# <https://qiuzhihui.gitbooks.io/r-book/content/dijkstras_algorithm.html>

<https://lefttree.gitbooks.io/google_questions/content/snapchat/amicable_number.html>

<http://qkxue.net/info/101153/Snapchat-Amicable-Pair>

# Maze solve

import java.util.\*;

public class Path{

public void findPathInMaze(int[][] maze) {

boolean[][] visited = new boolean[maze.length][maze[0].length];

//key: cur position; val: last postion

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

pathMap.put(0, -1);

if(!dfs(maze, 0, 0, visited, pathMap)) {

System.out.println("Defeat");

}

return;

}

public boolean dfs(int[][] maze, int x, int y, boolean[][] visited, HashMap<Integer, Integer> pathMap) {

if(maze[x][y] == 9) {

System.out.println("Victory");

printPathMap(pathMap, x, y, maze[0].length);

return true;

}

boolean ret = false;

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

visited[x][y] = true;

for(int[] dir : dirs) {

int newX = x + dir[0], newY = y + dir[1];

if(newX < 0 || newY < 0 || newX >= maze.length || newY >= maze[0].length || maze[newX][newY] == 5 || visited[newX][newY]) {

continue;

}

pathMap.put(newX \* maze[0].length + newY, x \* maze[0].length + y);

ret |= dfs(maze, newX, newY, visited, pathMap);

pathMap.remove(newX \* maze[0].length + newY);

}

visited[x][y] = false;

return ret;

}

public void printPathMap(HashMap<Integer, Integer> pathMap, int x, int y, int len) {

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

int pos = x \* len + y;

while(pathMap.containsKey(pos) && pathMap.get(pos) != -1) {

path.add(0, new int[]{x, y});

int curPos = pathMap.get(pos);

x = curPos / len;

y = curPos % len;

pos = curPos;

}

path.add(0, new int[]{x, y});

for(int[] p : path) {

System.out.println("<" + p[0] + "," + p[1] + ">");

}

}

public static void main(String[] args) {

Path path = new Path();

int[][] maze = {

{1,5,5,5,5},

{0,5,0,0,0},

{0,5,0,5,0},

{0,0,0,5,0},

{0,5,5,5,9}

};

path.findPathInMaze(maze);

}

}

# 两个String表示的小数相加。 Example: "1.23" + "3.5" = "4.73"

private static String addFloat(String first, String second){

String[] firstArr = first.split("\\.");

String[] secondArr = second.split("\\.");

String integerPart = "", floatPart = "";

StringBuilder sb = new StringBuilder();

//calculate float part

int i, j, v1, v2, sum;

i = Math.max(firstArr[1].length()-1, secondArr[1].length()-1);

boolean carry = false;

while(i >= 0){

v1 = i>=firstArr[1].length()?0:(firstArr[1].charAt(i)-'0');

v2 = i>=secondArr[1].length()?0:(secondArr[1].charAt(i)-'0');

sum = v1+v2+(carry?1:0);

if(sum > 9){

carry = true;

sum -= 10;

}

else

carry = false;

sb.append(sum);

i--;

}

floatPart = sb.reverse().toString();

i = firstArr[0].length()-1;j = secondArr[0].length()-1;

sb = new StringBuilder();

while(i >= 0 || j >= 0){

v1 = i >= 0? (firstArr[0].charAt(i)-'0'):0;

v2 = j >= 0? (secondArr[0].charAt(j)-'0'):0;

sum = v1+v2+(carry?1:0);

if(sum > 9){

carry = true;

sum -= 10;

}

else

carry = false;

sb.append(sum);

i--;

j--;

}

if(carry)

sb.append(1);

integerPart = sb.reverse().toString();

return integerPart+"."+floatPart;

}

# Triangle Minimum Path Sum, LC120. Triangle

Use DP to find the min cost,

**public** **static** **int** findMinCost(List<List<Integer>> list){

**int**[] dp = **new** **int**[list.size()+1];

**for**(**int** i = list.size()-1;i>= 0;i--)

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

dp[j] = Math.*min*(dp[j], dp[j+1])+list.get(i).get(j);

**return** dp[0];

}

Follow up: to find all min cost path. Need DFS

**public** **static** **void** dfs(List<List<Integer>> list, **int** index, **int** lvl, **int** target, String path, List<String> res){

**if**(lvl == list.size()-1){

**if**(target == list.get(lvl).get(index))

res.add(path+" "+target);

**return**;

}

*dfs*(list, index, lvl+1, target - list.get(lvl).get(index), path+" "+list.get(lvl).get(index), res);

*dfs*(list,index+1, lvl+1, target - list.get(lvl).get(index), path+" "+list.get(lvl).get(index), res);

}

# Matrix Minumum Path Sum, LC64. Minimum Path Sum

# Meeting room to find the room schedule:

**public** **static** **void** printSchedule(Interval[] schedule){

PriorityQueue<LinkedList<Interval>> pq = **new** PriorityQueue<>(**new** Comparator<LinkedList<Interval>>(){

**public** **int** compare(LinkedList<Interval> a, LinkedList<Interval> b){

**if**(a.getLast().end == b.getLast().end)

**return** a.getLast().start - b.getLast().start;

**return** a.getLast().end - b.getLast().end; }

});

Arrays.*sort*(schedule, **new** Comparator<Interval>(){

**public** **int** compare(Interval a, Interval b){

**if**(a.start == b.start)

**return** a.end - b.end;

**return** a.start - b.start;

}

});

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

LinkedList<Interval> room = (pq.isEmpty() || schedule[i].start < pq.peek().getLast().end)?**new** LinkedList<>():pq.poll();

room.add(schedule[i]);

pq.offer(room);

}

**int** i = 1;

**while**(!pq.isEmpty()){

LinkedList<Interval> room = pq.poll();

System.***out***.print("Room "+i+":");

**for**(Interval k: room){

System.***out***.print("["+k.start+","+k.end+"] ");

}

System.***out***.println();

i++;

}

}

# word ladder I

# word ladder II

# Big Integer Add and BigInt multiply

**public** String intAdd(String str1, String str2){

StringBuilder sb = **new** StringBuilder();

**boolean** carry = **false**;

**int** len1 = str1.length(), len2 = str2.length();

**for**(**int** i = len1-1, j = len2-1; i>= 0 || j>= 0;i--, j--){

**int** m = i >= 0?str1.charAt(i)-'0':0;

**int** n = j >= 0?str2.charAt(j)-'0':0;

**int** sum = m+n + (carry?1:0);

**if**(sum > 9){

sum -= 10;

}

**else** carry = **false**;

sb.append(sum);

}

**if**(carry)

sb.append("1");

**return** sb.reverse().toString();

}

**public** String multiply(String str1, String str2){

**int** len1 = str1.length(), len2 = str2.length(), i, j;

**int**[]result = **new** **int**[len1+len2];

**for**(i = len1-1;i >= 0;i--)

**for**(j = len2-1;j>=0;j--){

**int** m = str1.charAt(i)-'0';

**int** n = str2.charAt(j)-'0';

**int** product = m\*n+result[i+j+1];

result[i+j] += product/10;

result[i+j+1] = product%10;

}

StringBuilder sb = **new** StringBuilder();

**for**(i = 0;i < result.length;i++)

**if**(result[i] != 0)

**break**;

**for**(;i < result.length;i++)

sb.append(result[i]);

**return** sb.toString();

}

# find amicable numbers

// [220, 284], [1184, 1210], [2620, 2924], [5020, 5564], [6232, 6368], [10744, 10856], [12285, 14595], [17296, 18416], [66928, 66992]

**public** **static** List<List<Integer>> getAmicableNumber(**int** n){

**int**[] facterNumber = **new** **int**[n+1];

Arrays.*fill*(facterNumber, 1);

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

**for**(**int** i = 2; i <= n/2;i++)

**for**(**int** j = i+i; j <= n;j+=i)

facterNumber[j] += i;

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

**for**(**int** i = 2;i <= n;i++){

**if**(map.containsKey(facterNumber[i]) && map.get(facterNumber[i]) == i)

res.add(Arrays.*asList*(i, facterNumber[i]));

**else**

map.put(i, facterNumber[i]);

}

**return** res;

}

# 第一题是给一个 int 矩阵，从左上角到右下角斜对角打印出来所有数字。lz 一开始理解错了，幸好在写关键部分的 code 的时候自己反应过来了。

**int**[][] matrix = **new** **int**[][]{{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

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

**int** sum = m+n-2, i;

**for**(**int** k = 0;k <= sum;k++){

**for**(i = Math.*max*(0, k-n+1); i < Math.*min*(k+1, m);i++)

System.***out***.print(matrix[i][k-i]+",");

System.***out***.println();

