

COIMBATORE INSTITUTE OF TECHNOLOGY, COIMBATORE

Department of Artificial Intelligence & Data Science

DATA STRUCTURES AND ALGORITHMS

PROJECT REPORT

RAT AND THE MAZE

BY

KAVIN .T

71762108019

ALGORITHM

procedure rat\_maze(int maze[][8],int m)

//maze - (0:m+1,0:m+1)

//m - size of row and column of matrix

[

a <- stack;

i <- 0; j <- 0; k <- 0;

move[8][2]={1,1,1,0,0,1,1,-1,-1,1,0,-1,-1,0,-1,-1};

for(l <- 0.....m)

[

for(n <- 0.....m)

[

check[l][n]=0;

]

]

check[0][0]=1;

a.push(0);

while((i\*j)!=(m-1)\*(m-1))

[

while(k<=7)

[

if(check[i+move[k][0]][j+move[k][1]]==0 && maze[i+move[k][0]][j+move[k][1]]==1 && (j+move[k][1])<=7 && (i+move[k][0])<=7)

[

i=i+move[k][0];

j=j+move[k][1];

check[i][j]=1;

a.push((i\*m)+j);

print(Push i, j);

k=0;

if (i==m-1 && j==m-1)

[

print(Goal Reached);

return TRUE;

]

print(PATH);

continue;

]//end if

k++;

]//end inner while

a.pop();

print(Backtracking PATH);

print(Pop i, j);

k=0;

if(i==0 && j==0)

[

print(Solution does not exist);

]

top=a.top();

i=top/m;

j=top%m;

print(Top i, j);

]//end outer while

]//end procedure

WORKING CODE

#include<stdio.h>

#include<stack>

#include<graphics.h>

#include<conio.h>

#include<iostream>

#define SIZE 100

#define N 10

using namespace std;

int main()

{

int mark[SIZE][SIZE],stack[SIZE][3];

static int

move[8][2]={1,1,-1,1,0,1,-1,0,1,0,1,-1,0,-1,-1,-1};//17 17-160

//DOWN,TOP RIGHT,RIGHT,BOTTOM RIGHT,DOWN,BOTTOM LEFT,LEFT,TOP LEFT

//move[8][2]={1,1,1,0,0,1,1,-1,-1,1,0,-1,-1,0,-1,-1};

int i,m,n,sol3[SIZE][SIZE];

printf("Enter size: ");

scanf("%d %d",&m,&n);

int maze[SIZE][SIZE],b,g,h,top,j,mov;

int blocks;

printf("Enter no of blocks: ");

scanf("%d",&blocks);

int rands[blocks];

int num;

for(int i=1;i<=n;i++)

{

for(int j=1;j<=n;j++)

{

maze[i][j]=0;

sol3[i-1][j-1]=0;

}

}

for(b=0;b<blocks;b++)

{

num=rand()%(n\*n-1);

printf (" %d ", num+1);

rands[b]=num;

}

printf("\n");

int rem,quo;

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

{

rem=rands[i]%n;

quo=rands[i]/n;

maze[quo][rem]=1;

}

maze[1][1]=0;

maze[n][n]=0;

for(int i=1;i<=n;i++)

{

for(int j=1;j<=n;j++)

{

printf("%d, ",maze[i][j]);

sol3[i-1][j-1]=maze[i][j];

}

printf("\n\n");

}

for(i=0;i<=n+1;i++)

maze[0][i]=1;

for(i=0;i<=n+1;i++)

maze[m+1][i]=1;

for(i=0;i<=m+1;i++)

maze[i][0]=1;

for(i=0;i<=m+1;i++)

maze[i][n+1]=1;

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

for(i=0;i<n;i++){

for(j=0;j<n;j++){

printf(" %d ",sol3[i][j]);

}

printf("\n");

}

for(i=1;i<=m;i++)

{

for(j=1;j<=n;j++)

{

mark[i][j]=0;

printf(" %d ",maze[i][j]);

