Yahoo SDE2 VO

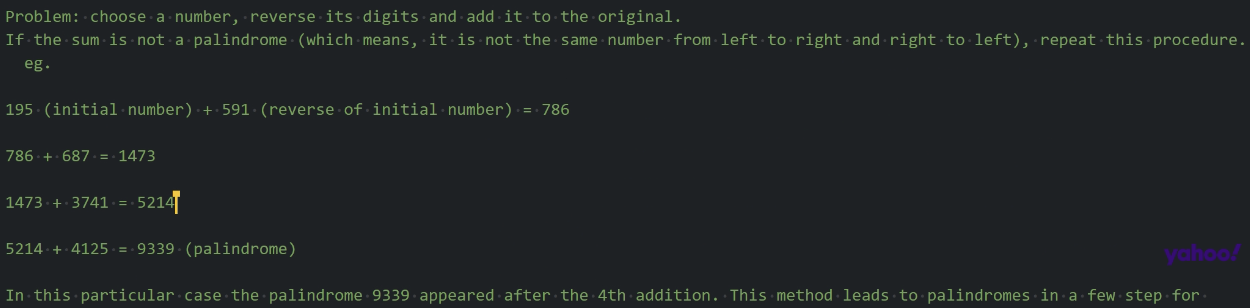
Discord: <https://discord.gg/kXDPs3mJrU>

感谢各位能来助攻，一起加油上岸！请尽量先贴代码

# Round 1 8 am - 8:50 am PT

BQ：问简历，good software engineer(general)

Code



你自己就假装写几个test case

反正他也不说话

\*\*\*忘了考虑一点，写个todo long 因为你可能int 之内怎么加都没有palindrome number，可以写个todo long多一层思考。 然后要有个判断， if num >= Integer.MAX\_VALUE,那就是没有palindrome 对对对 你小子，反应快啊

45行写错了吧，while的判断条件。一开始的number不是回文，你的while进不去的。我下面写的是直接while(true)，当发现是回文时break掉。如果是!isPalindrome()的话，那个if{break;}就不用了。随便用哪种方式都行，但是得统一一下，然后自己拿例子稍微走一下试试看还有没有其他问题 okok

少了一步赋值，先别run！sum算完之后要赋值给number吧，否则你在不停地算number。加一步number = sum；45行的叹号没加上

可能是sum = num + reverseNum那里需要判断一下。sum得定义成long?否则加出界的话是不是变负数了。。？我记不太清了 Integer.MAX\_VALUE + 1变负数了吧

TQL TQL

**!isPalindrome 感叹号**

你的while 条件换一下，你看一眼下面，while(number < Integer.*MAX\_VALUE*)

可以抽出来写个函数，反转input num

然后另一个函数判断某个数是否是回文数，最后找到回文再弄成String。keep一个count用来记录步数

private int reverseNumber(int num){

int reversed = 0;

while(num !=0){

int digit = num%10;

reversed = reversed \* 10 + digit;

num/=10;

}

return reversed;

}

private boolean isPalin (int num) {

if(num < 0) return false;

if(num == 0) return true;

int reverse = reverseNumber(num);

return reverse == num;

}

这么写 拆成两个

//reverse 然后加起来，然后用下面的判断是否palindrome，不palindrome，repeat 对的我也觉得reverse单独拉出来像我这样写

建议你写两个单独的函数，因为这个是要不停循环直到变成回文。

Int sum = 0;//这个sum应该是要定义在while外面，跳出while之后需要把它转成String

Int count = 0;

while(true){

Int reverseNum = reverseNumber(num); //两个单独的函数

sum = num + reverseNum;

Count++;

if(isPalindrome(sum)){

break;

}

}

String res = String.valueOf(sum);

最后再拼他那个要求的格式行了

StringBuilder sb = new StringBuilder();

sb.append(count);

sb.append(res);

Return sb.toString();  
上面老哥写了个corner case的check

LC9，判断是否是回文数：

public boolean isPalindrome(int x) {

if(x < 0) return false;

if(x == 0) return true;

int reverse = 0;

int n = x;