}

|  |  |  |
| --- | --- | --- |
|  | Meeting time, find free time (7am-7pm) | Sort + merge |
| 56 | [Merge Intervals](https://leetcode.com/problems/merge-intervals" \t "_blank) | Sort + merge |
| 146 | [LRU Cache](https://leetcode.com/problems/lru-cache" \t "_blank) | Hashmap + deque   /   hashmap + linkedlist |
| 253 | [Meeting Rooms II](https://leetcode.com/problems/meeting-rooms-ii" \t "_blank) | Sort + heap + greedy / [sweep line](http://www.jiuzhang.com/qa/933/" \t "_blank) |
| 212 | [Word Search II](https://leetcode.com/problems/word-search-ii" \t "_blank) | Trie + DFS |
|  | rate limiter | deque |
| 76 | [Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring" \t "_blank) | Hashmap + two pointer |
|  | 8 \* 8的棋盘给起点和终点，K 步，找一共多少条路径 | DP |
| 300 | [Longest Increasing Subsequence](https://leetcode.com/problems/longest-increasing-subsequence" \t "_blank) | A list keep the LIS, binary search to replace/append |
| 43 | [Multiply Strings](https://leetcode.com/problems/multiply-strings" \t "_blank) | array |
| 230 | [Kth Smallest Element in a BST](https://leetcode.com/problems/kth-smallest-element-in-a-bst" \t "_blank) | Tree |
| 139 | [Word Break](https://leetcode.com/problems/word-break" \t "_blank) | DP |
| 140 | [Word Break II](https://leetcode.com/problems/word-break-ii" \t "_blank) | DP + backtracking |
|  | 求3个数的数组和加起来是target的3个数 | DFS |
| 15 | [3Sum](https://leetcode.com/problems/3sum" \t "_blank) | Two pointers |
|  | N  person & N bike in matrix，配对的距离和最小 | [Hungarian algorithm](https://en.wikipedia.org/wiki/Hungarian_algorithm" \t "_blank).  <http://www.geeksforgeeks.org/hungarian-algorithm-assignment-problem-set-1-introduction/> |
| 151 | [Reverse Words in a String](https://leetcode.com/problems/reverse-words-in-a-string" \t "_blank) | Build-in function / reverse string + reverse each words |
|  | [求inorder traversal DAG的第k个node](http://www.1point3acres.com/bbs/thread-155964-1-1.html" \t "_blank)。此DAG上每个node out degrees最大为2.但是in degree可以大于2。max len = 2^n  exponential | DFS + memorization |
| 337 | [House Robber III](https://leetcode.com/problems/house-robber-iii" \t "_blank) | TreeDP (bottom up) |
| 113 | [Path Sum II](https://leetcode.com/problems/path-sum-ii" \t "_blank) | Tree + level-order |
| 112 | [Path Sum](https://leetcode.com/problems/path-sum" \t "_blank) | Tree + level-order |
| 332 | [Reconstruct Itinerary](https://leetcode.com/problems/reconstruct-itinerary" \t "_blank) | DFS + hashmap |
| 68 | [Text Justification](https://leetcode.com/problems/text-justification" \t "_blank) | String (corner cases: 1. last line, 2. one word a line) |
| 403 | [Frog Jump](https://leetcode.com/problems/frog-jump" \t "_blank) | DFS + memorization |
|  | 2D Frog Jump |  |
| 210 | [Course Schedule II](https://leetcode.com/problems/course-schedule-ii" \t "_blank) | Indegree & successors for each node,tpSort |
|  | [edit distance to palindrome](http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=205673&extra=page%3D1%26filter%3Dsortid%26sortid%3D311%26searchoption%5B3088%5D%5Bvalue%5D%3D1%26searchoption%5B3088%5D%5Btype%5D%3Dradio%26searchoption%5B3089%5D%5Bvalue%5D%5B3%5D%3D3%26searchoption%5B3089%5D%5Btype%5D%3Dcheckbox%26searchoption%5B3090%5D%5Bvalue%5D%3D1%26searchoption%5B3090%5D%5Btype%5D%3Dradio%26searchoption%5B3046%5D%5Bvalue%5D%3D21%26searchoption%5B3046%5D%5Btype%5D%3Dradio%26sortid%3D311" \t "_blank). The cost of add remove and replace are both 1. | http://stackoverflow.com/questions/31894027/how-to-convert-a-string-to-palindrome-with-minimum-number-of-removals-of-charact |
| 72 | [Edit Distance](https://leetcode.com/problems/edit-distance" \t "_blank) | a[i,j] means the edit distance of s[:i], t[:j] |
| 439 | [Ternary Expression Parser](https://leetcode.com/problems/ternary-expression-parser/" \t "_blank) | stack |
|  |  |  |
| 305 | [Number of Islands II](https://leetcode.com/problems/number-of-islands-ii" \t "_blank) | Union and find (practice) |
| 153 | [Find Minimum in Rotated Sorted Array](https://leetcode.com/problems/find-minimum-in-rotated-sorted-array" \t "_blank) | Binary search |
| 39 | [Combination Sum](https://leetcode.com/problems/combination-sum" \t "_blank) | DFS |
| 169 | [Majority Element](https://leetcode.com/problems/majority-element" \t "_blank) | Hashmap / count |
| 101 | [Symmetric Tree](https://leetcode.com/problems/symmetric-tree" \t "_blank) | Tree (recursive / stack) |
| 133 | [Clone Graph](https://leetcode.com/problems/clone-graph" \t "_blank) | DFS |
| 380 | [Insert Delete GetRandom O(1)](https://leetcode.com/problems/insert-delete-getrandom-o1" \t "_blank) | Hashmap |
| 381 | [Insert Delete GetRandom O(1) - Duplicates allowed](https://leetcode.com/problems/insert-delete-getrandom-o1-duplicates-allowed" \t "_blank) | Hashmap |
| 269 | [Alien Dictionary](https://leetcode.com/problems/alien-dictionary" \t "_blank) | Topological sorting.  Note: Initialize degree to -1  use HashMap<Character, HashSet<Charater>> to avoid duplicate  check validation before return final result |
| 51 | [N-Queens](https://leetcode.com/problems/n-queens" \t "_blank) | Backtracking |
| 291 | [Word Pattern II](https://leetcode.com/problems/word-pattern-ii" \t "_blank) | Backtracking + pruning + hash map |
| 290 | [Word Pattern](https://leetcode.com/problems/word-pattern" \t "_blank) | Hash map |
| 98 | [Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree" \t "_blank) | Tree / DFS |
| 317 | [Shortest Distance from All Buildings](https://leetcode.com/problems/shortest-distance-from-all-buildings" \t "_blank) | BFS + pruning |
| 377 | [Combination Sum IV](https://leetcode.com/problems/combination-sum-iv" \t "_blank) | DP |
|  | single thread log | Stack |
| 270 | [Closest Binary Search Tree Value](https://leetcode.com/problems/closest-binary-search-tree-value" \t "_blank) | Tree |
| 272 | [Closest Binary Search Tree Value II](https://leetcode.com/problems/closest-binary-search-tree-value-ii" \t "_blank) | Tree, find closest, compare its succ&pre |
| 10 | [Regular Expression Matching](https://leetcode.com/problems/regular-expression-matching" \t "_blank) | DP |

# LC 149找出在同一条线上最多的点的数量

# LC 329 Longest Increasing Path in a Matrix

# LC 351 题目是Android phone lock pattern.

九个数字的，没有利特版本难。每个数字只能用一次,只能上下左右对角线八个方向移动下一步。 超过四位是有效的。求多少组合

Basic dfs. Need to build a nextValidMove list for each digit since each digit has different nextMove. Also we only need to calculate starting point 1,2,5. Since 1,3,7,9 is symmetric while 2,4,6,8 is also symmetric.

**public** **static** **int** unlockPattern(){

List<List<Integer>> nextValidMove = **new** ArrayList<>();

nextValidMove.add(**null**);

nextValidMove.add(Arrays.*asList*(2,4,5));

nextValidMove.add(Arrays.*asList*(1,3,4,5,6));

nextValidMove.add(Arrays.*asList*(2,5,6));

nextValidMove.add(Arrays.*asList*(1,2,5,8,7));

nextValidMove.add(Arrays.*asList*(1,2,3,4,6,7,8,9));

nextValidMove.add(Arrays.*asList*(3,2,5,8,9));

nextValidMove.add(Arrays.*asList*(4,5,8));

nextValidMove.add(Arrays.*asList*(7,4,5,6,9));

nextValidMove.add(Arrays.*asList*(8,5,6));

**boolean**[] visited = **new** **boolean**[10];

**int** res = 0;

**for**(**int** i = 8; i >= 4;i--){

res += 4\**unlockPatternUtil*(1, i, nextValidMove, visited);

res += 4\**unlockPatternUtil*(2, i, nextValidMove, visited);

res += *unlockPatternUtil*(5, i, nextValidMove, visited);

}

**return** res;

}

**private** **static** **int** unlockPatternUtil(**int** start, **int** leftStep, List<List<Integer>> nextValidMove, **boolean**[] visited){

**if**(leftStep == 0)

**return** 1;

visited[start] = **true**;

**int** res = 0;

**for**(**int** i = 0;i < nextValidMove.get(start).size();i++){

**int** nextMove = nextValidMove.get(start).get(i);

**if**(!visited[nextMove])

res += *unlockPatternUtil*(nextMove, leftStep-1, nextValidMove, visited);

}

visited[start] = **false**;

**return** res;

}

241. Different Ways to Add Parentheses

124 Maximum path in binary tree

# terminal里输入文件名的一部分然后按tab补全，求能找到target文件要输入string的最短长度

Trie+priorityqueue

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

String[] word = {"apple","appb","app","appla","ac","banana","basket","basketball"};

Trie root = **new** Trie();

*buildTrie*(root, word);

String[] test = {"","a", "ap","appl","ba","bas", "basketballplayer","cbda"};

**for**(String t: test){

System.***out***.println("The auto complete for word "+t+" is "+*tab*(root, t));

}

}

**public** **static** **void** buildTrie(Trie root, String[] words){

Trie curr = **null**;

**for**(String w: words){

curr = root;

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

**int** index = w.charAt(i)-'a';

**if**(curr.next[index] == **null**)

curr.next[index] = **new** Trie();

curr = curr.next[index];

curr.q.offer(w);

}

curr.isWord = **true**;

}

}

**public** **static** String tab(Trie root, String t){

**if**(t.isEmpty()) **return** **null**;

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

**int** index = t.charAt(i)-'a';

**if**(root.next[index] == **null**)

**return** **null**;

root = root.next[index];

}

**if**(root.isWord)

**return** t;

**else** **return** root.q.peek();

}

}

**class** Trie{

**public** Trie[] next;

**public** PriorityQueue<String> q;

**public** **boolean** isWord;

**public** Trie(){

next = **new** Trie[26];

q = **new** PriorityQueue<String>(**new** Comparator<String>(){

**public** **int** compare(String a, String b){

**if**(a.length() == b.length())

**return** a.compareTo(b);

**return** a.length()-b.length();

}

});

isWord = **false**;

}

}

# LC 322, different version, doesn’t define the start airport, need to find all possible itineraries. So we need treat each of depart airport as start to find out all the solution.

public static List<String> reconstruct(String[][] ticket){

HashSet<String> set = new HashSet<String>();

HashMap<String, PriorityQueue<String>> map = new HashMap();

for(String[] pair: ticket){

set.add(pair[0]);

if(!map.containsKey(pair[0]))

map.put(pair[0], new PriorityQueue<String>());

map.get(pair[0]).offer(pair[1]);

}

int stops = ticket.length;

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

for(String depart: set){

dfs(depart, map, res);

if(res.size() != stops+1){

res.clear();

map.clear();

for(String[] pair: ticket){

if(!map.containsKey(pair[0]))

map.put(pair[0], new PriorityQueue<String>());

map.get(pair[0]).offer(pair[1]);

}

}

else break;

}

return res;

}

public static void dfs(String depart, HashMap<String, PriorityQueue<String>> map, List<String> res){

while(map.containsKey(depart) && !map.get(depart).isEmpty()){

String next = map.get(depart).poll();

dfs(next, map, res);

}

res.add(0, depart);

}

# XML Parser

LZ的方法是用stack来存parent, 遇到open和text建新node, 并把新node加到stack顶部node的child list里面，如果是open就把新node压栈，遇到close就pop掉stack顶端的node-google 1point3acres  
写完以后小哥的follow up是如果输入的token有问题，比如close tag和open tag不匹配，如何做verification

**public** **static** Tree xmlParse(tokenizer token){

String[] nextToken = token.getNextToken();

**if**(nextToken == **null**) **return** **null**;

Stack<Tree> stack = **new** Stack();

Tree root = **new** Tree(nextToken[0],nextToken[1]);

Tree curr = root;

nextToken = token.getNextToken();

stack.push(root);

**while**(nextToken != **null**){

**switch**(nextToken[1]){

**case** "open":

curr.children.add(**new** Tree(nextToken[0], nextToken[1]));

stack.push(curr);

curr = curr.children.get(curr.children.size()-1);