}

printf("\n");

}

int win1=initwindow(2000,2000);

setcurrentwindow(win1);

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

{

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

{

rectangle(i\*40,j\*40, (i\*40)+40, (j\*40)+40);

{

if(sol3[i][j]==1)

{

setfillstyle(SOLID\_FILL,10);

rectangle(j\*40,i\*40,(j\*40)+40,(i\*40)+40);

floodfill(j\*40+1,i\*40+1,WHITE);

continue;

}

setfillstyle(SOLID\_FILL,BLACK);

rectangle(j\*40,i\*40,(j\*40)+40,(i\*40)+40);

floodfill(j\*40+1,i\*40+1,WHITE);

}

}

}

int f2=0;

int f1=0;

setfillstyle(SOLID\_FILL,YELLOW);

rectangle(f2\*40,f1\*40,(f2\*40)+40,(f1\*40)+40);

floodfill(f2\*40+1,f1\*40+1,WHITE);

f1=n-1;

f2=n-1;

setfillstyle(SOLID\_FILL,YELLOW);

rectangle(f2\*40,f1\*40,(f2\*40)+40,(f1\*40)+40);

floodfill(f2\*40+1,f1\*40+1,WHITE);

int cnt2=2;

mark[1][1]=1;

stack[0][0]=1;

stack[0][1]=1;

stack[0][2]=2;

top=1;

i=stack[0][0];

j=stack[0][1];

while(top!=0)

{

mov=stack[0][2];

top=top-1;

while(mov<=7)//returns to the start position if all the direction not exists.

{

g=i+move[mov][0];

h=j+move[mov][1];

if(mark[g][h]==0 && maze[g][h]==0)

{

printf(" PATH : \n");

int x=g-1,y=h-1;

setfillstyle(SOLID\_FILL,9);

rectangle(y\*40,x\*40,(y\*40)+40,(x\*40)+40);

floodfill(y\*40+1,x\*40+1,WHITE);

setfillstyle(SOLID\_FILL,YELLOW);

rectangle(f2\*40,f1\*40,(f2\*40)+40,(f1\*40)+40);

floodfill(f2\*40+1,f1\*40+1,WHITE);

delay(500);

mark[g][h]=1;

top++;

cnt2=cnt2+1;

stack[top][0]=i;

stack[top][1]=j;

mov=-1;

printf(" %d %d ",g,h);

printf("\n");

i=g;

j=h;

}

mov=mov+1;

if(g==m&&h==n)

{

printf("path made by the rat is\n");

for(i=1;i<=top;i++){

printf("(%d,%d)",stack[i][0],stack[i][1]);

printf("\n");

}

printf("(%d,%d)",m,n);

int sol[SIZE][SIZE];

for(i=1;i<=n;i++){

for(j=1;j<=n;j++){

sol[i][j]=0;

}

}

printf("\n");

int cnt=1;

for(i=1;i<=top;i++){

sol[stack[i][0]][stack[i][1]]=cnt;

cnt++;

}

sol[n][n]=cnt;

for(i=1;i<=n;i++){

for(j=1;j<=n;j++){

printf(" %d, ",sol[i][j]);

}

printf("\n");

}

delay(3000);

exit(0);

}

if(mov>7){

printf(" top: %d ",top);

i=stack[top][0];

j=stack[top][1];

int x=i-1,y=j-1;

//grapho(i-1,j-1,n-1,n-1,0,sol3,10);

setfillstyle(SOLID\_FILL,WHITE);

rectangle(y\*40,x\*40,(y\*40)+40,(x\*40)+40);

floodfill(y\*40+1,x\*40+1,WHITE);

delay(500);

cnt2=cnt2+1;

printf(" backtracking : %d %d ",i,j);

printf("\n");

}

}

}

}

OUTPUT

WHITE- BACKTRACKING

PURPLE-PATH

GREEN-BLOCKS

YELLOW – START AND DESTINATION



