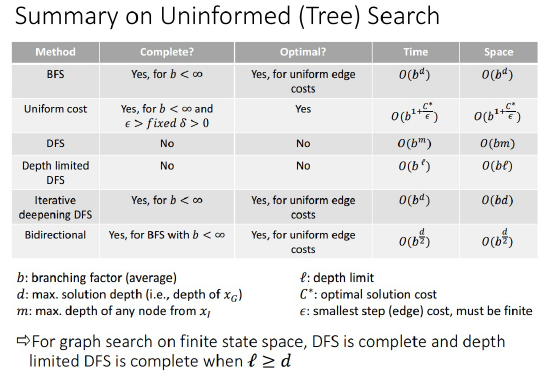
**BFS DFS分析**



BFS / DFS都可以做

130. Surrounded Regions

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

 -> 

**S1. DFS**

先把靠墙的’O’做上标记，

然后把没被标记的全部标为’X’，顺便把被标记的改回来

public void solve(char[][] board) {

if (board == null || board.length < 3 || board[0].length < 3)

return;

int m = board.length, n = board[0].length;

*// 1.check the 'O next to the wall, mark them to '1',*

for (int i = 0;i < m;i++) {

if (board[i][0] == 'O') dfs(board, i, 0);

if (board[i][n - 1] == 'O') dfs(board, i, n - 1);

}

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

if (board[0][i] == 'O') dfs(board, 0, i);

if (board[m - 1][i] == 'O') dfs(board, m - 1, i);

}

*// 2. all the 'O' not marked would be set to 'X',*

*// set '1' back to 'O'.*

for (int i = 0;i < m;i++) {

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

if (board[i][j] == '1') {

board[i][j] = 'O';

} else {

board[i][j] = 'X';

}

}

}

}

private void dfs(char[][] board, int i, int j) {

if (board[i][j] != 'O') return;

board[i][j] = '1';

if (i > 1) dfs(board, i - 1, j);

if (i < board.length - 1) dfs(board, i + 1, j);

if (j > 1) dfs(board, i, j - 1);

if (j < board[0].length - 1) dfs(board, i, j + 1);

}

547. Friend Circles

There are N students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a direct friend of B, and B is a direct friend of C, then A is an indirect friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a N\*N matrix M representing the friend relationship between students in the class. If M[i][j] = 1, then the ith and jth students are direct friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

[[1,1,0],

[1,1,0],

[0,0,1]]

Output: 2

Explanation:The 0th and 1st students are direct friends, so they are in a friend circle.

The 2nd student himself is in a friend circle. So return 2.

**S1. Union-Find**

可以看成一个连通性的问题，如果某个点M[i][j]为1，就可以把i, j看作属于一个group

class Solution {

public int findCircleNum(int[][] M) {

if (M == null || M.length == 0) return 0;

int n = M.length;

UnionFind uf = new UnionFind(n);

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

for (int j = 0;j < i;j++) {

if (M[i][j] == 1) uf.union(i, j);

}

}

return uf.count();

}

}

class UnionFind {

private int[] id;

private int[] size;

private int count;

public UnionFind(int n) {

id = new int[n];

size = new int[n];

Arrays.fill(size, 1);

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

id[i] = i;

}

count = n;

}

private int find(int p) { *// find root of p*

while (p != id[p]) {

id[p] = id[id[p]]; *// compression*

p = id[p];

}

return p;

}

public boolean connected(int p, int q) {

return find(p) == find(q);

}

public void union(int p, int q) {

int rp = find(p), rq = find(q);

if (rp == rq) return;

if (size[rp] > size[rq]) {

id[rq] = rp;

size[rp] += size[rq];

} else {

id[rp] = rq;

size[rq] += size[rp];

}

count--;

}

public int count() {

return count;

}

}

**S2. DFS**

这道题好像和number of islands是完全一样的，做法如下

public int findCircleNum(int[][] M) {

if (M == null || M.length == 0) return 0;

int m = M.length, n = M[0].length;

int count = 0;

for (int i = 0;i < m;i++) {

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

if (M[i][j] == 1) {

count++;

dfs(M, i, j);

}

}

}

return count;

}

private void dfs(int[][] nums, int i, int j) {

nums[i][j] = 0;

if (i > 0 && nums[i-1][j] == 1) dfs(nums, i-1, j);

if (i < nums.length - 1 && nums[i+1][j] == 1) dfs(nums, i+1, j);

if (j > 0 && nums[i][j-1] == 1) dfs(nums, i, j-1);

if (j < nums[0].length - 1 && nums[i][j+1] == 1) dfs(nums, i, j+1);

}

690. Employee Importance

You are given a data structure of employee information, which includes the employee's unique id, his importance value and his direct subordinates' id.