**break**;

**case** "close":

**if**(!curr.name.equals(nextToken[0]))

**throw** **new** IllegalArgumentException("Invalid XML input!"+nextToken[0]+" "+nextToken[1]);

curr = stack.pop();

**break**;

**case** "text":

curr.children.add(**new** Tree(nextToken[0], nextToken[1]));

**break**;

}

nextToken = token.getNextToken();

}

**if**(!stack.isEmpty())

**throw** **new** IllegalArgumentException("Invalid XML input!");

**else** **return** root;

}

}

# Snapdiff

对f2简历一个map，来记录每个line的

对于f1里面每一个line

如果map里面不包含这个line

打印"-line"

否则 把这条记录的计数减一

如果减完是0 就删除

如果map还有剩的，就打印出来“+line”

**public** **static** **void** printDiff(List<String> f1, List<String> f2){

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

**for**(String line: f2)

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

**for**(String line: f1){

**if**(!map.containsKey(line))

System.***out***.println("-"+line);

**else**{

map.put(line, map.get(line)-1);

**if**(map.get(line) == 0)

map.remove(line);

}

}

**for**(String line: map.keySet())

**for**(**int** i = 0;i < map.get(line);i++)

System.***out***.println("-"+line);

}

# unit test + modular test + sy[stem](http://www.1point3acres.com/%E7%BE%8E%E5%9B%BD%E5%9B%BD%E5%9C%9F%E5%AE%89%E5%85%A8%E9%83%A82012%E5%B9%B4%E5%BA%A6%E6%9B%B4%E6%96%B0stem%E4%B8%93%E4%B8%9A%E5%90%8D%E5%8D%95-%E7%9C%8B%E7%9C%8B%E4%BD%A0%E7%9A%84%E4%B8%93%E4%B8%9A/" \t "_blank) test

## Unit test:

If you want to use JUnit to create the tests, you need to create your testcase class, then create individual test methods that test specific functionality of your class/module under tests (single testcase classes are usually associated with a single "production" class that is being tested) and inside these methods execute various operations and compare the results with what would be correct. It is especially important to try and cover as many corner cases as possible.

 Define the expected and desired output for a normal case, with correct input.

 Now, implement the test by declaring a class, name it anything (Usually something like TestAddingModule), and add the testAdd method to it (i.e. like the one below) :

import static org.junit.Assert.\*;

public class AdditionTests {

@Test

public void testSimpleAddition() { ... }

@Test

public void testPositiveNegativeAddition() { ... }

@Test

public void testNegativePositiveAddition() { ... }

@Test

public void testNegativeAddition() { ... }

@Test

public void testOverflow() { ... }

}

1. Group of Friends (3) - Union - Find : 发现matrix*[j] == 1 就union两人在一`起，最后遍历parent数组找独立的subset个数即可*
2. *Meeting Room III : room带有weight怎么排能出最大的weight ？？？？？？*
3. *Valid Palindrome : two pointers经典*
4. *Edit Distance : LC-72 :*
5. *Copy List with Random Pointer : LC-138*
6. *Remove digits from a number to construct lowest number LC-402*
7. *Valid Sudoku*
8. *Sudoku Solver*
9. *Find Minimum Path between two points in a matrix with walls : 给定起点和终点，找到从起点到终点的最短路径，如果走不通则返回-1。 BFS Follow up :墙可以打破 求最小COST*
10. *Edit distance into Palindrome (对string 和它的 reverse 做Edit distance 左下到右上对角线的值的min 如果是奇数则取对角线和对角线上下一个格的最小值）*
11. *Is prime number*
12. *Count Primes : LC-204*
13. *Pow(a, b) : LC-50*
14. *Sqrt(x) : LC-69*
15. *LC-290*
16. *Number of Islands I (2)*
17. *LRU Cache*
18. *Burst Balloon*
19. *Word Pattern I*
20. *Word Pattern II (2) Backtracking 暴力解 用hashmap维护当前的mapping, hashset判断是否重复*
21. *给一个数组，求index使得该index左侧和右侧sum相等。*
22. *House Robber : LC-198, DP*
23. *Combination Sum IV : DP = Sum(DP[i-nums[j]] for all nums[j] <= i)*
24. *Word Finder : 给一个字典，chat, ever, snapchat, snap, salesperson, per, person, sales, son, whatsoever, what, so 找出所有非复合词， 比如snapchat = snap + chat。 解法：TRIE*
25. *Log Parser (2) : [http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=200906&extra=page%3D1%26filter%3Dsortid%26sortid%3D311%26searchoption%255B3089%255D%255Bvalue%255D%255B2%255D%3D2%26searchoption%255B3089%255D%255Btype%255D%3Dcheckbox%26searchoption%255B3046%255D%255Bvalue%255D%3D21%26searchoption%255B3046%255D%255Btype%255D%3Dradio&page=1](http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=200906&extra=page%3D1%26filter%3Dsortid%26sortid%3D311%26searchoption%255B3089%255D%255Bvalue%255D%255B2%255D%3D2%26searchoption%255B3089%255D%255Btype%255D%3Dcheckbox%26searchoption%255B3046%255D%255Bvalue%255D%3D21%26searchoption%255B3046%255D%255Btype%255D%3Dradio&page=1" \t "_blank)*

*[https://discuss.leetcode.com/top ... arse-cpu-log-file/7](https://discuss.leetcode.com/topic/54105/snapchat-parse-cpu-log-file/7" \t "_blank)*

1. *Course Schedule II (2)*
2. *Implement TreeMap*
3. *IPFilter : 一定要注意字符串到binary string之间的转化，非常容易出BUG*
4. *Delete Kth node from a Linked List from back :  LC -19 Dummy node + two pointers*
5. *Multiply Strings : 遵循乘法的概念可解 开一个m+n长度的array 从后往前乘*
6. *Implement Queue using Stack LC-232 : 维护两个stack, 一个用来input另一个用来Output*
7. *Reverse Words in String : 翻转整个string再翻转每一个单词*
8. *BigInt  + - \* (N)*
9. *Evaluate String*
10. *[http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=197498&extra=page%3D1%26filter%3Dsortid%26sortid%3D311%26searchoption%5B3089%5D%5Bvalue%5D%5B2%5D%3D2%26searchoption%5B3089%5D%5Btype%5D%3Dcheckbox%26searchoption%5B3046%5D%5Bvalue%5D%3D21%26searchoption%5B3046%5D%5Btype%5D%3Dradio%26sortid%3D311](http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=197498&extra=page%3D1%26filter%3Dsortid%26sortid%3D311%26searchoption%5B3089%5D%5Bvalue%5D%5B2%5D%3D2%26searchoption%5B3089%5D%5Btype%5D%3Dcheckbox%26searchoption%5B3046%5D%5Bvalue%5D%3D21%26searchoption%5B3046%5D%5Btype%5D%3Dradio%26sortid%3D311" \t "_blank)*
11. *Repeated DNA Sequence*
12. *Frog jump*
13. *Find Duplicate number (3) HashSet, Sort, Binary Search*
14. *Coins in a Line II*
15. *Construct tree from Inorder & preorder*
16. *Min Stack*
17. *Reverse integer*
18. *Least Important Boss : LCA的变形版本*
19. *Binary Tree Vertical Traversal : LC-314 BFS  2 Queues*
20. *Basic Calculator (3)*
21. *N Queues (3)*
22. *构建BST*
23. *Amicable Number*
24. *ZigZag print (3)*
25. *Substring Anagram (3)*
26. *24点游戏*
27. *[http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=165890&extra=page%3D2%26filter%3Dsortid%26sortid%3D311%26searchoption%5B3089%5D%5Bvalue%5D%5B2%5D%3D2%26searchoption%5B3089%5D%5Btype%5D%3Dcheckbox%26searchoption%5B3046%5D%5Bvalue%5D%3D21%26searchoption%5B3046%5D%5Btype%5D%3Dradio%26sortid%3D311](http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=165890&extra=page%3D2%26filter%3Dsortid%26sortid%3D311%26searchoption%5B3089%5D%5Bvalue%5D%5B2%5D%3D2%26searchoption%5B3089%5D%5Btype%5D%3Dcheckbox%26searchoption%5B3046%5D%5Bvalue%5D%3D21%26searchoption%5B3046%5D%5Btype%5D%3Dradio%26sortid%3D311" \t "_blank) Build Tree from given string input and print it inorderly*
28. *Reverse Linked List*
29. *Excel Column Conversion LC-168 LC-171 注意从数字转到字符串的时候要先-1再%26得到当前位置的字母*
30. *BST ZigZag Traversal LC-103 s*
31. *Symmetric Tree Iterative*
32. *Closest BST Value  LC-270*
33. *Closest BST Value II*

# 单词变复数

**public** **static** String getPluralUtil(String s){

**char** c = s.charAt(s.length()-1);

**if**(c == 'y'){

**if**(s.length() >= 2 && *isVowel*(s.charAt(s.length()-2)))

**return** s+'s';

**else**

**return** s.substring(0, s.length()-1)+"ies";

}

**else** **if**(c == 'o'){

**if**(s.length() >= 2 && *isVowel*(s.charAt(s.length()-2)))

**return** s+'s';

**else**

**return** s+"es";

}

**else** **if**(c == 's' || c == 'x' || c == 'z'){

**return** s+"es";

}

**else** **if**(c == 'f'){

**return** s.substring(0, s.length()-1)+"ves";

}

String sub = s.substring(s.length()-2);

**if**(sub.equals("fe")){

**return** s.substring(0, s.length()-2)+"ves";

}

**else** **if**(sub.equals("ch") || sub.equals("sh")){

**return** s+"es";

}

**else** **return** s+'s';

}

**public** **static** **boolean** isVowel(**char** c){

**return** c == 'a'||c =='e'||c=='i'||c=='o'||c=='u';

}

# Helicopter Path

这道题跟之前做过的Rain Path很相似，区别在于：  
Rain Path是求从src到dst最高高度最小的路径。  
Helicopter Path是求从src到dst高度之和最小的路径。其中路径上下一个点的高度必需不小于前一个点。  
Dijkstra algorithm

class Solution {

public:

int minimumHeightsSum(const pair<int, int>& src, const pair<int, int>& dst, const vector<vector<int>>& matrix) const {

const vector<pair<int, int>> directions{ { 0, 1 }, { 0, -1 }, { 1, 0 }, { -1, 0 } };

int sum = 0;

int n = matrix.size();

if (n == 0) {

return 0;

}

int m = matrix.size();

if (m == 0) {

return 0;

}

unordered\_map<int, int> visited; //key: index, value: cumulated sum

//below two data structue simulate hash heap

unordered\_map<int, MyHeap::iterator> hash; //key: index, value: position of index in heap

