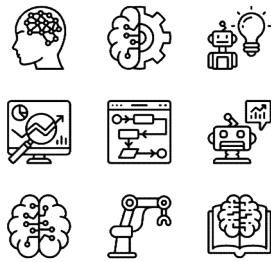


Computer Science for Practicing Engineers

Một số bài toán giải bằng phương pháp quay lui



TS. Huỳnh Bá Diệu
Email: dieuhb@gmail.com
Phone: 0914146868

Backtracking Problems

1. Sudoku
2. Rat in a Maze
3. Count number of ways to reach destination in a Maze
- 4. *Count number of ways to reach destination in a maze 2***

Backtracking - Sudoku

Given a partially filled 9×9 2D array 'grid[9][9]', the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.



Backtracking - Sudoku

Naive Algorithm

The Naive Algorithm is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found.

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Trò chơi theo nhóm – Giải ô Sudoku

Ta có thể giải bài toán sudoku bằng thử cách gán từng số một vào các ô trống trên bảng.

Trước khi gán 1 số, ta cần kiểm tra số đó có gán được hay không (*safe to assign*).

4	1		2	7		8		5
	8	5	1	4	6		9	7
	7		5	8			4	
9	2	7	4	5	1	3	8	6
5	3	8	6	9	7	4	1	2
1	6	4	3	2	8	7	5	9
8	5	2	7		4	9		
	9		8		2	5	7	4
7	4		9	6	5		2	8

Trò chơi theo nhóm – Giải ô Sudoku

Hãy cho biết lời giải bảng sau trong thời gian nhanh nhất!!!



4	1		2	7		8		5
	8	5	1	4	6		9	7
	7		5	8			4	
9	2	7	4	5	1	3	8	6
5	3	8	6	9	7	4	1	2
1	6	4	3	2	8	7	5	9
8	5	2	7		4	9		
	9		8		2	5	7	4
7	4		9	6	5		2	8

Backtracking Algorithm - Sudoku

Một số được gọi là an toàn khi gán vào ô nếu như số đó chưa được dùng (trong hàng, trong cột và trong khối 3X3).

Chúng ta thử gán số đó vào ô trống rồi tiếp tục tìm các ô trống khác để thử.

Trong trường hợp việc gán thử đó có thể dẫn đến lời giải (các ô được điền xong các số) thì return true, ngược lại ta quay lại chọn phương án số khác để gán vào ô trống.

Trong trường hợp các phép thử đều không cho lại kết quả thì return false (không có lời giải)

3	6	5	8	4				
5	2							
8	7					3	1	
		3	1			8		
9		8	6	3			5	
	5		9		6			
1	3				2	5		
						7	4	
	5	2	6	3				

Backtracking Algorithm - Sudoku

Cách làm:

- + Tìm ô chưa được gán giá trị [hàng, cột]
- + Nếu không có ô nào như vậy thì **return true**

Xét các số từ 1 đến 9

a) Chọn 1 số, nếu số đó không xung đột (conflict) với các số ở cùng hàng, cùng cột, cùng khối (3X3) thì gán số đó cho ô trống, tiếp tục tìm đệ quy cho các ô còn lại trên bảng

b) Nếu việc đệ quy thành công thì return true

c) Ngược lại bỏ số vừa chọn thử ở trên và chọn số khác

Nếu các số đều đã thử nhưng không thành công thì return false

1	2	3
6		4
7	8	9

Backtracking - Sudoku

```
bool SolveSudoku(int grid[9][9]) {
    int row, col;
    if (!EmptyCell(grid, row, col)) return true;
    for (int num = 1; num <= 9; num++)
        if (isSafe(grid, row, col, num)) { // số num có thể đặt vào ô r,c
            grid[row][col] = num;
            if (SolveSudoku(grid)) return true; // đã điền hết thì trả về true
            grid[row][col] = 0;
        }
    return false;
}
```

Backtracking - Sudoku

```
bool EmptyCell(int grid[9][9], int &row, int &col)
{
    for (row = 0; row < 9; row++)
        for (col = 0; col < 9; col++)
            if (grid[row][col] == 0)
                return true;
    return false;
}
```

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Lưu ý hàm sử dụng hai biến row, col là hai tham biến. Nếu có ô chưa gán giá trị thì sẽ lưu vị trí ô lại để xử lý!