For example, employee 1 is the leader of employee 2, and employee 2 is the leader of employee 3. They have importance value 15, 10 and 5, respectively. Then employee 1 has a data structure like [1, 15, [2]], and employee 2 has [2, 10, [3]], and employee 3 has [3, 5, []]. Note that although employee 3 is also a subordinate of employee 1, the relationship is not direct.

Now given the employee information of a company, and an employee id, you need to return the total importance value of this employee and all his subordinates.

Input: [[1, 5, [2, 3]], [2, 3, []], [3, 3, []]], 1，Output: 11

Employee 1 has importance value 5, and he has two direct subordinates: employee 2 and employee 3. They both have importance value 3. So the total importance value of employee 1 is 5 + 3 + 3 = 11.

**S1. BFS**

这是非常典型的，套着故事的皮的BFS问题了，从id开始逐级往下找-一遍bug free.

public int getImportance(List<Employee> employees, int id) {

if (employees == null || employees.size() == 0) return 0;

Map<Integer, Employee> map = new HashMap<>(); *// <id, employee>*

for (Employee e:employees) {

map.put(e.id, e);

}

if (!map.containsKey(id)) return 0;

int res = 0;

Queue<Employee> queue = new LinkedList<>();

queue.add(map.get(id));

while (!queue.isEmpty()) {

int size = queue.size();

for (int i = 0;i < size;i++) {

Employee cur = queue.poll();

res += cur.importance;

for (int suborinate:cur.subordinates) {

queue.add(map.get(suborinate));

}

}

}

return res;

}

BFS

542. 01 Matrix

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

input: output:

Note:

The number of elements of the given matrix will not exceed 10,000.

There are at least one 0 in the given matrix.

The cells are adjacent in only four directions: up, down, left and right.

**S1. BFS**

题目非常适合用BFS的方法做，一开始把0的点的坐标位置加入即可，其他设为Integer.MAX

然后BFS一层层往下找，每次对每个位置求min

public int[][] updateMatrix2(int[][] matrix) {

if (matrix == null || matrix.length == 0) return matrix;

int m = matrix.length, n = matrix[0].length;

Queue<int[]> queue = new LinkedList<>();

for (int i = 0; i < m; i++) {

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

if (matrix[i][j] == 0) {

queue.offer(new int[] {i, j});

} else {

matrix[i][j] = Integer.MAX\_VALUE;

}

}

}

int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

while (!queue.isEmpty()) {

int[] cell = queue.poll();

for (int[] dir : dirs) {

int r = cell[0] + dir[0];

int c = cell[1] + dir[1];

if (r < 0 || r >= m || c < 0 || c >= n ||

matrix[r][c] <= matrix[cell[0]][cell[1]] + 1) continue;

queue.add(new int[] {r, c});

matrix[r][c] = matrix[cell[0]][cell[1]] + 1;

}

}

return matrix;

}

**S2. 更加酷炫的方法，只需要遍历两遍**

这个题有个重点是，4个方向都需要检查，但我们如果是向右，向下走的话，每次有效的数据只是已经走过的部分。

所以在第一遍遍历时，只能利用左侧和上侧的数据；

在第二遍遍历的时候，反方向走，只利用右侧和下侧的数据。time O(mn), space O(1)

public int[][] updateMatrix(int[][] matrix) {

int row = matrix.length, col = matrix[0].length;

for (int i = 0; i < row; i++) {

for (int j = 0; j < col; j++) {

if (matrix[i][j] == 1) {

matrix[i][j] = Integer.MAX\_VALUE;

if (i - 1 >= 0 && matrix[i - 1][j] != Integer.MAX\_VALUE)

matrix[i][j] = Math.min(matrix[i][j], 1 + matrix[i - 1][j]);

if (j - 1 >= 0 && matrix[i][j - 1] != Integer.MAX\_VALUE)

matrix[i][j] = Math.min(matrix[i][j], 1 + matrix[i][j - 1]);