MyHeap heap; //key: cumulated sum, value: index

int src\_index = src.first \* n + src.second;

int dst\_index = dst.first \* n + dst.second;

visited[src\_index] = matrix[src.first][src.second];

auto res\_it = heap.emplace(visited[src\_index], src\_index);

hash[src\_index] = res\_it;

while (!heap.empty()) {

auto cur\_it = heap.begin();

int cur\_sum = cur\_it->first;

int cur\_x = cur\_it->second / n;

int cur\_y = cur\_it->second % n;

for (const auto& d : directions) {

int next\_x = cur\_x + d.first;

int next\_y = cur\_y + d.second;

int next\_index = next\_x \* n + next\_y;

if (next\_x >= 0 && next\_x < n && next\_y >= 0 && next\_y < m && matrix[next\_x][next\_y] >= matrix[cur\_x][cur\_y]) {

auto v\_it = visited.find(next\_index);

if (v\_it == visited.end()) {

//if next point has not been visited

int next\_sum = cur\_sum + matrix[next\_x][next\_y];

visited[next\_index] = next\_sum;

auto res\_it = heap.emplace(visited[next\_index], next\_index);

hash[next\_index] = res\_it;

}

else {

int next\_sum = v\_it->second;

if (next\_sum > cur\_sum + matrix[next\_x][next\_y]) {

//if current next\_sum > new next\_sum

auto h\_it = hash.find(next\_index);

if (h\_it != hash.end()) {

//if next point is already in heap

heap.erase(h\_it->second);

}

visited[next\_index] = cur\_sum + matrix[next\_x][next\_y];

auto res\_it = heap.emplace(visited[next\_index], next\_index);

hash[next\_index] = res\_it;

}

}

if (next\_index == dst\_index) {

return visited[dst\_index];

}

}

}

hash.erase(cur\_it->second);

heap.erase(cur\_it);

}

return -1;

}

private:

using MyHeap = multimap<int, int, less<int>>;

};

# Max Value Formula

maxValue()是原题目的DP解法。可以看到状态转移方程很简单明了。  
原题中包含()作为运算符，我一开始纠结了半天怎么在状态中表示并处理()。  
不过其实仔细观察，我们可发现我们并不用处理()：  
假设我们有三个数a1, a2, a3，这三个数可以有下面这些组合：  
1. a1 + a2 + a3        =>        (a1 + a2) + a3  
2. a1 + a2 \* a3                =>        a1 + (a2 \* a3)   
3. (a1 + a2) \* a3  
4. a1 \* a2 \* a3                =>        (a1 \* a2) \* a3  
5. a1 \* (a2 + a3)  
6. a1 \* a2 + a3                =>        (a1 \* a2) + a3  
我们给上述运算都强制加上()后可看到，  
max(dp[0][1] + dp[2][2], dp[0][1] \* dp[2][2]) 处理了1，3，4，6  
max(dp[0][0] + dp[1][2], dp[0][0] \* dp[1][2]) 处理了2，5  
当我们在代码中运行k loop时，dp[j][k]相当于括号内aj到ak的结果，dp[k + 1]相当于括号内ak+1到ai的结果。

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

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

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

dp\_[j][i] = max(dp\_[j][i], max(dp\_[j][k] + dp\_[k + 1][i], dp\_[j][k] \* dp\_[k + 1][i]));

}

}

}

maxValueFormula()是follow up的解法。  
一看到follow up，大体上我们可以想到是recursion + backtracking的题目。可是我很难想出如何写出一段简洁的一边计算最大值，一边保存当前最大值的公式的代码。  
然后我想到既然我们已经在maxValue()中得到了表示最大值的状态转移数列，可不可在follow up中直接backtracking这个数列？  
已知dp[0][n - 1]，并且我们知道dp[0][n - 1]等于dp[0][k] + or \* dp[k + 1][n - 1]，我们可以得到（dp[0][k]） + or \* （dp[k + 1][n - 1]）这个公式。然后我们可以再分治dp[0][k]和dp[k + 1][n - 1]，直到我们得到完整的公式。  
我运用了STL list来构建这个公式，因为我们可以在list的任意位置添加元素。运用了list::iterator来记录子公式应该被添加到的位置。最后返回结果的时候，再把list转换为string。  
通过last\_op我们可以知道子公式是否需要被()包围。  
  
我们用{ 0, -1, 4, -9, 0.5 }做例：  
max\_value = 5.5  
dp[0][4] = dp[0][0] + dp[1][4]  
list: +  
dp[0][0] = 0  
list: 0 +  
dp[1][4] = dp[1][3] + dp[4][4]  
list: 0 + +  
dp[1][3] = dp[1][1] \* dp[2][3]  
list: 0 + \* +  
dp[1][1] = -1  
list: 0 + -1 \* +  
dp[2][3] = dp[2][2] + dp[3][3]  
list: 0 + -1 \* （ + ） +  
dp[2][2] = 4  
list: 0 + -1 \* （4 + ） +  
dp[3][3] = -9  
list: 0 + -1 \* （4 + -9） +  
dp[4][4] = 0.5  
list: 0 + -1 \* （4 + -9） + 0.5  
we got the result!  
  
昨晚家里网络不好，登不上地里。  
  
  
  
**补充内容 (2016-7-6 10:55):**  
-numeric\_limits<double>::max()可以替换为numeric\_limits<double>::lowest()

class Solution {

public:

double maxValue(const vector<double>& nums) {

int s = nums.size();

if (s == 0) {

return 0;

}

dp\_ = vector<vector<double>>(s, vector<double>(s, -numeric\_limits<double>::max()));

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

dp\_[i][i] = nums[i];

}

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

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

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

dp\_[j][i] = max(dp\_[j][i], max(dp\_[j][k] + dp\_[k + 1][i], dp\_[j][k] \* dp\_[k + 1][i]));

}

}

}

return dp\_[0][s - 1];

}

string maxValueFormula(const vector<double>& nums) {

string res = "";

int s = nums.size();

if (s == 0) {

return res;

}

maxValue(nums);

list<string> res\_list;

maxValueFormulaHelper(nums, 0, s - 1, res\_list, res\_list.begin(), res\_list.begin(), "+");

for (auto it = res\_list.begin(); it != res\_list.end(); ++it) {

res.append(\*it);

}

return res;

}

private:

void maxValueFormulaHelper(const vector<double>& nums, int start\_i, int end\_i, list<string>& res\_list, list<string>::iterator list\_b, list<string>::iterator list\_e, string last\_op) {

if (start\_i == end\_i) {

res\_list.insert(list\_b, to\_string(nums[start\_i]));

return;

}

auto next\_b = list\_b;

auto next\_e = list\_e;

for (int k = start\_i; k < end\_i; ++k) {

if ((dp\_[start\_i][k] + dp\_[k + 1][end\_i]) == dp\_[start\_i][end\_i]) {

next\_b = res\_list.insert(list\_b, "+");

if (last\_op != "+") {

res\_list.insert(next\_b, "(");

next\_e = res\_list.insert(list\_e, ")");

}

maxValueFormulaHelper(nums, start\_i, k, res\_list, next\_b, next\_b, "+");

maxValueFormulaHelper(nums, k + 1, end\_i, res\_list, next\_e, next\_e, "+");

break;

}

else if ((dp\_[start\_i][k] \* dp\_[k + 1][end\_i]) == dp\_[start\_i][end\_i]) {

next\_b = res\_list.insert(list\_b, "\*");

maxValueFormulaHelper(nums, start\_i, k, res\_list, next\_b, next\_b, "\*");

maxValueFormulaHelper(nums, k + 1, end\_i, res\_list, next\_e, next\_e, "\*");

break;

}

}

}

vector<vector<double>> dp\_;

};

# Edit distance to palindrome

Let dp[i, j] = minimum number of removals needed to convert the substring [i, j] to a palindrome. We have:

dp[i, i] = 0 for all i (every single character is a palindrome)

To find dp[i, j], let's consider a random string. We have two possibilities:

1. The first and last characters are equal: a[i] == a[j]. In this case, we can reduce the problem to finding the minimum number of characters that need to be deleted in order to make the substring [i+1, j-1] a palindrome.
2. The first and last characters are not equal: a[i] != a[j]. In this case, we need to remove or replace one of them. We'll choose that which leads us to a better solution.

So we have:

dp[i, j] = dp[i + 1, j - 1] # if a[i] == a[j]

min(dp[i + 1, j], dp[i, j - 1], dp[i+1,j-1]) + 1 # otherwise

For your example of anuja. we'd get:

| 1 2 3 4 5

-------------

1 0 1 1 2 1

2 0 0 1 1 2

3 0 0 0 1 1

4 0 0 0 0 1

5 0 0 0 0 0

int len = s.length();

int[][] dp = new int[len][len];

for(int i = 0;i < len;i++)

dp[i][i] = 0;

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

for(int i = j-1;i >= 0;i--)

if(s.charAt(i) == s.charAt(j))

dp[i][j] = dp[i+1][j-1];

else

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

# Number of islands 1, 2;

# 用mutex 实现deadlock的一种情况

**public** **class** myDeadLock {

**public** **static** String *lock1* = "lock1";

**public** **static** String *lock2* = "lock2";

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Thread1 myThread1 = **new** Thread1();

Thread2 myThread2 = **new** Thread2();

myThread1.start();

myThread2.start();

}

**static** **class** Thread1 **extends** Thread{

**public** **void** run(){

**synchronized**(*lock1*){

System.***out***.println("Thread 1 has lock1 right now");

**try**{

Thread.*sleep*(100);

}**catch**(InterruptedException e){}

System.***out***.println("Thread 1 is waiting for lock2");

**synchronized**(*lock2*){

System.***out***.println("Thread 1 has both lock1 and lock2 right now");

}

}

}

}

**static** **class** Thread2 **extends** Thread{

**public** **void** run(){

**synchronized**(*lock2*){

System.***out***.println("Thread 2 has lock2 right now");

**try**{

Thread.*sleep*(100);

}**catch**(InterruptedException e){};

System.***out***.println("Thread 2 is waiting for lock1");

**synchronized**(*lock1*){

System.***out***.println("Thread 2 has both lock1 and lock2 right now");

}

}

}

}

}

# 1) 怎么分类垃圾邮件

贝叶斯过滤器是一种统计学过滤器，建立在已有的统计结果之上。所以，我们必须预先提供两组已经识别好的邮件，一组是正常邮件，另一组是垃圾邮件。

我们用这两组邮件，对过滤器进行"训练"。这两组邮件的规模越大，训练效果就越好。Paul Graham使用的邮件规模，是正常邮件和垃圾邮件各4000封。

"训练"过程很简单。首先，解析所有邮件，提取每一个词。然后，计算每个词语在正常邮件和垃圾邮件中的出现频率。比如，我们假定"sex"这个词，在4000封垃圾邮件中，有200封包含这个词，那么它的出现频率就是5%；而在4000封正常邮件中，只有2封包含这个词，那么出现频率就是0.05%。（【注释】如果某个词只出现在垃圾邮件中，Paul Graham就假定，它在正常邮件的出现频率是1%，反之亦然。这样做是为了避免概率为0。随着邮件数量的增加，计算结果会自动调整。）