Backtracking - Sudoku

```
bool isSafe(int grid[N][N], int row, int col, int num) { }
```

```
bool UsedInBox(int grid[N][N], int SR, int SC, int num) { }
```

```
bool UsedInRow(int grid[N][N], int row, int num) { }
```

```
bool UsedInCol(int grid[N][N], int col, int num){ }
```

				7	5		
7			1			4	
5					2		
		1	3	9			8
3			7	8	6		4
8			4	1	7		
		8					9
	5			3			1
		4	6				

Backtracking - Sudoku

3		6	5	8	4		
5	2						
	8	7				3	1
		3		1			8
9			8	6	3		5
	5			9		6	
1	3					2	5
		5	2	6	3		4

```
bool isSafe(int grid[9][9], int row, int col, int num)
{
    return    !UsedInRow(grid, row, num) &&
               !UsedInCol(grid, col, num) &&
               !UsedInBox(grid, row - row % 3 , col - col % 3, num) &&
               grid[row][col] == 0;
}
```

Backtracking - Sudoku

```
bool UsedInBox(int grid[9][9], int SR, int SC, int num) {
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row + SR][col + SC] == num) return true;
    return false;
}
```

4	1		2	7		8		5
	8	5	1	4	6		9	7
	7		5	8			4	
9	2	7	4	5	1	3	8	6
5	3	8	6	9	7	4	1	2
1	6	4	3	2	8	7	5	9
8	5	2	7		4	9		
	9		8		2	5	7	4
7	4		9	6	5		2	8

Backtracking - Sudoku

```
bool UsedInRow(int grid[9][9], int row, int num) {
    for (int col = 0; col < 9; col++)
        if (grid[row][col] == num) return true;
    return false;
}

bool UsedInCol(int grid[9][9], int col, int num) {
    for (int row = 0; row < 9; row++)
        if (grid[row][col] == num) return true;
    return false;
}
```

3		6	5	8	4			
5	2							
	8	7				3	1	
		3		1		8		
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2	6	3			

Backtracking - Sudoku

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

```
void printGrid(int grid[9][9]) {
    for (int row = 0; row < 9; row++) {
        for (int col = 0; col < 9; col++) cout << grid[row][col] << " ";
        cout << endl;
    }
}
```

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Backtracking - Sudoku

```
int main() {
    int grid[9][9] = {{3, 0, 6, 5, 0, 8, 4, 0, 0},
                      {5, 2, 0, 0, 0, 0, 0, 0, 0},
                      {0, 8, 7, 0, 0, 0, 0, 3, 1},
                      {0, 0, 3, 0, 1, 0, 0, 8, 0},
                      {9, 0, 0, 8, 6, 3, 0, 0, 5},
                      {0, 5, 0, 0, 9, 0, 6, 0, 0},
                      {1, 3, 0, 0, 0, 0, 2, 5, 0},
                      {0, 0, 0, 0, 0, 0, 0, 7, 4},
                      {0, 0, 5, 2, 0, 6, 3, 0, 0}};

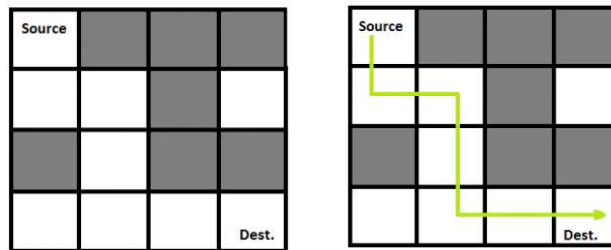
    if (SolveSudoku(grid) == true) printGrid(grid); else cout << "No solution exists";
    return 0;
}
```

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Backtracking - Rat in a Maze

A Maze is given as $N \times N$ binary matrix of blocks where source block is the upper left most block i.e., `maze[0][0]` and destination block is lower rightmost block i.e., `maze[N-1][N-1]`.

A rat starts from source and has to reach the destination. The rat can move only in two directions: forward and down.



Backtracking - Rat in a Maze

Trong ma trận maze, giá trị 0 có nghĩa là điểm không đi được (dead end), điểm 1 có thể đi được.

Đây là phiên bản đơn giản. Trường hợp tổng quát, con chuột có thể có thể đi theo 4 hướng.

