**MINI PROJECT REPORT**

Topic: RAT MAZE PROBLEM USING BACKTRACKING

Members: Princeton Baretto(8316)

Sherwin Pillai(8358)

Cassia Vaz(8374)

Problem Statement: Given a maze (2D) matrix in which some cells are blocked. One of the cels is the source node from where we have to start .And another one is destination node where we have to reach. An effective path has to be found between the source node and the destination node withot moving into any of the blocked cells.

Solution: To solve this puzzle, we first start with the source cell and move in a direction where the path is not blocked. If taken path makes us reach to the destination then the puzzle is solved else, we come back and change our direction of the path taken. We represent the puzzle using 2D matrix. The matrix consists of two elements 0 and 1. 1 will represent the blocked cell and 0 will represent the cells in which we can move. The matrix for the maze shown above is:

0 1 0 1 1

0 0 0 0 0

1 0 1 0 1

0 0 1 0 0

1 0 0 1 0

The steps involved for tracing the path are

1. Check for the current cell, if it is the destination cell, if it is,then the puzzle is solved.
2. If not, then we will try to move downward and see if we can move in the downward cell or not (to move in a cell it must be vacant and not already present in the path).
3. If we can move there, then we will continue with the path taken to the next downward cell.
4. If not, we will try to move to the rightward cell. And if it is blocked or taken, we will move upward.
5. Similarly, if we can't move up as well, we will simply move to the left cell.
6. If none of the four moves (down, right, up, or left) are possible, we will simply move back and change our current path (backtracking).

Thus, the summary is that we try to move to the other cell (down, right, up, and left) from the current cell and if no movement is possible, then just come back and change the direction of the path to another cell.

The printsolution ()function is used for printing the solution while solvemaze() is a function where backtracking is implemented and we are checking whether a particular cell can be solved or not. . Firstly, we are checking of our cell is the destination cell or not if (r==SIZE-1) and (c==SIZE-1). If it is the destination cell then our puzzle is already solved. If not, then we are checking if it a valid cell to move or not. A valid cell must be in the matrix i.e., indices must between 0 to SIZE-1 r>=0 && c>=0 && r<SIZE; must not be blocked maze[r][c] == 0 and must not be taken in the path solution[r][c] == 0. If it is a valid move then we are free to take it and move to the next cell. Firstly, we will try the downward cell if(solveMaze(r+1, c)). If it doesn't give us the solution then we will move to the rightward cell, and similarly to the upward and the leftward cells. If all of the cells fail to give us the solution, we will leave the cell solution[r][c] = 0 and go to some other cell.

Code: #include <stdio.h>

#define SIZE 5

//the maze problem

int maze[SIZE][SIZE] = {

{0,1,0,1,1},

{0,0,0,0,0},

{1,0,1,0,1},

{0,0,1,0,0},

{1,0,0,1,0}};

//matrix to store the solution

int solution[SIZE][SIZE];//function to print the solution matrix

void printsolution()

{

int i,j;

for(i=0;i<SIZE;i++)

{

for(j=0;j<SIZE;j++)

{

printf("%d\t",solution[i][j]);

}

printf("\n\n");

}

}

//function to solve the maze

//using backtracking

int solvemaze(int r, int c)

{

//destination is the last cell(maze[SIZE-1][SIZE-1])

if((r==SIZE-1) && (c==SIZE-1))

{

solution[r][c] = 1;

return 1;

}

//checking if we can visit in this cell or not

//the indices of the cell must be in (0,SIZE-1)

//and solution[r][c] == 0 is making sure that the cell is not already visited

//maze[r][c] == 0 is making sure that the cell is not blocked as 1 represents dead end

if(r>=0 && c>=0 && r<SIZE && c<SIZE && solution[r][c] == 0 && maze[r][c] == 0)

{

//if safe to visit then visit the cell

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()

{

//making all elements of the solution matrix 0

int i,j, k,m;

// for printing the initial maze

printf(" THE INITIAL MAZE IS : \n");

for (k=0;k<SIZE;k++)

{

for(m=0;m<SIZE;m++)

{

printf("%d\t ",maze[k][m]);// 1 represnts a dead end

}

printf("\n");

}

printf("\n\n");

printf("The solution matrix that leads the rat to the destination is \n");

// print("\n");

for(i=0; i<SIZE; i++)

{

for(j=0; j<SIZE; j++)

{

solution[i][j] = 0;

}

}

if (solvemaze(0,0))

printsolution();

else

printf("No solution\n");

return 0;

}

output:universe@hp11:~/Desktop$ cc ratm.c

universe@hp11:~/Desktop$ ./a.out

THE INITIAL MAZE IS :

0 1 0 1 1

0 0 0 0 0

1 0 1 0 1

0 0 1 0 0

1 0 0 1 0

The solution matrix that leads the rat to the destination is

1 0 0 0 0

1 1 1 1 0

0 0 0 1 0

0 0 0 1 1

0 0 0 0 1

universe@hp11:~/Desktop$