**九、贝叶斯过滤器的使用过程**

现在，我们收到了一封新邮件。在未经统计分析之前，我们假定它是垃圾邮件的概率为50%。（【注释】有研究表明，用户收到的电子邮件中，80%是垃圾邮件。但是，这里仍然假定垃圾邮件的"先验概率"为50%。）

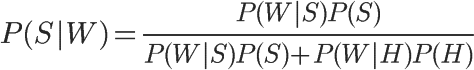
我们用S表示垃圾邮件（spam），H表示正常邮件（healthy）。因此，P(S)和P(H)的先验概率，都是50%。

http://chart.googleapis.com/chart?cht=tx&chl=P(S)%3DP(H)%3D50%25&chs=40

然后，对这封邮件进行解析，发现其中包含了sex这个词，请问这封邮件属于垃圾邮件的概率有多高？

我们用W表示"sex"这个词，那么问题就变成了如何计算P(S|W)的值，即在某个词语（W）已经存在的条件下，垃圾邮件（S）的概率有多大。

根据条件概率公式，马上可以写出



公式中，P(W|S)和P(W|H)的含义是，这个词语在垃圾邮件和正常邮件中，分别出现的概率。这两个值可以从历史资料库中得到，对sex这个词来说，上文假定它们分别等于5%和0.05%。另外，P(S)和P(H)的值，前面说过都等于50%。所以，马上可以计算P(S|W)的值：

因此，这封新邮件是垃圾邮件的概率等于99%。这说明，sex这个词的推断能力很强，将50%的"先验概率"一下子提高到了99%的"后验概率"。

**十、联合概率的计算**

做完上面一步，请问我们能否得出结论，这封新邮件就是垃圾邮件？

回答是不能。因为一封邮件包含很多词语，一些词语（比如sex）说这是垃圾邮件，另一些说这不是。你怎么知道以哪个词为准？

Paul Graham的做法是，选出这封信中P(S|W)最高的15个词，计算它们的联合概率。

将P(S|W1)记为P1

**最终的计算公式**

将上面的公式扩展到15个词的情况，就得到了最终的概率计算公式：

# Knight tour. Give a start point, to find the shortest path to destination. Provide the API to return all valid next move for current position.

bfs(board, **int**[] orig, **int**[] dest){

HashSet<**int**[]> visited = **new** HashSet<>();

HashMap<**int**[], **int**[]> path = **new** HashMap<>();

path.put(orig, **null**);

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

q.add(orig);

**int** step = 1;

**boolean** found = **false**;

**while**(!q.isEmpty()){

**int** size = q.size();

**while**(size-- > 0){

**int**[] currPos = q.poll();

**int**[][] nextMoves = getNextMove(currPos);

**for**(**int**[] nextPos: nextMoves){

**if**(visited.add(nextPos)){

path.put(nextPos, currPos);

**if**(nextPos == dest){

found = **true**;

**break**;

}

}

}

**if**(found) **break**;

}

**if**(found)**break**;

step++;

}

**if**(found){

**while**(dest != **null**){

System.out.println(dest);

dest = path.get(dest);

}

}

## Follow up, to find all shortest pathes,

DFS+backtracking by using the mini step we find in BFS function

dfs(**int**[] orig, **int**[] dest, **int** currStep, **int** maxStep, List<**int**[]> path, List<List<**int**[]>> res){

**if**(currStep == maxStep){

**if**(orig == dest){

List<**int**[]>copyPath = **new** ArrayList<**int**[]>(path);

copyPath.add(dest);

res.add(copyPath);

}

**return**;

}

path.add(orig);

**for**(**int**[] next: getNextMove(orig)){

dfs(next, dest, currStep+1, maxStep, path, res);

}

path.remove(path.size()-1);

}

# 给一个DAG，要找inorder traversal的第k个char

其中DAG中有重复的char，但是不同的node，而且一个node可以被访问多次，因为不是树。. Waral 鍗氬鏈夋洿澶氭枃绔�,  
树的inorder traversal和DAG有什么差别，然后面试官提了下有些点会被重复访问才开窍了，用一个hashmap存一下之前访问过的node的traversal长度即可。

# 输出任意字符组合成的在字典里的所有String.

public List<String> solution (char[] array, List<String> dic) {}   
举个例子: char[] array = {'A', 'B', 'C', 'A'}, List<String> = {"A","AA","BA"};  正确的输出: "A", "AA","BA";   非法输出:"AAA","BB","AB"........

# 设计bloom filter， add, check, remove,rescale,不需要hash

counting bloom filter  每个slot不再是一个bit 而是一个小的counter  这样就可以remove了

By given number of items n and probability of false positive p, we can derive the total bits we need(m) and how many hash function we need to achieve optimal result

This results in:

{\displaystyle m=-{\frac {n\ln p}{(\ln 2)^{2}}}}

with the corresponding number of hash functions *k*:

**private** **final** **int** capability;

**private** **final** **int**[] hashSeeds;

**private** **final** BitSet bs;

**public** BloomFilter(**int** m, **int** h){

bs = **new** BitSet(m);

capability = h;

hashSeeds = **new** **int**[h];

Random rd = **new** Random(System.*currentTimeMillis*());

**for**(**int** i = 0;i < h;i++)

hashSeeds[i] = rd.nextInt();

}

## //add

**public** **void** add(**int** val){

**byte**[] b = **new** **byte**[]{

(**byte**)(val>>24),

(**byte**)(val>>16),

(**byte**)(val>>8),

(**byte**)(val)

};

**for**(**int** i = 0;i < capability;i++){

**int** index = hashCode(b, 4, hashSeeds[i]);

bs.set(index, **true**);

}

}

## //check

**public** **boolean** check(**int** val){

**byte**[] b = **new** **byte**[]{

(**byte**)(val>>24),

(**byte**)(val>>16),

(**byte**)(val>>8),

(**byte**)(val)

};

**for**(**int** i = 0;i < capability;i++){

**int** index = hashCode(b, 4, hashSeeds[i]);

**if**(!bs.get(index))

**return** **false**;

}

**return** **true**;

}

## For resize

since the bloom filter doesn’t include the info about all items, so we need delete current bloom filter and create a new one by recomputing the m and k.

# 怎么设计discover的搜索功能

parse文章成单词，做成trie上的倒排索引，因为是prefix 所以每一个node都要记录当前能输出的文章列表 然后关联查询时找出多个prefix对应的列表 最好是排序过的  
然后做intersection查找，就是基本搜索引擎的做法（nextGEQ）

# 给一个树的root node,返回same subtree及个数:

                    10  
                /        \  
            8                3  
          /  \            /     \  
         4    5        8           5  
             /       /   \        /  
            3       4     5      3  
                         /  
                        3  
      Expects:  
             8  
            / \  
           4   5      Count: 2  
              /  
             3. 鐣欏鐢宠璁哄潧-涓€浜╀笁鍒嗗湴  
  
             5  
            /         Count: 3  
           3  
    \*/

// This is the text editor interface.

// Anything you type or change here will be seen by the other person in real time

import java.io.\*;

import java.util.\*;

public class Solution {

public static class TreeNode {

int val;

TreeNode left, right;

public TreeNode(int val) {

this.val = val;

left = null;

right = null;

}

}

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

public static Map<String, Integer> sameSubTree(TreeNode root) {

dfs(root);

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

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

if (key.length() > 3 && map.get(key) > 1) res.put(key, map.get(key)); }

return res;

}

private static String dfs(TreeNode root) {

if (root == null) return "#";

String left = dfs(root.left);

String right = dfs(root.right);

String all = Integer.toString(root.val) + left + right;

if (!map.containsKey(all)) map.put(all, 0);

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

return all;

}

public static void main(String[] args) {

TreeNode root = new TreeNode(10);

root.left = new TreeNode(3);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

root.right.left = new TreeNode(4);

root.right.right = new TreeNode(5);

Map<String, Integer> res = sameSubTree(root);

for (String key : res.keySet()) {

System.out.println(key + " " + map.get(key

)

}

}

# 一幅画给你一个点和一个要填的颜色， 把周围相同颜色的都变成这个颜色。

BFS or DFS

**public** **static** **void** paint(**int**[][] matrix, **int** x, **int** y, **int** color){

**int**[][] direction = {{-1, 0},

{1, 0},

{0,-1},

{0,1}};

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

q.add(**new** **int**[]{x, y});

**int** origColor = matrix[x][y];

**while**(!q.isEmpty()){

**int**[] curr = q.poll();

matrix[curr[0]][curr[1]] = color;

**for**(**int**[] dir: direction){

**int** newX = curr[0]+dir[0];

**int** newY = curr[1]+dir[1];

**if**(newX >= 0 && newX < matrix.length && newY >= 0

&& newY < matrix[0].length && matrix[newX][newY] == origColor)

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

}

}

}

# 输入是一个Interval list，要求random输出其中一个数。

比如：输入[1, 6] [10, 12], 就从1， 2， 3， 4， 5， 6， 10， 11， 12这九个数里随机选一个。要求用reservoir sampling

后面产生的随机数如果落在(0, 之前intervals中所有点的个数之和)，就不更新， 否则更新。用reservoir sampling的话复杂度我觉得应该就是O(k)的吧

public int get(List<Interval> lists) {  
    int candidate; 鏉ユ簮涓€浜�.涓夊垎鍦拌鍧�.   
    int total=0;  
    Random rand=new Random();  
    for(Interval it:lists) {  
         int len=it.end-it.start+1;  
         int index=rand.next(total+len);  
         if(index>=total) {  
               candidate=it.start+index-total;. 1point 3acres 璁哄潧  
         }  
         total+=len;  
   }  
   return candidate;

}

# 多个节点的最小公共祖先

**public** **static** **void** main(String[] args) {

TreeNode root = **new** TreeNode(3);

root.left = **new** TreeNode(5);

root.right = **new** TreeNode(1);

root.left.left = **new** TreeNode(6);

root.left.right = **new** TreeNode(2);

root.left.right.left = **new** TreeNode(7);

root.left.right.right = **new** TreeNode(4);

root.right.left = **new** TreeNode(0);

root.right.right = **new** TreeNode(8);

TreeNode[] children = {root.left.left, root.left.right.left, root.left.right.right, root.right.right, **new** TreeNode(10)};

TreeNode common = *getAncestor*(root, children);

**if**(common != **null**)

System.***out***.println("The common ancestor is "+common.val);

**else**

System.***out***.println("There is no ancestor!");

}

**public** **static** TreeNode getAncestor(TreeNode root, TreeNode[] children){

**if**(children.length == 2){

**return** *getAncestorHelper*(root, children[0], children[1]);

}

**else**{

TreeNode[] sub = **new** TreeNode[children.length-1];