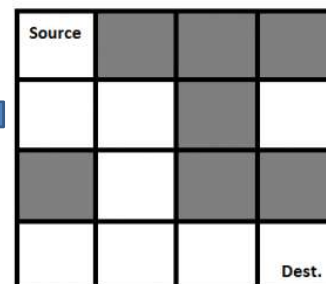
{1, 0, 0, 0}

{1, 1, 0, 0}

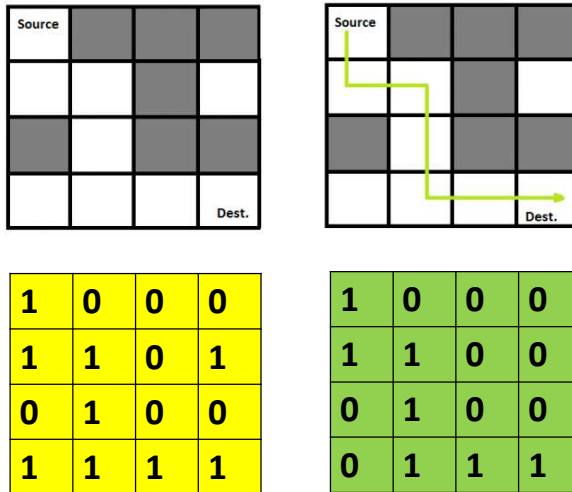
{0, 1, 0, 0}

{0, 1, 1, 1}

All entries in solution path are marked as 1.



Backtracking - Rat in a Maze



Backtracking Algorithm - Rat in a Maze

Nếu ô đang đứng là đích (destination) thì in ma trận kết quả (solution matrix)
Ngược lại

- Đánh dấu ô đang đứng (trong **solution** matrix) là 1 // *chọn ô này trên đường đi*
- Di chuyển đến đích **theo hướng ngang** và tiếp tục di chuyển đệ qui như vậy. Kiểm tra xem cách di chuyển đó có đi đến đích không
- Nếu cách di chuyển ở trên không đi đến đích thì **di chuyển theo hướng xuống** và tiếp tục gọi đệ qui như vậy và kiểm tra có đến đích hay không
- Nếu hai cách trên không thể dẫn đến kết quả thì đánh dấu ô đang chọn trong ma trận kết quả là 0 (backtrack, hàm ý không chọn cách đi này) và return false.

Backtracking - Rat in a Maze

```
bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]) {
    if (x == N - 1 && y == N - 1) { sol[x][y] = 1; return true; } // đã đến đích
    if (isSafe(maze, x, y) == true) { // nếu ô đang đứng là an toàn
        sol[x][y] = 1;
        if (solveMazeUtil(maze, x + 1, y, sol) == true) return true; // di chuyển ngang
        if (solveMazeUtil(maze, x, y + 1, sol) == true) return true; // di chuyển xuống
        sol[x][y] = 0; return false; // quay lui
    }
    return false; // trường hợp không có lời giải
}
```

Backtracking - Rat in a Maze

```
bool isSafe(int maze[N][N], int x, int y)
{
    // nếu vị trí đứng là trong bảng và ô chứa số 1
    if (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1)
        return true;
    return false;
}
```

Backtracking - Rat in a Maze

```
#define N 4
int sol[4][4];
bool solveMaze(int maze[N][N]) {    }
int main() {
    int maze[N][N] = {    { 1, 0, 0, 0 },
                           { 1, 1, 0, 1 },
                           { 0, 1, 0, 0 },
                           { 1, 1, 1, 1 } };
    solveMaze(maze);    return 0;
}
```

Source			
			Dest.

1	0	0	0
1	1	0	1
0	1	0	0
1	1	1	1

Backtracking - Rat in a Maze

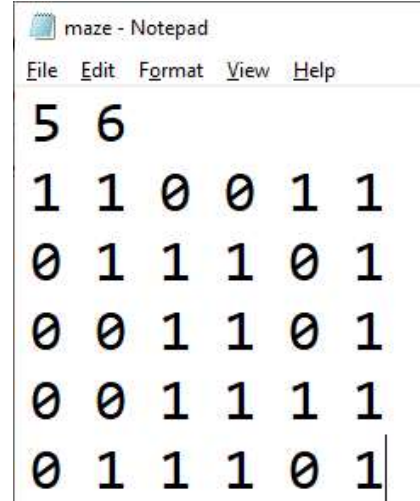
```
void printSolution(int sol[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) cout<<sol[i][j]<<" ";
        cout<<"\n";
    }
}

bool solveMaze(int maze[N][N]) {
    if (solveMazeUtil(maze, 0, 0, sol) == false) { printf("No Solution!"); return false; }
    printSolution(sol); return true;
}
```

