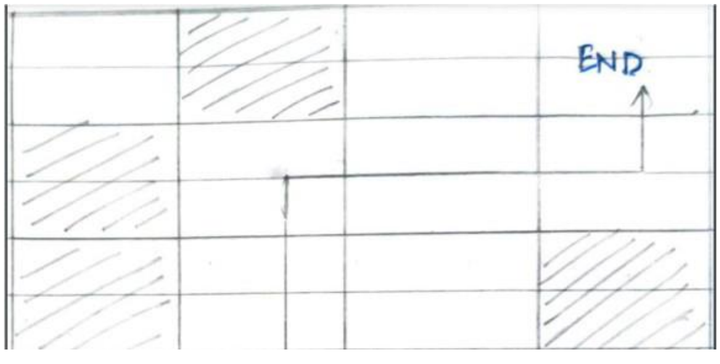
Design technique used:

Backtracking is the design technique that we have used for building the solution to this problem. Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

Algorithm:

1. **Create a solution matrix, initially filled with 0's.**
2. **Create a recursive function, which takes initial matrix, output matrix and position of rat (i, j).**
3. **If the position is out of the matrix or the position is not valid then return.**
4. **Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.**
5. **Recursively call for position (i+1, j) and (i, j+1).**
6. **Unmark position (i, j), i.e output[i][j] = 0.**

Manual solution of the problem:



Source Code:

```
#include <iostream>
using namespace std;
#define SIZE 5
int maze[SIZE][SIZE]=
{
  {1,1,0,1,0},
  {0,0,0,0,0},
  {1,0,1,0,1},
  {0,0,1,0,0},
  {0,1,0,0,1}
};
int solution[SIZE][SIZE];
#funtion to print solution matrix
void printsolution()
{
  int i,j,cost=0;
  for(i=0;i<SIZE;i++)
  {
    for(j=0;j<SIZE;j++)
    {
      printf("%d\t",solution[i][j]);
      if(solution[i][j]==1)
      {
        cost++;
      }
    }
    printf("\n\n");
  }
  printf("cost of this path is: %d",cost);
}

#funtion to solve the maze using backtracking

int solvemaze(int r,int c)
{
  #if destination is reached maze solved then destination is top right
  corner(maze[0][SIZE-1] if((r==0) && (c==SIZE-1))
        {
          solution[r][c] = 1;
          return 1;
        }
        #checking if we can visit the cell
        if(r>=0 && c>=0 && r<SIZE && c<=SIZE && solution[r][c] == 0 && maze[r][c] == 0)
        {
          solution[r][c] = 1;
          #going down
          if(solvemaze(r+1,c))
            return 1;
          #going right
          if(solvemaze(r,c+1))
            return 1;
```

```
  #going up
  if(solvemaze(r-1,c))
    return 1;
  #going left
  if(solvemaze(r,c-1))
    return 1;
  #backtracking
  solution[r][c]=0;
  return 0;
}
return 0;
}
int main()
//Elemets of solution matrix 0
{
  for(i=0;i<SIZE;i++)
  {
    for(j=0;j<SIZE;j++)
    {
      solution[i][j] = 0;
    }
  }
  if(solvemaze(SIZE-1,0))
    printsolution();
  else
    printf("No Solution\n");
  return 0;
```

Test Cases:

1. SIZE=5

Maze= 1 1 0 1 0

00000

10101

00100

01001

Solution= 0 0 0 0

1 01111

01000

11000

10000

Cost of this path is: 9

2. SIZE=4

Maze= 0 1 0 0

1001

0010

0111

Solution=0 0 1 1

0110

1100

1000

Cost of this path is: 7

Complexity Analysis

1. Time Complexity:

Number of total cells=SIZE^2

Each cell has a maximum of 3 unvisited cells

Therefore, time complexity= O(3^(SIZE^2))=O(3^(n^2))

2. Space Complexity:

As there can only be a maximum of 3 unvisited cells
for each cell 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.