**for**(**int** i = 0; i < sub.length;i++)

sub[i] = children[i];

**return** *getAncestorHelper*(root, *getAncestor*(root, sub), children[children.length-1]);

}

}

**public** **static** TreeNode getAncestorHelper(TreeNode root, TreeNode p, TreeNode q){

**if**(root == **null** || p == **null** && q == **null**) **return** **null**;

TreeNode leftResult = *getAncestorHelper*(root.left, p, q);

TreeNode rightResult = *getAncestorHelper*(root.right, p, q);

**if**(leftResult != **null** && rightResult != **null**)

**return** root;

**else** **if**(leftResult != **null**)

**return** leftResult;

**else**

**return** rightResult;

}

}

# 设计视频网站的记录你上次看一个视频看到哪里来的视频标记，要求讲了一下后端架构，客户端和服务器之间怎么交互

# 设计windows paint系统，

问这个画图要好几个组来做，如何把他分成不同模块，交给不同组独立开发，然后各个组之间的API要怎么设计之后写了个面经里的算法题

# Check whether a given graph is Bipartite or not

1. Assign RED color to the source vertex (putting into set U).  
2. Color all the neighbors with BLUE color (putting into set V).  
3. Color all neighbor’s neighbor with RED color (putting into set U).  
4. This way, assign color to all vertices such that it satisfies all the constraints of m way coloring problem where m = 2.  
5. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite)

**class** Graph{

**public** List<Integer>[] adjacent;

**public** **int** size;

**public** Graph(**int** s){

size = s;

adjacent = **new** ArrayList[size];

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

adjacent[i] = **new** ArrayList<Integer>();

}

**public** **void** addEdge(**int** i, **int** j){

adjacent[i].add(j);

}

**public** **boolean** isBipartite(){

**int**[] color = **new** **int**[size];

Arrays.*fill*(color, -1);

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

**if**(color[i] == -1)

color[i] = 1;

**for**(**int** j: adjacent[i]){

**if**(color[j] == color[i]){

**return** **false**;

}

**else** **if**(color[j] == -1)

color[j] = 1-color[i];

}

}

**return** **true**;

}

}

# coding 一道保龄球算分的integer array题目

1. Strike: When all ten pins are knocked down with the first ball (called a strike and typically rendered as an "X" on a scoresheet), a player is awarded ten points, plus a bonus of whatever is scored with the next two balls.
2. Spare: A "spare" is awarded when no pins are left standing after the second ball of a frame; i.e., a player uses both balls of a frame to clear all ten pins. A player achieving a spare is awarded ten points, plus a bonus of whatever is scored with the next ball (only the first ball is counted)
3. Otherwise, there is no bonus.
4. The maximum score is 300 with 12 strike

**public** BowlingGame(){

roll = **new** **int**[21];

size = 0;

}

**public** **void** addRoll(**int** i){

**if**(size == 21) **return**;

roll[size++] = i;

}

**public** **int** getScore(){

**int** frame = 0, score = 0;

**for**(**int** i = 0;i < 10;i++){

**if**(isStrike(frame)){

score += 10+getStrikeBonus(frame);

frame++;

}

**else** **if**(isSpare(frame)){

score += 10+getSpareBonus(frame);

frame += 2;

}

**else**{

score += getRollSum(frame);

frame += 2;

}

}

**return** score;

}

**private** **boolean** isStrike(**int** f){

**return** roll[f] == 10;

}

**private** **boolean** isSpare(**int** f){

**return** getRollSum(f) == 10;

}

**private** **int** getStrikeBonus(**int** f){

**return** getRollSum(f+1);

}

**private** **int** getSpareBonus(**int** f){

**return** roll[f+2];

}

**private** **int** getRollSum(**int** f){

**return** roll[f]+roll[f+1];

}

# 从很大的integer steam里找出duplicate数字（followup需要用到bloom filter）

# BST的构建

# Java stream io

Scanner sc = new Scanner(System.in);

int i = sc.nextInt();

Scanner scanner = new Scanner(System.in);

String username = scanner.nextLine();

# Bidirection BFS

# array里只有一个数字出现超过50%，其他都只有一次。1. return超过50%次的数字。 follow up： array里只有一个数字出现超过一次，其他都只有一次。返回超过超过一次的数字的出现概率。要求：constant space, time: O(n)

I think there is another info that all number is in 0-n-1 or something, so we can use original arr to store the info.

    static int maxRepeating(int arr[], int n, int k)

    {

        // Iterate though input array, for every element

        // arr[i], increment arr[arr[i]%k] by k

        for (int i = 0; i< n; i++)

            arr[(arr[i]%k)] += k;

        // Find index of the maximum repeating element

        int max = arr[0], result = 0;

        for (int i = 1; i < n; i++)

        {

            if (arr[i] > max)

            {

                max = arr[i];

                result = i;

            }

        }

        /\* Uncomment this code to get the original array back

        for (int i = 0; i< n; i++)

          arr[i] = arr[i]%k; \*/

        // Return index of the maximum element

        return result;

}

# 按层打印公司结构

Input是”first name, last name, jointime, boss”

Topologic sort

**public** **static** **void** main(String[] args) {

String[] input = {"Jack Liu,20021203,Xin Zhang",

"Didi Wang,20011122,Xin Zhang",

"Cici Smith,20121212,Didi Wang",

"Ella Via,20130104,Didi Wang",

"Frank Lee,20080808,Dong Di",

"Sheng Tao,20091203,Dong Di",

"Ta Ta,20101009,Frank Lee",

"Ou Yu,20160103,Ta Ta"

};

*printCompanyLevel*(input);

}

**public** **static** **void** printCompanyLevel(String[] info){

HashMap<String, Integer> degree = **new** HashMap<>();

HashMap<String, List<String>> relation = **new** HashMap<>();

**for**(String in: info){

String[] arr = in.split(",");

**if**(!relation.containsKey(arr[2]))

relation.put(arr[2], **new** ArrayList<String>());

relation.get(arr[2]).add(arr[0]);

**if**(!degree.containsKey(arr[2]))

degree.put(arr[2], 0);

degree.put(arr[0], degree.getOrDefault(arr[0], 0)+1);

}

Queue<String> q = **new** LinkedList<String>();

**for**(String key: degree.keySet()){

**if**(degree.get(key) == 0)

q.add(key);

}

**while**(!q.isEmpty()){

**int** size = q.size();

**while**(size-- > 0){

String curr = q.poll();

System.***out***.print(curr+" ");

**if**(relation.containsKey(curr))

**for**(String sub: relation.get(curr)){

degree.put(sub, degree.get(sub)-1);

**if**(degree.get(sub) == 0)

q.add(sub);

}

}

System.***out***.println();

}

}

提供一个method， add(task,delayms). 添加task， delayms表示的是从现在过后多久开始运行这个task。这是个多线程的设计题，问了怎么保存task，怎么consumetask，怎么开始新的thread，memoryusage， 设计thread pool等等，还要我写multi-thread的code

# 给两个string的数组和一个pattern数组，判断将两个string数组分别和pattern转化后的结果是否相同。

例如s1={"abc", "a", "ccc"}, s2={"bc", "aa", "d"}, pattern={1, 0}，则s1和p的转化结果是"aabc"，s2和p的转化结果也是是"aabc"，则返回true。如果pattern是{0, 1}，则转化结果分别是"abca", "bcaa"，则返回false。followup是，如果给定s1和s2，判断是否存在一个pattern使得转化结果一致。 过程中要求分析算法复杂度。

public class StringPattern {  
        private String SEP = ",";  
          
        public static void main(String[] args) {  
                String[] words1 = {"abc", "a", "cc"};  
                String[] words2 = {"bc", "aa", "c"};. 鍥磋鎴戜滑@1point 3 acres  
                StringPattern sp = new StringPattern();  
                System.out.println(sp.hasPattern(words1, words2));  
        }.         
          
        public boolean hasPattern(String[] words1, String[] words2) {-google 1point3acres  
                int len = Math.min(words1.length, words2.length);  
                Map<Integer, Set<String>> map = new HashMap<Integer, Set<String>>();  
                for (int i=0; i<len; i++) map.put(i, new HashSet<String>());.鐣欏璁哄潧-涓€浜�-涓夊垎鍦�  
                for (int i=0; i<len; i++) {  
                        if (helper(words1, words2, words1*, words2, i, map)) return true;  
                }  
                return false;  
        }.1point3acres缃�  
          
          
        private boolean helper(String[] words1, String[] words2, String s1, String s2, int pos, Map<Integer, Set<String>> map) {  
                if (map.get(pos).contains(""+s1.length()+SEP+s2.length())) return false;  
                map.get(pos).add(""+s1.length()+SEP+s2.length());  
                int i=0;  
                int j=0;  
                int len = Math.min(words1.length, words2.length);  
                for (; i<s1.length() && j<s2.length(); i++, j++) {. more info on 1point3acres.com  
                        if (s1.charAt(i) != s2.charAt(j)) break;  
                }  
                if (i!=s1.length() && j!=s2.length()) return false;  
                if (i == s1.length() && j == s2.length()) return true;  
                if (i < s1.length()) {  
                        String prefix = s1.substring(i);  
                        for (int index=0; index<len; index++) {. 1point 3acres 璁哄潧  
                                if (words2[index].startsWith(prefix)) {  
                                        if (helper(words1, words2, words1[index], words2[index].substring(prefix.length()), index, map))  
                                                return true;  
                                }  
                        }  
                } else {  
                        String prefix = s2.substring(j);  
                        for (int index=0; index<len; index++) {  
                                if (words1[index].startsWith(prefix)) {  
                                        if (helper(words1, words2, words1[index].substring(prefix.length()), words2[index], index, map))  
                                                return true;  
                                }  
                        }  
                }  
                return false;  
        }  
}.*

# Median of stream

Solution1 two priorityqueue

Solution2 BST

[https://discuss.leetcode.com/top ... 9-82-of-submissions](https://discuss.leetcode.com/topic/61789/22ms-java-solution-using-binary-tree-beats-99-82-of-submissions" \t "_blank)

# Cycle Detection

DFS VS BFS

The graph has a cycle if and only if there exists a back edge. A back edge is an edge that is from a node to itself (selfloop) or one of its ancestor in the tree produced by DFS forming a cycle.

Both approaches above actually mean the same. However, this method can be applied only to **undirected graphs**.

The reason why this algorithm doesn't work for directed graphs is that in a directed graph 2 different paths to the same vertex don't make a cycle. For example: A-->B, B-->C, A-->C - don't make a cycle whereas in undirected ones: A--B, B--C, C--A does.

## Find a cycle in undirected graphs

An undirected graph has a cycle if and only if a depth-first search (DFS) finds an edge that points to an already-visited vertex (a back edge).

BFS could also work.