Backtracking - Rat in a Maze

Xử lý maze với số dòng cột tùy ý

```
using namespace std;
int M,N; // so hang va so cot cua ma tran
int **maze;
int **sol;
```



```
maze - Notepad
File Edit Format View Help
5 6
1 1 0 0 1 1
0 1 1 1 0 1
0 0 1 1 0 1
0 0 1 1 1 1
0 1 1 1 0 1
```

Backtracking - Rat in a Maze: Đọc ma trận từ file

```
void docfile(char* tenfile) {
    ifstream fi(tenfile); if(fi==NULL) { cout<<" loi doc file!"; return ;}
    fi>>M>>N;
    maze= new int*[M]; sol= new int*[M]; //cap phat bo nho cho hai mang
    for(int i=0; i<M; i++) { maze[i]= new int[N]; sol[i]=new int[N]; }
    // doc du lieu tu file vao ma tran
    for(int i=0; i<M; i++)
        for(int j=0; j<N; j++) { fi>>maze[i][j]; sol[i][j]=0;}
    fi.close();
}
```

Backtracking - Rat in a Maze: Đọc ma trận từ file

```
bool isSafe(int **maze, int x, int y)
{
    // neu vi trí dung là trong bang và ô chưa số 1
    if (x >= 0 && x < M && y >= 0 && y < N && maze[x][y] == 1)
        return true;
    return false;
}
```

Backtracking - Rat in a Maze: Đọc ma trận từ file

```
bool solveMazeUtil(int **maze, int x, int y, int **&sol) {
    if (x == M - 1 && y == N - 1) { sol[x][y] = 1; return true; } // đã đến đích
    if (isSafe(maze, x, y) == true) {
        sol[x][y] = 1;
        if (solveMazeUtil(maze, x + 1, y, sol) == true) return true; // đi chuy?n xuống
        if (solveMazeUtil(maze, x, y + 1, sol) == true) return true; // đi chuy?n ngang
        sol[x][y] = 0; return false; // // tru?ng h?p không có l?i gì?i
    }
    return false;
}
```

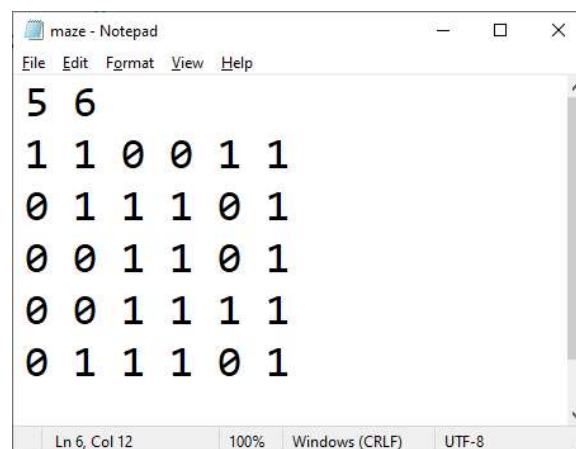
Backtracking - Rat in a Maze: Đọc ma trận từ file

```
void printSolution(int **sol) {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++)    printf(" %d ", sol[i][j]);
        printf("\n");
    }
}

bool solveMaze(int **maze, int **&sol) {
    if (solveMazeUtil(maze, 0, 0, sol) == false) { printf("Solution doesn't exist"); return false; }
    printSolution(sol); return true;
}
```

Backtracking - Rat in a Maze: Đọc ma trận từ file

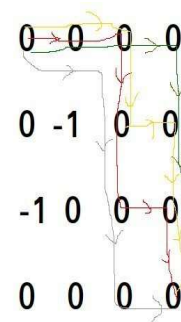
```
int main() {
    docfile("E:\\maze.txt");
    solveMaze(maze, sol);
    return 0;
}
```



Count number of ways to reach destination in a Maze

Given a maze with obstacles (*chướng ngại vật*), count number of paths to reach rightmost-bottommost cell from topmost-leftmost cell. A cell in given maze has value -1 if it is a blockage (*bao vây*) or dead end, else 0. From a given cell, we are allowed to move to cells $(i+1, j)$ and $(i, j+1)$ only.

Theo bạn, có bao nhiêu cách đi cho hình bên???