}

}

}

for (int i = row - 1; i >= 0; i--) {

for (int j = col - 1; j >= 0; j--) {

if (i + 1 < row && matrix[i + 1][j] != Integer.MAX\_VALUE)

matrix[i][j] = Math.min(matrix[i][j], 1 + matrix[i + 1][j]);

if (j + 1 < col && matrix[i][j + 1] != Integer.MAX\_VALUE)

matrix[i][j] = Math.min(matrix[i][j], 1 + matrix[i][j + 1]);

}

}

return matrix;

}

286. Walls and Gates

You are given a m x n 2D grid initialized with these three possible values.

-1 - A wall or an obstacle.

0 - A gate.

INF - Infinity means an empty room. We use the value 2 31 - 1 = 2147483647 to represent INF as you may assume that the distance to a gate is less than 2147483647.

Fill each empty room with the distance to its nearest gate. If it is impossible to reach a gate, it should be filled with INF.

**S1 BFS**

找到所有为gate的点一层层向外延展，每层distance加一并更新。

public void wallsAndGates(int[][] rooms) {

if (rooms == null || rooms.length == 0) return;

int m = rooms.length, n = rooms[0].length;

Queue<int[]> queue = new LinkedList<>(); *// store position (x, y)*

for (int i = 0;i < m;i++) {

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

if (rooms[i][j] == 0) queue.offer(new int[]{i, j});

}

}

int[][] bias = new int[][]{{0, -1}, {0, 1}, {1, 0}, {-1, 0}};

int distance = 1;

while (!queue.isEmpty()) {

int size = queue.size();

for (int i = 0;i < size;i++) {

int[] cur = queue.poll();

for (int[] b:bias) {

int x = cur[0] + b[0], y = cur[1] + b[1];

if (x < 0 || x >= m || y < 0 || y >= n) continue;

if (rooms[x][y] > distance) {

rooms[x][y] = distance;

queue.offer(new int[]{x, y});

}

}

}

distance++;

}

}

127,126. Word Ladder I, II

Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord, such that:

* Only one letter can be changed at a time.
* Each transformed word must exist in the word list. Note that beginWord is not a transformed word.

For example, Given: beginWord = "hit" endWord = "cog"

wordList = ["hot","dot","dog","lot","log","cog"]

As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

I: 返回的是word ladder的长度，return its length 5.

II 的区别就是需要return的不是长度，而是从beginWord到endWord（包含两者）的最短路径。

[ ["hit","hot","dot","dog","cog"],

["hit","hot","lot","log","cog"]]

**S1. BFS**

用set记录已经走过的words，保证不会重走；又因为是BFS，所以一定是最短路径。

Queue记录接下来的words, every time poll a word, and then insert all words that only 1 char different with it

public int ladderLength(String beginWord, String endWord, List<String> wordList) {

if (beginWord.length() != endWord.length() || !wordList.contains(endWord)) return 0;

if (beginWord.equals(endWord)) return 1;

Set<String> wordSet = new HashSet<>(wordList);

Queue<String> queue = new LinkedList<>();*// words to check next*

Set<String> visited = new HashSet<>(); *// make sure not go back*

int length = 1;

queue.offer(beginWord);

visited.add(beginWord);

while (!queue.isEmpty()) {

length++;

int size = queue.size(); *// those words only 1 char different*

for (int i = 0;i < size;i++) {

char[] word = queue.poll().toCharArray();

for (int j = 0;j < word.length;j++) { *// change each char*

char origin = word[j];

for (char c = 'a';c <= 'z';c++) {

if (c == origin) continue;

word[j] = c;

String newWord = new String(word);

if (!wordSet.contains(newWord)) continue;

if (newWord.equals(endWord)) return length;

if (!visited.contains(newWord)) {

queue.offer(newWord);

visited.add(newWord);

}

}

word[j] = origin;

}

}

}

return 0;

}

**S2 Word Ladder II,**

需要有两步完成，第一步和之前一样，用BFS, 记录到达每一个word的shortest path, 用map来存，

map1, <word， # of steps needed from start to this word>

map2, <word, List of words that this word could get from/从哪些单词可以到达这个>

第二步用DFS，从endWord往前走，直到beginWord, 每次从map1中取出# of steps, m, then check word list in map2, every word that # of steps are m-1.