**class** Graph{

**private** List<Integer>[] adj;

**private** **int** v;

**public** Graph(**int** v){

**this**.v = v;

adj = **new** ArrayList[v];

**for**(**int** i = 0;i < v;i++)

adj[i] = **new** ArrayList<Integer>();

}

**public** **void** addEdge(**int** i, **int** j){

adj[i].add(j);

adj[j].add(i);

}

**public** **boolean** isCycleBFS(){

**boolean**[] visited = **new** **boolean**[v];

**for**(**int** i = 0;i < v;i++)

**if**(!visited[i] && isCycleBFSUtil(i, visited))

**return** **true**;

**return** **false**;

}

**public** **boolean** isCycleBFSUtil(**int** i, **boolean**[] visited){

visited[i] = **true**;

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

q.offer(**new** **int**[]{i, -1});

**while**(!q.isEmpty()){

**int**[] curr = q.poll();

**for**(**int** nbr: adj[curr[0]]){

**if**(!visited[nbr]){

visited[nbr] = **true**;

q.add(**new** **int**[]{nbr, curr[0]});

}

**else** **if**(nbr != curr[1])

**return** **true**;

}

}

**return** **false**;

}

**public** **boolean** isCycleDFS(){

**boolean**[] visited = **new** **boolean**[v];

**for**(**int** i = 0;i < v;i++)

**if**(!visited[i] && isCycleDFSUtil(i, visited, -1))

**return** **true**;

**return** **false**;

}

**public** **boolean** isCycleDFSUtil(**int** i, **boolean**[] visited, **int** parent){

visited[i] = **true**;

**for**(**int** nbr: adj[i]){

**if**(!visited[nbr]){

**if**(isCycleDFSUtil(nbr, visited, i))

**return** **true**;

}

**else** **if**(nbr != parent)

**return** **true**;

}

**return** **false**;

}

}

## Find a cycle in directed graphs

In addition to visited vertices we need to keep track of vertices currently in recursion stack of function for DFS traversal. If we reach a vertex that is already in the recursion stack, then there is a cycle in the tree.

<http://www.geeksforgeeks.org/detect-cycle-in-a-graph/>

But BFS could not work for directed graph， since we need to track the path. Even we has visited one vertex, but it may be no cycle, since we reach to this vertex by other path.

**public** **boolean** hasCycle(){

**boolean**[] visited = **new** **boolean**[size];

**boolean**[] recStack = **new** **boolean**[size];

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

**if**(hasCycleUtil(i, visited, recStack))

**return** **true**;

**return** **false**;

}

**public** **boolean** hasCycleUtil(**int** curr, **boolean**[] visited, **boolean**[] recStack){

**if**(!visited[curr]){

visited[curr] = **true**;

recStack[curr] = **true**;

**for**(**int** i: adjacent[curr]){

**if**(!visited[i] && hasCycleUtil(i, visited, recStack))

**return** **true**;

**else** **if**(recStack[i])

**return** **true**;

}

}

recStack[curr] = **false**;

**return** **false**;

}

# Company Party

[**http://blog.csdn.net/edonlii/article/details/8623058**](http://blog.csdn.net/edonlii/article/details/8623058)

Similar to house robber III or same if there are only two children.

Below code is for general solution

**class** Person {

**public** **int** like;

**public** List<Person> subordinate;

**public** Person(**int** likeness){

like = likeness;

subordinate = **new** ArrayList<Person>();

}

**public** **void** addSubordinate(Person p){

subordinate.add(p);

}

}

**public** **static** **int**[] calculateMax(Person p){

**int**[][] likeArr = **new** **int**[p.subordinate.size()][2];

**for**(**int** i = 0; i < likeArr.length;i++)

likeArr[i] = *calculateMax*(p.subordinate.get(i));

**int**[] res = **new** **int**[2];

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

res[0] += likeArr[i][1];

res[1] += Math.*max*(likeArr[i][0], likeArr[i][1]);

}

res[0] += p.like;

**return** res;

}

**LC 120**

**Follow up return all min path**

# 实现 无向图 的数据结构，支持方法包括 add边, has边, #of边, 序列化, 反序列化equals( another graph);

**class** Graph{

**public** **int** size, edgeNum;

**public** List<Integer>[] adjacent;

**public** Graph(**int** size){

**this**.size = size;

edgeNum = 0;

adjacent = **new** ArrayList[size];

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

adjacent[i] = **new** ArrayList<Integer>();

}

**public** **void** addEdge(**int** orig, **int** dest){

edgeNum++;

adjacent[orig].add(dest);

adjacent[dest].add(orig);

}

**public** **int** numberOfEdge(){

**return** edgeNum;

}

**public** **boolean** hasEdge(**int** i, **int** j){

**if**(i >= size || j >= size || i < 0 || j < 0)

**return** **false**;

**if**(adjacent[i].indexOf(j) != -1)

**return** **true**;

**else** **return** **false**;

}

For serialize and de-serialize, basically you have two (good) options:

* [an adjacency list representation](http://en.wikipedia.org/wiki/Adjacency_list)
* [an adjacency matrix representation](http://en.wikipedia.org/wiki/Adjacency_matrix)

in which the adjacency list representation is best used for a sparse graph, and a matrix representation for the dense graphs.

# 找到一个index使它左边的和与右边的和相等。比如｛1，2，3，4，3，2，1｝->3

**public** **static** **int** findEqualIndex(**int**[] arr){

**int** sum = 0, currSum = 0;

**for**(**int** i = 1; i < arr.length;i++)

sum += arr[i];

**for**(**int** i = 1;i < arr.length;i++){

sum -= arr[i];

currSum += arr[i-1];

**if**(sum == currSum)

**return** i;

}

**return** -1;

}

## follow up: does this work for negative integer? Yes

## follow up2: If there are multiple index can make it equal? Add the index to arraylist without early return;

# 比方说给你50个task，有个API假设已经有了，是用来run这些task的，但是这个API最多每次只能同时run3个。并且run的顺序根据priority level来定，让实现。具体包括 get，change priority 和run，get就是新的task，change priority就是把已有的还没运行的task更改priority level，run就是运行同时保证运行完一个马上推入下一个运行，

# 给了一个Person class， 有score和nextSnap两个属性，每个Person有一个朋友的list，也就是nextSnap，输入是（Person， maxStep）， 在maxStep步数以内算max score

注意的是下一个Friend可能会指向上一个Person，要注意回溯的问题

**public** **class** PeopleScore {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

HashSet<People> set = **new** HashSet<>();

People a = **new** People(4);

People b = **new** People(5);

People c = **new** People(8);

People d = **new** People(1);

People e = **new** People(6);

People f = **new** People(5);

a.add(b);

a.add(c);

a.add(d);

b.add(c);

b.add(e);

c.add(f);

d.add(f);

c.add(e);

c.add(a);

System.***out***.println(*getMaxScore*(a, 5, set));

}

**public** **static** **int** getMaxScore(People p, **int** k, HashSet<People> set){

**if**(k < 1) **return** 0;

**if**(k == 1) **return** p.getScore();

**int** max = 0;

set.add(p);

**for**(People nextP: p.nextSnap()){

**if**(!set.contains(nextP))

max = Math.*max*(max, *getMaxScore*(nextP, k-1, set));

}

set.remove(p);

max += p.getScore();

**return** max;

}

**static** **class** People{

**private** **int** score;

**private** List<People> nbr;

**public** People(**int** s){

score = s;

nbr = **new** ArrayList<>();

}

**public** **void** add(People p){

nbr.add(p);

}

**public** **int** getScore(){

**return** score;

}

**public** List<People> nextSnap(){

**return** nbr;

}

}

}

给一个整数n，输出俩数x和y，使得x\*y的值在 [n, n+2] 的范围内，同时保证 |x - y| 最小， e.g. n=25, return x=y=5 或  n=22， return x=4 y=6

int[] searchXandY(int n) {

int[] result = new int[2];

if(n <= 0) {

throw new IllegalArgumentException("Illegal input");

}

int cand1 = (int)Math.sqrt(n+2);

int cand2 = (int)Math.sqrt(n+2);

while(true) {

if(cand1 \* cand2 >= n && cand1 \* cand2 <= n +2) {

result[0] = cand1;

result[1] = cand2;

break;

} else if(cand1 \* cand2 < n) {

cand1++;

} else {

cand2--;

}

}

return result;

}

# given an int n, write a function to generate false in a probability of 1/(2^n).

e.g. n = 3, 那么函数以1/8的概率返回false， 以7/8的概率返回true

public static boolean probablisticFalse(int n){

boolean res = false;

Random rd = new Random();

while(n-- > 0)

res |= rd.nextInt(2) == 0;

return res;

# k-Snap point:

Consider a grid where all the points are represented by integers.  
  
.........................................  
...(-2,2)  (-1,2)  (0,2)  (1,2)  (2,2)...  
...(-2,1)  (-1,1)  (0,1)  (1,1)  (2,1).... 鍥磋鎴戜滑@1point 3 acres  
...(-2,0)  (-1,0)  (0,0)  (1,0)  (2,0)....鏈枃鍘熷垱鑷�1point3acres璁哄潧  
...(-2,-1) (-1,-1) (0,-1) (1,-1) (2,-1)...  
...(-2,-2) (-1,-2) (0,-2) (1,-2) (2,-2)...  
..........................................  
  
k-Snap point: A point whose digits sum up to less than or equal to k. In this  
question, we ignore all the signs in the number.  For exxample, (1, 0) is a 1-snap point, (0, 10) is a 1-snap point, and (-100, 0) is also a 1-snap point; however (11, 0) is not a 1-snap point.  
  
Question 1: Implement the following function  
boolean isSnapPoint(Point p, int k)  
  
Returns true if p is a k-snap point, and false otherwise.  
  