while(n > 0){

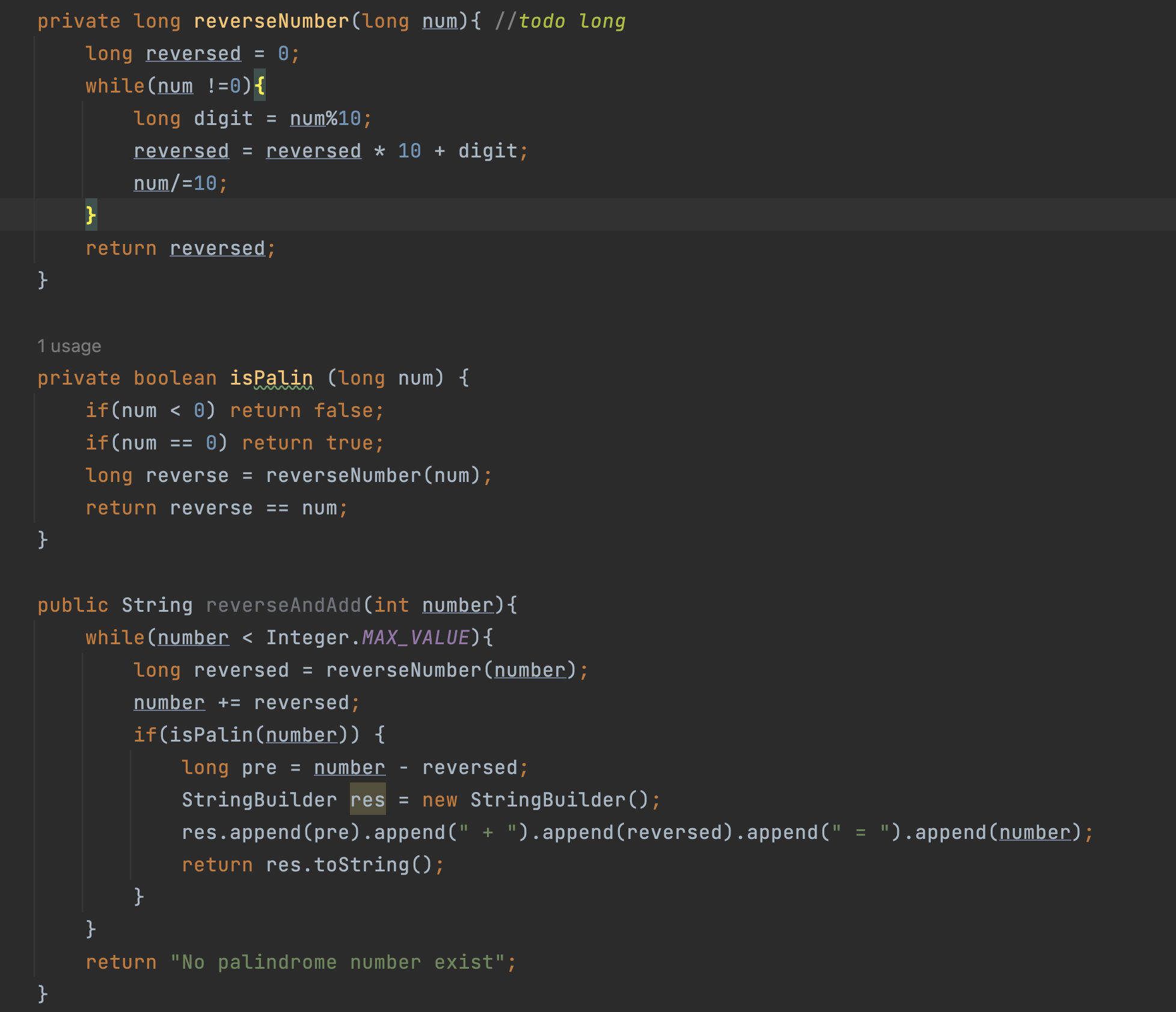
reverse = reverse \* 10 + (n % 10);

n /= 10;

}

return reverse == x;

}

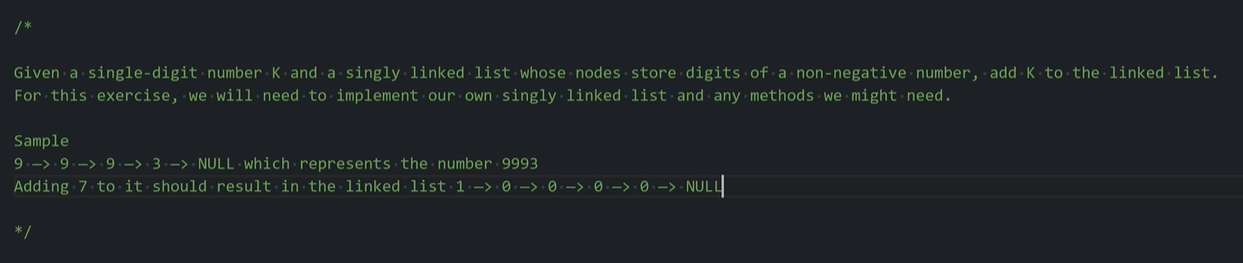


# Round 2 11 am - 11:50 am PT

BQ

Code

LC 2 ? 应该是445



原题应该是，稍等我找找，不是LC2，是另一个。需要先反转一下，对445。先反转list，然后就是LC2了。

<https://leetcode.com/problems/add-two-numbers-ii/>

ListNode list4 = new ListNode(3,null);