-1 represents blockage.

$(0,0)$ is source.

$(3,3)$ is destination.

Count number of ways to reach destination in a Maze

Input: `maze[R][C] =`

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

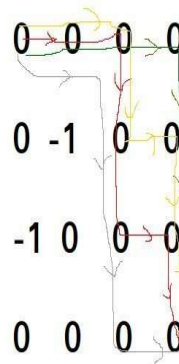
`{0, -1, 0, 0},`

`{-1, 0, 0, 0},`

`{0, 0, 0, 0};;`

Output: 4

There are four possible paths.



-1 represents blockage.

$(0,0)$ is source.

$(3,3)$ is destination.

There are total four paths from source to destination

Count number of ways to reach destination in a Maze

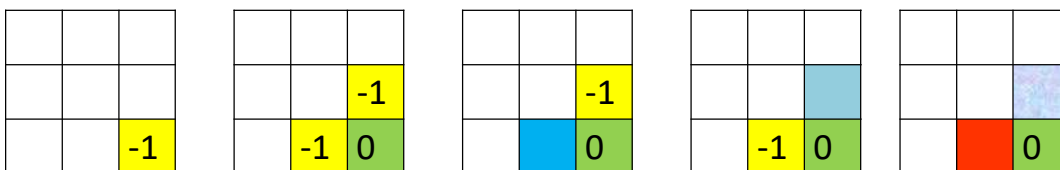
The idea is to modify the given `grid[][]` so that `grid[i][j]` contains count of paths to reach `(i, j)` from `(0, 0)` if `(i, j)` is not a blockage, else `grid[i][j]` remains -1.

Dựa trên gợi ý trên, hãy thảo luận theo nhóm về giải pháp để giải bài toán. Nêu các trường hợp có thể có !!!



Count number of ways to reach destination in a Maze

The idea is to modify the given `grid[][]` so that `grid[i][j]` contains count of paths to reach `(i, j)` from `(0, 0)` if `(i, j)` is not a blockage, else `grid[i][j]` remains -1.



Count number of ways to reach destination in a Maze

The idea is to modify the given grid[][] so that **grid[i][j]** contains count of paths to reach (i, j) from (0, 0) if (i, j) is not a blockage, else grid[i][j] remains -1. We can recursively compute grid[i][j] using below formula and finally return grid[R-1][C-1]

// If current cell is a blockage

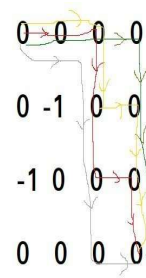
if (maze[i][j] == -1) maze[i][j] = -1; // Do not change

// If we can reach maze[i][j] from maze[i-1][j] then increment count.

else {if (maze[i-1][j] > 0) maze[i][j] = (maze[i][j] + maze[i-1][j]);

// If we can reach maze[i][j] from maze[i][j-1] then increment count.

if (maze[i][j-1] > 0) maze[i][j] = (maze[i][j] + maze[i][j-1]);}



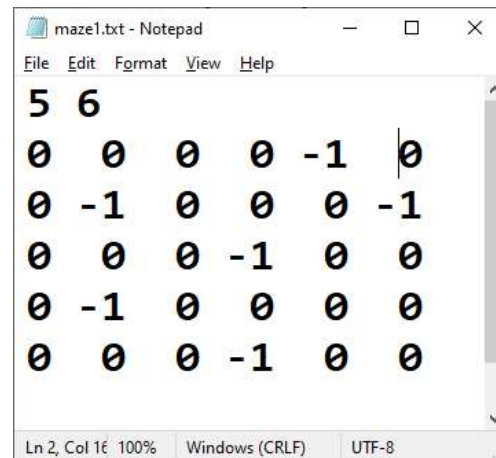
Count number of ways to reach destination in a Maze

```
int countPaths(int **maze) {
    if (maze[0][0] == -1) return 0;
    for (int i=0; i<R; i++) if (maze[i][0] == 0) maze[i][0] = 1; else break;
    for (int i=1; i<C; i++) if (maze[0][i] == 0) maze[0][i] = 1; else break;
    for (int i=1; i<R; i++)
        for (int j=1; j<C; j++) {
            if (maze[i][j] == -1) continue;
            if (maze[i-1][j] > 0) maze[i][j] = (maze[i][j] + maze[i-1][j]);
            if (maze[i][j-1] > 0) maze[i][j] = (maze[i][j] + maze[i][j-1]);
        }
    return (maze[R-1][C-1] > 0)? maze[R-1][C-1] : 0;
}
```

Cho biết độ phức tạp của thuật toán countPaths???