Reachable k-snap point: A k-snap point is a reachable k-snap point if there is a path from (0,0) to that point, where the path only consists of k-snap points.-google 1point3acres  
. visit 1point3acres.com for more.  
Question 2: Given k, return all the reachable k-snap points.

public boolean  isSnapPoint(Point p, int k) {  
                int x = Math.abs(p.x);  
                int y = Math.abs(p.y);  
                int val = 0;  
                while (x > 0) {  
                        val += x % 10;  
                        x /= 10;. more info on 1point3acres.com  
                }.鏈枃鍘熷垱鑷�1point3acres璁哄潧  
                while (y > 0) {  
                        val += y % 10;  
                        y /= 10;  
                }  
                return val <= k;  
        }  
.1point3acres缃�  
        public HashSet<Point> ReachableKSnapPoint(int k) {  
                HashSet<Point> points = new HashSet<Point>();  
                if (k < 0) return points;  
                helper(new Point(0, 0), k, points);  
                return points;  
        }  
          
        public void helper(Point point, int k, HashSet<Point> points) {  
                if(!isSnapPoint(point, k)) return;  
                points.add(point);  
                Point left = new Point(point.x - 1, point.y);  
                if (!points.contains(left)) helper(left, k, points);  
                Point right = new Point(point.x + 1, point.y);  
                if (!points.contains(right)) helper(right, k, points);.  
                Point upper = new Point(point.x, point.y + 1);.1point3acres缃�  
                if (!points.contains(upper)) helper(upper, k, points);鍗氬鏈夋洿澶氭枃绔�,  
                Point lower = new Point(point.x, point.y - 1);  
                if (!points.contains(lower)) helper(lower, k, points);-google 1point3acres  
        }@1point 3 acres  
        public class Point {  
                public int x, y;  
                public Point(int x, int y) {  
                        this.x = x;. 鍥磋鎴戜滑@1point 3 acres  
                        this.y = y;  
                }.鐣欏璁哄潧-涓€浜�-涓夊垎鍦�  
                public boolean equals(Object point) {  
                if (!(point instanceof Point)) {  
                    return false;  
                }  
                        Point p = (Point) point;  
                        return this.x == p.x && this.y == p.y;  
                }  
                public int hashCode() {  
                     return (x + " " + y).hashCode();  
                }  
        }

# 给我看了一个snapchat给好友群发消息的功能，可以任意选中和删除想要投递消息的好友，并显示群发好友的list(按先后选中的顺序)，设计一个数据结构，实现

toggle(String username);  
getList();  
LZ给的hashmap + doubly linked list的设计，类比LRU cache, 详细解释了一下，分析了时间复杂度

# storeBadIps()

Give a bad ip list, check whether input ip in the range E.g. storeBadIps(["7.0.0.0/8", "10.3.0.0/16", "6.7.8.134/32"])

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<String> rules = **new** ArrayList<String>(Arrays.*asList*("7.7.7.7/12","123.2.67.3/8"));

List<String> IPs = **new** ArrayList<String>(Arrays.*asList*("23.2.4.1","7.3.4.1","5.1.2.3","123.2.67.3","7.8.4.1","7.16.1.1"));

List<Boolean> res = *isBadIP*(rules, IPs);

System.***out***.println(res);

}

**public** **static** List<Boolean> isBadIP(List<String> IPRange, List<String> IPs){

List<Boolean> res = **new** ArrayList<>();

Trie root = **new** Trie();

*insert*(root, IPRange);

**for**(String ip: IPs){

res.add(*isBadIPUtil*(root, ip));

}

**return** res;

}

**public** **static** **boolean** isBadIPUtil(Trie root, String ip){

**if**(ip.length() == 0) **return** **false**;

String[] ipFourNum = ip.split("\\.");

StringBuilder sb = **new** StringBuilder();

**for**(String part: ipFourNum){

sb.append(*getBinary*(part));

}

**return** *isStartWith*(root, sb.toString());

}

**public** **static** **boolean** isStartWith(Trie root, String s){

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

**int** index = s.charAt(i)-'0';

**if**(root.next[index] == **null**)

**return** **false**;

**if**(root.next[index].isFilter)

**return** **true**;

root = root.next[index];

}

**return** root.isFilter;

}

**public** **static** **void** insert(Trie root, List<String> IPRange){

Trie curr = **null**;

**for**(String ip: IPRange){

String range = *getRange*(ip);

curr = root;

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

**int** index = range.charAt(i)-'0';

**if**(curr.next[index] == **null**)

curr.next[index] = **new** Trie();

curr = curr.next[index];

}

curr.isFilter = **true**;

}

}

**public** **static** String getRange(String ip){

StringBuilder sb = **new** StringBuilder();

String[] ipPart = ip.split("/");

**int** bitCount = Integer.*parseInt*(ipPart[1]);

String[] ipFourNumber = ipPart[0].split("\\.");

**for**(String number: ipFourNumber)

sb.append(*getBinary*(number));

**return** sb.toString().substring(0,bitCount);

}

**public** **static** String getBinary(String s){

**int** number = Integer.*parseInt*(s);

StringBuilder sb = **new** StringBuilder();

**for**(**int** i = 7;i >= 0;i--){

sb.append(number&1);

number>>=1;

}

**return** sb.reverse().toString();

}

**static** **class** Trie{

**public** Trie[] next;

**public** **boolean** isFilter;

**public** Trie(){

next = **new** Trie[2];

isFilter = **false**;

}

}

}

# 小车通过雷达

Method：we can ignore the car size, so the problem turns to be that whether the radar area cover [0,1]. For given radars, we need to merge any intersection area, and update the covered area (only y dimension is enough)

**public** **static** **boolean** canCarPass(List<radar> list){

List<radarArea> areaList = **new** ArrayList<>();

**for**(radar i: list){

**boolean** merge = **false**;

**for**(radarArea ra: areaList){

merge = ra.merge(i);

**if**(merge)

**break**;

}

**if**(!merge){

radarArea newRA = **new** radarArea();

newRA.area.add(i);

areaList.add(newRA);

}

}

**for**(radarArea ra: areaList){

**if**(!ra.canPass())

**return** **false**;

}

**return** **true**;

}

**static** **class** radarArea{

**public** List<radar> area;

**public** **double** high;

**public** **double** low;

**public** radarArea(){

area = **new** ArrayList<radar>();

high = 0.0;

low = 1.0;

}

**public** **boolean** canMerge(radar i, radar j){

**return** Math.*pow*(i.radius+j.radius, 2)>=Math.*pow*(i.x-j.x, 2)+Math.*pow*(i.y-j.y,2);

}

**public** **boolean** merge(radar i){

**for**(radar rd: area){

**if**(canMerge(rd, i)){

high = Math.*max*(high, i.y+i.radius);

low = Math.*min*(low, i.y-i.radius);

area.add(i);

**return** **true**;

}

}

**return** **false**;

}

**public** **boolean** canPass(){

**return** high < 1.0 || low > 0.0;

}

}

**static** **class** radar{

**public** **double** x, y, radius;

**public** radar(**double** x, **double** y, **double** r){

**this**.x = x;

**this**.y = y;

radius = r;

}

}

# Group Record using Union-find

题目是手机上的通讯录，每条记录只有(name, number)这种pair,有些记录名字重复，有些记录号码重复，让我返回一个list<list>， // 将所有记录按人分组。比较tricky的点在于(ABC,123), (ABC, 456), (BCD, 456) // 三条记录，第一条和第三条也属于同一个人。要求时间复杂度尽量小</list

1.首先ufMap里面存<Record, Record>，初始化把所有entry过一遍，都存入，使它最初的root节点都是它自己

2. find部分。

1）首先找到这个record的root节点

2）然后把从这个record开始直到root节点所有的节点的root节点都更新成root 　　3）返回root

3. union部分，没什么特别的

1）对record1,record2找到root1,root2

2）如果root1 != root2，那么就设置其中一个是另外一个的root节点

1.建立一个nameMap<nameString, Record>和一个numMap<number, Record>

2.对于所有entry走一遍，

1) 如果nameMap里面已经存在当前entry的name，那么就说明要union；否则就把[nameString, entry]加入map

2）如果numMap里面已经存在当前entry的number，那么就说明要union；否则就把[number, entry]加入map

3. 所以此刻ufMap里面已经有已经对于name和number group之后的结果，现在就需要把结果保存下来.所以可以先建一个Map<String, List<Record>>整理好，再根据这个map得到List<List<Record>>

**import** java.util.\*;

**import** java.lang.\*;

**public** **class** GroupRecord {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<List<Record>> res = **new** ArrayList<List<Record>>();

ArrayList<Record> records = **new** ArrayList<>();

records.add(**new** Record("ABC","123"));

records.add(**new** Record("ABC","456"));

records.add(**new** Record("BCD","456"));

records.add(**new** Record("AD","342"));

records.add(**new** Record("AddD","342"));

records.add(**new** Record("AddD","56"));

records.add(**new** Record("cv","56"));

records.add(**new** Record("cv","11"));

records.add(**new** Record("AdD","11"));

records.add(**new** Record("App","1000"));

**long** startTime = System.*currentTimeMillis*();

**for**(**int** i = 0;i < 1000;i++)

res = *group*(records);

**long** endTime = System.*currentTimeMillis*();

**long** duration = (endTime - startTime);

System.***out***.println(duration);

}

**public** **static** List<List<Record>> group(List<Record> list){

HashMap<String, Record> nameMap = **new** HashMap<>();

HashMap<String, Record> numMap = **new** HashMap<>();

HashMap<Record, Record> roots = **new** HashMap<>();

HashMap<Record, Integer> size = **new** HashMap<>();

**for**(Record r: list){

roots.put(r,r);

size.put(r, 1);

}

**for**(Record r: list){

**if**(nameMap.containsKey(r.name))

*union*(r, nameMap.get(r.name), roots, size);

**else**

nameMap.put(r.name, r);

**if**(numMap.containsKey(r.number))

*union*(r, numMap.get(r.number),roots, size);

**else**

numMap.put(r.number, r);

}

HashMap<String, List<Record>> groupList = **new** HashMap<>();

**for**(Record r: list){

Record root = *find*(roots, r);

**if**(!groupList.containsKey(root.name))

groupList.put(root.name, **new** ArrayList<Record>());

groupList.get(root.name).add(r);

}

**return** **new** ArrayList<List<Record>>(groupList.values());

}

**//union with weighted and path compression**

**private** **static** **void** union(Record i, Record j, HashMap<Record, Record> roots, HashMap<Record, Integer> size){

Record rootI = *find*(roots, i);

Record rootJ = *find*(roots, j);

**if**(rootI != rootJ){

**if**(size.get(rootI) < size.get(rootJ)){

roots.put(rootI, rootJ);

size.put(rootI, size.get(rootI)+size.get(rootJ));

}

**else**{

roots.put(rootJ, rootI);

size.put(rootJ, size.get(rootI)+size.get(rootJ));

}

}

}

**private** **static** Record find(HashMap<Record, Record> roots, Record r){

Record root = r;

**while**(roots.get(root) != root){

roots.put(roots.get(root), roots.get(roots.get(root)));

root = roots.get(root);

}

**return** root;

}

}

**class** Record{

**public** String name;

**public** String number;

**public** Record(String n, String num){

name = n;

number = num;

}

**public** String toString(){

**return** "(" + name + "," + number + ")";

}

}

# Questions to ask

1. Recently Snap launched the new feature, group chat. I see each group will have group size limit 16. 16 means small group, why only allow small group and why is 16?
2. For ads, it seems the ad will continue play over unless I tapped.
3. Snapcash, the feature is cooperated with Square. So it looks like snapchat is client of Square, and every request and response is to and from Square. Why don’t Snap build itself payment/transfer money feature?
4. For discover feature, looks like I can’t choose the topic I am interested in.
5. Why snap. As being loyal snapchatter for 3 years, I am so impressed by fancy features have been launched already, geo filter, lenses and bitmoji. I have to say Snap is the most creative company I have seen so far. Snap represents young, passion and creation. By talking to my friend Miao and reading other news about Snap, I know there are thousands of talented engineers are developing the fancy features. I really would like to join the team and be creative and make my contribution.