DFS时要注意一点：

public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {

List<List<String>> res = new ArrayList<>();

if (beginWord == null || endWord == null || !wordList.contains(endWord)) return res;

Set<String> wordSet = new HashSet<>(wordList);

wordSet.add(beginWord);

Map<String, List<String>> map = new HashMap<>();

Map<String, Integer> distance = new HashMap<>();

Queue<String> queue = new LinkedList<>();

queue.offer(beginWord);

distance.put(beginWord, 0);

bfs(map, distance, queue, wordSet);

dfs(res, new ArrayList<>(), endWord, beginWord, distance, map);

return res;

}

void bfs(Map<String, List<String>> map, Map<String, Integer> distance,

Queue<String> queue, Set<String> wordSet) {

while (!queue.isEmpty()) {

int size = queue.size();

for (int i = 0;i < size;i++) {

String curWord = queue.poll();

char[] chars = curWord.toCharArray();

for (int j = 0;j < chars.length;j++) {

char save = chars[j];

for (char c = 'a';c <= 'z';c++) {

if (c == save) continue;

chars[j] = c;

String newWord = new String(chars);

if (!wordSet.contains(newWord)) continue;

if (!map.containsKey(newWord)) {

map.put(newWord, new ArrayList<>());

}

map.get(newWord).add(curWord);

if (!distance.containsKey(newWord)) {

distance.put(newWord, distance.get(curWord) + 1);

queue.offer(newWord);

}

}

chars[j] = save;

}

}

}

}

void dfs(List<List<String>> res, List<String> list, String cur, String beginWord, Map<String, Integer> distance, Map<String, List<String>> map) {

list.add(0, cur);

if (cur.equals(beginWord)) {

res.add(new ArrayList<String>(list));

} else {

if (map.containsKey(cur)) {

int curDistance = distance.get(cur);

for (String neighbor:map.get(cur)) {

if (distance.containsKey(neighbor) && distance.get(neighbor) + 1 == curDistance) {

dfs(res, list, neighbor, beginWord, distance, map);

}

}

}

}

list.remove(0);

}

582. Kill Process

Given n processes, each process has a unique PID (process id) and its PPID (parent process id).

Each process only has one parent process, but may have one or more children processes. This is just like a tree structure. Only one process has PPID that is 0, which means this process has no parent process. All the PIDs will be distinct positive integers.

We use two list of integers to represent a list of processes, where the first list contains PID for each process and the second list contains the corresponding PPID.

Input:  pid = [1, 3, 10, 5] ppid = [3, 0, 5, 3] kill = 5 Output: [5,10]

Explanation: Kill 5 will also kill 10.

   3

  / \

 1   5

    /

  10

**S1 HashMap + BFS**

先遍历确认<parent, list of kids>关系并放入map + 用queue做BFS，找齐该node的所有kids

public List<Integer> killProcess(List<Integer> pid, List<Integer> ppid, int kill) {

if (pid == null || ppid == null || pid.size() != ppid.size()) return null;

*// 1. store all <ppid, List of pid>, which is <parent, kids> in map:*

Map<Integer, List<Integer>> map = new HashMap<>();

int size = ppid.size();

for (int i = 0;i < size;i++) {

int key = ppid.get(i);

if (!map.containsKey(ppid.get(i))) {

map.put(key, new ArrayList<Integer>());

}

map.get(key).add(pid.get(i));

}

*// 2. handle one number at a time, add to res and push its kids to queue*

*// for further processing.*

Queue<Integer> queue = new LinkedList<>();

List<Integer> res = new ArrayList<>();

queue.offer(kill);

while (!queue.isEmpty()) {

int key = queue.poll();

if (map.containsKey(key)) {

for (int num:map.get(key)) {

queue.offer(num);

}

}

res.add(key);

}

return res;

}

**2D 矩阵DFS**

200. Number of Islands

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

 Answer: 3

**S1. DFS**

非常经典的DFS做法，每次遇到1就找所有相邻点，标为0.