ListNode list3 = new ListNode(9,list4);

ListNode list2 = new ListNode(9,list3);

ListNode list1 = new ListNode(9,list2);

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

if (l1 == null) return l2;

if (l2 == null) return l1;

l1 = reverse(l1);

l2 = reverse(l2);

ListNode dummy = new ListNode(0);

ListNode cur = dummy;

int sum = 0;

while (l1 != null || l2 != null || sum != 0) {

if (l1 != null) {

sum += l1.val;

l1 = l1.next;

}

if (l2 != null) {

sum += l2.val;

l2 = l2.next;

}

ListNode node = new ListNode(sum % 10);

sum /= 10;

cur.next = node;

cur = cur.next;

// node.next = dummy.next;

// dummy.next = node;

}

return reverse(dummy.next);

}

private ListNode reverse(ListNode head) {

ListNode prev = null;

ListNode cur = head;

while (cur != null) {

ListNode next = cur.next;

cur.next = prev;

prev = cur;

cur = next;

}

return prev;

}

}

中文力扣题解

本题的主要难点在于链表中数位的顺序与我们做加法的顺序是相反的，为了逆序处理所有数位，我们可以使用栈：把所有数字压入栈中，再依次取出相加。计算过程中需要注意进位的情况。

Time O(max(m,n)) Space O(m + n)

class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

Deque<Integer> stack1 = new ArrayDeque<Integer>();

Deque<Integer> stack2 = new ArrayDeque<Integer>();

while (l1 != null) {

stack1.push(l1.val);

l1 = l1.next;

}

while (l2 != null) {

stack2.push(l2.val);

l2 = l2.next;

}

int carry = 0;

ListNode ans = null;

while (!stack1.isEmpty() || !stack2.isEmpty() || carry != 0) {

int a = stack1.isEmpty() ? 0 : stack1.pop();

int b = stack2.isEmpty() ? 0 : stack2.pop();

int cur = a + b + carry;

carry = cur / 10;

cur %= 10;

ListNode curnode = new ListNode(cur);

curnode.next = ans;

ans = curnode;

}

return ans;

}

}

ListNode list4 = new ListNode(3,null);

ListNode list3 = new ListNode(9,list4);

ListNode list2 = new ListNode(9,list3);

ListNode list1 = new ListNode(9,list2);

int sum = l1.val + k;

l1 = l1.next;

while (l1 != null || sum != 0) {

ListNode node = new ListNode(sum % 10);

sum /= 10;

cur.next = node;

cur = cur.next;

if (l1 != null) {

sum += l1.val;

l1 = l1.next;

}

}

# Round 3 12 pm - 12:50 pm PT

BQ：database consistency model, message queue

singleton?

intp[]res = solution.topK…(int[]nums,int k)

system.out.println(res);

bucket.add(new ArrrayList<>());

bucket.get(j).add(key)

放括号里

Code top K

LC347

qiu qiu

public int[] topKFrequent(int[] nums, int k) {

int n = nums.length;

int[]res = new int[k];

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

for(int i:nums){

map.put(i,map.getOrDefault(i,0)+1);

}

List<Integer>[]bucket = new List[n+1];

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

bucket[i]= new ArrayList<>();

}

for(int key:map.keySet()){

int j = map.get(key);

bucket[j].add(key);

}

int index = 0;

for(int i =n; i >= 0; i--){

if(bucket[i].size()>0){

for(int j:bucket[i]){

res[index++]=j;

if(res.length == k){

return res;

}

}

}

}

return res;

}

}

力扣解法

Space O(n) Time O(nlogk)

class Solution {

public List<Integer> topKFrequent(int[] nums, int k) {

// 使用字典，统计每个元素出现的次数，元素为键，元素出现的次数为值

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

for(int num : nums){

if (map.containsKey(num)) {

map.put(num, map.get(num) + 1);

} else {

map.put(num, 1);

}

}

// 遍历map，用最小堆保存频率最大的k个元素

PriorityQueue<Integer> pq = new PriorityQueue<>(new Comparator<Integer>() {

@Override

public int compare(Integer a, Integer b) {

return map.get(a) - map.get(b);

}

});

for (Integer key : map.keySet()) {

if (pq.size() < k) {

pq.add(key);

} else if (map.get(key) > map.get(pq.peek())) {

pq.remove();

pq.add(key);

}

}

// 取出最小堆中的元素

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

while (!pq.isEmpty()) {

res.add(pq.remove());

}

return res;

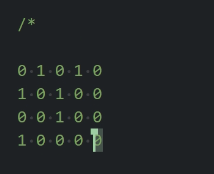
}

}

# Round 4 11 am - 12:0 pm PT 9/23

BQ

Code



Shortest path from top left to bottom right. All directions are accepted including diagonal.