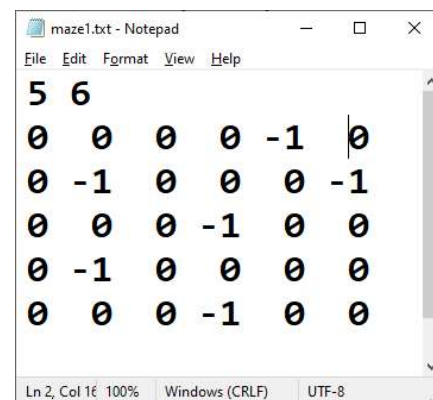
Count number of ways to reach destination in a Maze

```
int **maze, R, C;
int main() {
    docfile("E:\\maze1.txt");
    cout<<countPaths (maze);
    return 0;
}
```



Count number of ways to reach destination in a Maze

```
void docfile(char* tenfile) {
    ifstream fi(tenfile); if(fi==NULL) { cout<<" loi doc file!"; return ;}
    fi>>R>>C;
    maze= new int*[R]; // cap phat bo nho
    for(int i=0; i<R; i++) maze[i]= new int[C];
    // doc du lieu tu file vao ma tran
    for(int i=0; i<R; i++)
        for(int j=0; j<C; j++) fi>>maze[i][j];
    fi.close();
}
```



Backtracking - *Count number of ways to reach destination in a maze 2*

Given a maze of 0 and -1 cells, the task is to find all the paths from (0, 0) to (m-1, n-1), and every path should pass through at least one cell which contains -1. From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only.

Input: `maze[][] = {`

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

`{0, -1, 0, 0},`

`{-1, 0, 0, 0},`

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

Output: 16

Backtracking - *Count number of ways to reach destination in a maze 2*

Given a maze of 0 and -1 cells, the task is to find all the paths from (0, 0) to (m-1, n-1), and every path should pass through at least one cell which contains -1.

Dựa trên ý tưởng giải bài trước, hãy thảo luận theo nhóm về giải pháp để giải bài toán này!!!



Backtracking - *Count number of ways to reach destination in a maze 2*

Approach:

To find all the paths which go through at least one marked cell (cell containing -1). If we find the paths that do not go through any of the marked cells and all the possible paths from (0, 0) to (m-1, n-1) then we can find all the paths that go through at least one of the marks cells.

Number of paths that pass through at least one marked cell = (Total number of paths – Number of paths that do not pass through any marked cell)

Backtracking - *Count number of ways to reach destination in a maze 2*

A= Total number of paths

B= Number of paths that pass through at least one marked cell

C= Number of paths that do not pass through any marked cell)

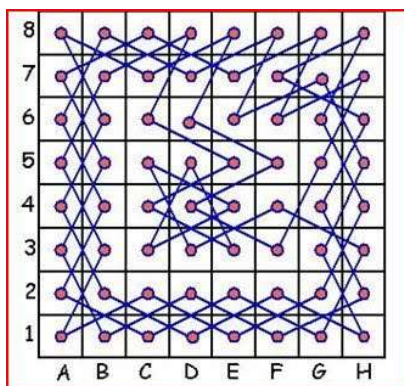
$$\rightarrow B = A - C$$

Bài tập về nhà

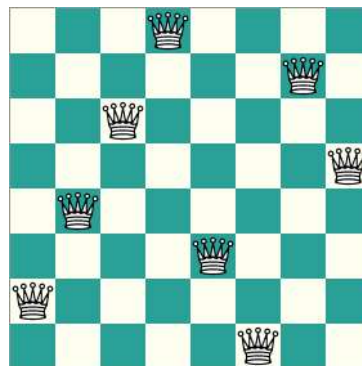
- 1/ Implement a program to *count number of ways to reach destination in a maze* 2.
- 2/ Find the solution for the Knight Tour Problem.
- 3/ Find the solution for the EightQueen Problem.

Bài tập về nhà

Knight Tour Problem



Eight Queen Problem



Link YouTube

<https://www.youtube.com/watch?v=xouin83ebxE>

<https://www.youtube.com/watch?v=wGbuCyNpxlg>