public int numIslands(char[][] grid) {

if (grid == null || grid.length == 0) return 0;

int m = grid.length, n = grid[0].length;

int count = 0;

for (int i = 0;i < m;i++) {

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

if (grid[i][j] == '1') {

count++;

dfs(grid, i, j); *// set all '1' connected with this one to '0'*

}

}

}

return count;

}

private void dfs(char[][] grid, int x, int y) {

if (x < 0 || y < 0 || x >= grid.length || y >= grid[0].length ||

grid[x][y] != '1') return;

grid[x][y] = '0';

dfs(grid, x - 1, y);

dfs(grid, x + 1, y);

dfs(grid, x, y - 1);

dfs(grid, x, y + 1);

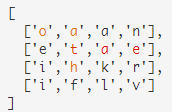
}

212. Word Search II

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

For example, Given words = ["oath","pea","eat","rain"] and board =

 Return ["eat","oath"].

**S1.tire + backtrack**

关于单词的信息，通过建trie存在trie里，然后通过backtrack来找存在的单词。

class TrieNode {

Character val;

boolean isWord;

TrieNode[] child;

public TrieNode(Character c) {

val = c;

child = new TrieNode[26];

isWord = false;

}

}

public class Solution {

private void addWord(String word, TrieNode cur) {

for (char c:word.toCharArray()) {

int idx = c - 'a';

if (cur.child[idx] == null) {

cur.child[idx] = new TrieNode(c);

}

cur = cur.child[idx];

}

cur.isWord = true;

}

static int[][] dirs = new int[][]{{0,1},{0,-1},{1,0},{-1,0}};

private void searchWords(char[][] board, int i, int j, TrieNode root,

List<String> res, StringBuilder sb) {

*// check leaf condition*

if (root.isWord && !res.contains(sb.toString())) {

res.add(sb.toString());

}

if (i < 0 || i >= board.length || j < 0 || j >= board[0].length ||

root == null || board[i][j] == '#') return;

char c = board[i][j];

int idx = board[i][j] - 'a';

if (root.child == null || root.child[idx] == null) return;

*// set it to '#' means this one is being processed right now.*

board[i][j] = '#';

int len = sb.length();

for (int[] dir:dirs) {

searchWords(board, i + dir[0], j + dir[1], root.child[idx], res, sb.append(c));

sb.setLength(len);

}

*// set it back*

board[i][j] = c;

}

public List<String> findWords(char[][] board, String[] words) {

List<String> res = new ArrayList<>();

if (board == null || words == null ||

board.length == 0 || words.length == 0) return res;

TrieNode root = new TrieNode(null);

for (String word:words) {

addWord(word, root);

}

int m = board.length, n = board[0].length;

for (int i = 0;i < m;i++) {

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

searchWords(board, i, j, root, res, new StringBuilder());

}

}

return res;

}

79. Word Search

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example, Given board =

[ ['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']]

word = "ABCCED", -> returns true,

**S1 DFS**

backtrack的第一种情况，求是否有解。

从每个点尝试往下走，正在走，或者路上已经走过的点标为’#’ 避免重复。

走完之后再替换回来，以免影响下一个。

public boolean exist(char[][] board, String word) {

if (board == null || board.length == 0) return false;

int m = board.length, n = board[0].length;

for (int i = 0;i < m;i++) {

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

if (dfs(board, i, j, word, 0)) return true;

}

}

return false;

}

boolean dfs(char[][] board, int x, int y, String word, int idx) {

if (x < 0 || y < 0 || x >= board.length || y >= board[0].length ||

board[x][y] != word.charAt(idx)) return false;

if (idx == word.length() - 1) return true;

int[] vert = {0, 0, -1, 1};

int[] hori = {-1, 1, 0, 0};

board[x][y] = '#';

for (int i = 0;i < vert.length;i++) {

if (dfs(board, x + hori[i], y + vert[i], word, idx + 1)) return true;