1 is an obstacle and 0 is reachable.

LEETCODE 1091

class Solution {

private static final int[][] directions =

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

public int shortestPathBinaryMatrix(int[][] grid) {

// Firstly, we need to check that the start and target cells are open.

if (grid[0][0] != 0 || grid[grid.length - 1][grid[0].length - 1] != 0) {

return -1;

} //corner case 你刚刚的问题就要在这判断一下

// Set up the BFS.

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

grid[0][0] = 1;

queue.add(new int[]{0, 0});

// Carry out the BFS

while (!queue.isEmpty()) {

int[] cell = queue.remove();

int row = cell[0];

int col = cell[1];

int distance = grid[row][col];

if (row == grid.length - 1 && col == grid[0].length - 1) {

return distance;

}

for (int[] neighbour : getNeighbours(row, col, grid)) {

int neighbourRow = neighbour[0];

int neighbourCol = neighbour[1];

queue.add(new int[]{neighbourRow, neighbourCol});

grid[neighbourRow][neighbourCol] = distance + 1;

}

}

// The target was unreachable.

return -1;

}

private List<int[]> getNeighbours(int row, int col, int[][] grid) {

List<int[]> neighbours = new ArrayList<>();

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

int newRow = row + directions[i][0];

int newCol = col + directions[i][1];

if (newRow < 0 || newCol < 0 || newRow >= grid.length

|| newCol >= grid[0].length

|| grid[newRow][newCol] != 0) {

continue;

}

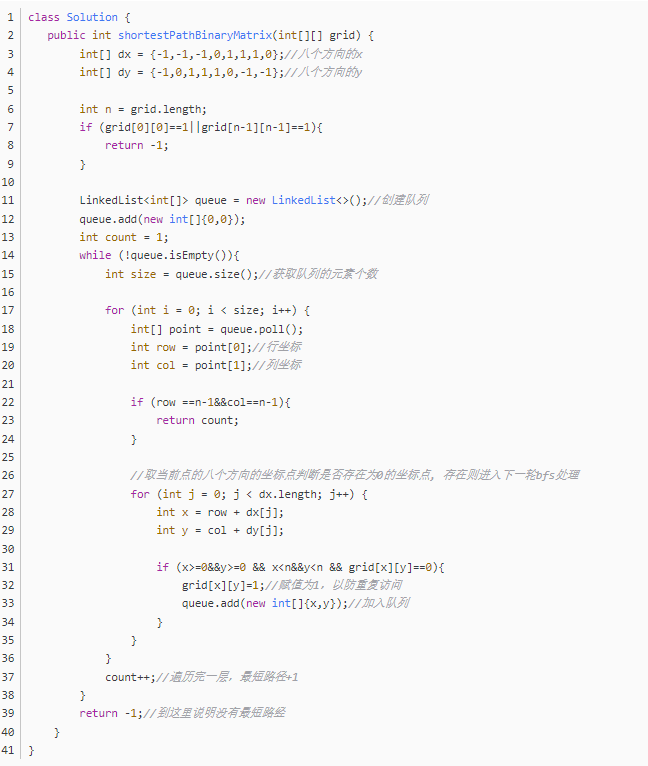
neighbours.add(new int[]{newRow, newCol});

}

return neighbours;

}

}



不是m,是dx.length

public class shortestPathBinaryMatrix {

private static int[][] directions = {{0,1}, {0, -1}, {1, -1}, {1, 0}, {1, 1}, {-1, -1}, {-1, 0}, {-1, 1}};

private int row, col;

public int shortestPathBinaryMatrix(int[][] grid) {

row = grid.length;

col = grid[0].length;

if(grid[0][0] == 1 || grid[row - 1][col - 1] == 1) return -1;

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

grid[0][0] = 1; // 直接用grid[i][j]记录从起点到这个点的最短路径长。按照题意 起点也有长度1

pos.add(new int[]{0,0});

while(!pos.isEmpty() && grid[row - 1][col - 1] == 0){ // 求最短路径 使用BFS

int[] xy = pos.remove();

int preLength = grid[xy[0]][xy[1]]; // 当前点的路径长度

for(int i = 0; i < 8; i++){

int newX = xy[0] + directions[i][0];

int newY = xy[1] + directions[i][1];

if(inGrid(newX, newY) && grid[newX][newY] == 0){

pos.add(new int[]{newX, newY});

grid[newX][newY] = preLength + 1; // 下一个点的路径长度要+1

}

}

}

return grid[row - 1][col - 1] == 0 ? -1 : grid[row - 1][col - 1]; // 如果最后终点的值还是0，说明没有到达

}

private boolean inGrid(int x, int y){

return x >= 0 && x < row && y >= 0 && y < col;

}

}

????

???

Http 最新的3.0 已经改用udp作为http的底层协议了