}

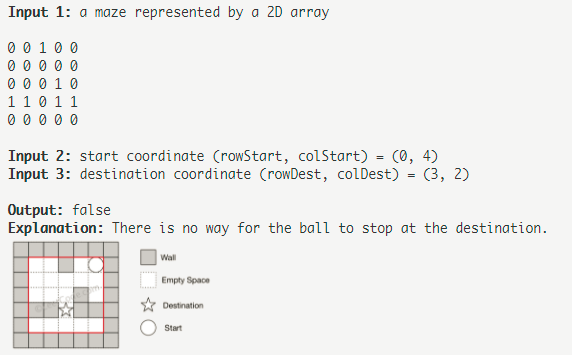
board[x][y] = word.charAt(idx);

return false;

}

490. The Maze

给一个二维矩阵，0代表可以走到的地方，1代表障碍物，并且给出起始地点和终点，看小球能不能从起始点滚到终点。上下左右四个方向都可以。



**S1. BFS**

从起始点开始，尝试向各个方向滚动，滚动停止的条件是到达边界 OR maze[i][j] = 1.

并且要注意，需要用visited来记录已经走过的点，避免进入无限循环。

public boolean hasPath(int[][] maze, int[] start, int[] destination) {

if (start[0] == destination[0] && start[1] == destination[1]) return true;

int m = maze.length, n = maze[0].length;

Queue<int[]> queue = new LinkedList<int[]>();

queue.offer(start);

boolean[][] visited = new boolean[m][n];

visited[start[0]][start[1]] = true;

int[][] dir = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};

while (!queue.isEmpty()) {

int[] cur = queue.poll();

for (int k = 0; k < dir.length; k++) {

int x = cur[0];

int y = cur[1];

*// the ball is rolling.*

while (x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == 0) {

x += dir[k][0];

y += dir[k][1];

}

x -= dir[k][0];

y -= dir[k][1];

if (visited[x][y]) continue;

visited[x][y] = true;

if (x == destination[0] && y == destination[1]) return true;

queue.offer(new int[] {x, y});

}

}

return false;

}

505. The Maze II

和上一道的区别是：这里找的是最短路径的长度。

**S1. BFS**

在BFS的过程中，要记录的不仅是能不能到达某一点，还要记录从起点到这一点走过的路径长度。

并且可以用dp数组做记录，并且用来剪枝。

class Solution {

public static final int[][] dirs = new int[][] {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

public int shortestDistance(int[][] maze, int[] start, int[] destination) {

if(maze == null || maze.length == 0 || maze[0].length == 0) {

return -1;

}

int m = maze.length, n = maze[0].length;

int[][] dp = new int[m][n]; *// distance from start to (i, j)*

Queue<Pair> queue = new LinkedList<>();

queue.offer(new Pair(start[0], start[1], 0));

for(int i = 0; i < m; i++) {

Arrays.fill(dp[i], Integer.MAX\_VALUE);

}

while(!queue.isEmpty()) {

Pair cur = queue.poll();

for(int[] dir : dirs) {

int nextX = cur.x;

int nextY = cur.y;

int len = cur.len;

*// the ball moving till hit the wall/obstacle*

while(nextX < m && nextX >= 0 && nextY < n && nextY >= 0 && maze[nextX][nextY] == 0) {

nextX += dir[0];

nextY += dir[1];

len++;

}

nextX -= dir[0];

nextY -= dir[1];

len--;

*// avoid going through unneccessary cases.*

if(len > dp[destination[0]][destination[1]]) {

continue;

}

if(len < dp[nextX][nextY]) {

dp[nextX][nextY] = len;

queue.offer(new Pair(nextX, nextY, len));

}

}

}

return dp[destination[0]][destination[1]] == Integer.MAX\_VALUE ?

-1 : dp[destination[0]][destination[1]];

}

}

class Pair {

int x;

int y;

int len;

public Pair(int x, int y, int len) {

this.x = x;

this.y = y;

this.len = len;

}

}

与其他分类相关

Houzz - Check whether a given graph is Bipartite or not

<http://www.geeksforgeeks.org/bipartite-graph/>

本质上等于：check if it is possible to paint the graph in two colors, and any adjacent node don’t have the same color.

为什么可以这样看：一种颜色其实就代表one side, 能不能划分成2 sides, 也就是和划分颜色一样。

**S1. BFS**

给root染红色，相邻的染蓝色.....类推，如果走着走着发现有相邻的node有同样的颜色，则not bipartite.