Google Onsite Preparation

1. 汇率问题(Leetcode 399)

创建两个HashMap，一个存储parent关系，另外一个存储一个节点和parent之间的数学关系。
所以，在创建的时候，存入的代码是：

```
//e[i][0]/e[i][1] = value[i]
String p1 = find(root, dist, e[i][0]);
String p2 = find(root, dist, e[i][1]);
root.put(p1,p2);
dist.put(p1,dist.get(e[i][1])*values[i]/dist.get(e[i][0]));
```

在计算的时候的代码是：(需要提前判断一下这两个点是否share一个parent)

```
// 如果query中两个节点都在Union Tree内
result = dist.get(q[i][0])/dist.get(q[i][1]);
```

find函数更新、插入的算法是：

```
private String find(Map<String, String> root, Map<String, Double> dist, String s){
  if(!root.containsKey(s)){
    //插入新的节点
    root.put(s,s);
    dist.put(s,1.0);
    return s;
  }
  if(root.get(s).equals(s)){
    return s;
  }
  String lastParent = root.get(s);
  String p = find(root, dist, lastParent);
  root.put(s,p);
  dist.put(s,dist.get(lastParent)*dist.get(s));
  return p;
}
```

2. 人和自行车问题

a) 初级题目：

一个2D数组。'.'表示Road，'#'表示Building，'B'表示自行车。

. . . . . #

. . E . . #

.

. B . . . .

. . . . . B

寻找最近的自行车。

```
// BFS
public Pair BFS(char[][] map, Pair employee){
  boolean[][] visited = new boolean[map.length][map[0].length];
  Queue<Pair> queue = new Queue<>();
  queue.add(employee);
  visited[employee.x][employee.y] = true;
  while(!queue.isEmpty()){
    Pair cur = queue.poll();
    if(map[cur.x][cur.y] == '#'){
      continue;
    }
    if(map[cur.x][cur.y] == 'B'){
      return cur;
    }
    if(cur.x + 1 < map.length && !visited[cur.x+1][cur.y]){
      queue.add(new Pair(cur.x+1,cur.y));
      visited[cur.x+1][cur.y] = true;
    }
    if(cur.x - 1 >= 0 && && !visited[cur.x-1][cur.y]){
      queue.add(new Pair(cur.x-1,cur.y));
      visited[cur.x-1][cur.y] = true;
    }
    if(cur.y - 1 >= 0 && && !visited[cur.x][cur.y-1]){
      queue.add(new Pair(cur.x,cur.y-1));
      visited[cur.x][cur.y-1] = true;
    }
    if(cur.y + 1 < map[0].length && && !visited[cur.x][cur.y+1]){
      queue.add(new Pair(cur.x,cur.y+1));
      visited[cur.x][cur.y+1] = true;
    }
  }
  return null;
}
```

b) follow up: 给每个人匹配一个自行车

目前没有找到原题，但是根据网上大概做法是：

每个人进行一次BFS返回所有Employee-Bicycle对，然后放到一个minheap中，逐个弹出。(用Set)记录下人和车的状态就好。

3. Redundant Connection I && II as follow up

a) 无向图

只需要写并查集，每次插入的时候检测一下他们是否有共同的parent，如果有，就return那条边，没有，就设置一个点为另外一个点的parent。

```java
class Solution {
    public int[] findRedundantConnection(int[][] edges) {
        int [] parent = new int[edges.length+1];
        for(int i = 0; i < parent.length;i++){
            parent[i] = i;
        }

        for(int [] e: edges){
            if(find(parent, e[0]) == find(parent,e[1])){
                return e;
            }else{
                parent[find(parent,e[0])] = find(parent, e[1]);;
            }
        }
        return new int[2];
    }

    private int find(int [] parent, int node){
        if(parent[node] != node){
            parent[node] = find(parent, parent[node]);
        }
        return parent[node];
    }
}
```

b) 有向图

也是用并查集做，检测方式有些区别。先(parent[e[i][1]]!=0)判断第二个点是否有两个同时指向它,存储一下当前的边和另外一个指向它的边，将当前边设置为-1。接着按照上面的方法来通过parent是否相等判断是否有环。然后通过判断选择返回值。

```java
class Solution {
    public int[] findRedundantDirectedConnection(int[][] edges) {
        int[] ans1 = new int[2];
        int[] ans2 = new int[2];
        int [] parent = new int[edges.length+1];
        for(int i = 0; i < edges.length;i++){
            if(parent[edges[i][1]]!=0){
                ans1[0] = edges[i][0];
                ans1[1] = edges[i][1];
                ans2[0] = parent[edges[i][1]];
                ans2[1] = edges[i][1];
                //ans2 = new int[2]{,edges[i][1]};
                edges[i][0] = edges[i][1] = -1;
            }else{
                parent[edges[i][1]] = edges[i][0];
            }
        }
        for(int i = 0; i <parent.length;i++){
            parent[i] = i;
        }
        for(int i = 0; i <edges.length;i++){
            if(edges[i][0] < 0 || edges[i][1] < 0 ){
                continue;
            }
            int p1 = find(parent,edges[i][0]);
            int p2 = find(parent,edges[i][1]);
            if(p1 == p2){
                return (ans1[0]!= 0 && ans1[1]!= 0)?ans2:edges[i];
            }else{
                parent[edges[i][1]] = edges[i][0];
            }
        }
        return ans1;
    }

    private int find(int [] parent, int node){
        if(parent[node] != node){
            return find(parent,parent[node]);
        }
        return node;
    }
}
```

4. Unique Path I && II as follow up(Leetcode 62 & Leetcode 63)

a) 无障碍

标准的DP，一层一层扫描。空间复杂度为O(mn)，进一步优化可以到O(n)。
Base case: M[i][0] = M[0][i] = 1, M[i][j] = M[i-1][j] + M[j-1][1]

```java
// 用for-for loop更简单，懒得写了
class Solution {
  public int uniquePaths(int m, int n) {
    if(m==0 && n==0){
      return 0;
    }
    if(m==1 || n==1){
      return 1;
    }
    int[][] M = new int[m][n];
    //index都是从0开始的
    return helper(m-1,n-1,M);
  }

  private int helper(int m, int n,int[][] M){
    if(m == 0 || n==0){
      return 1;
    }
    if(M[m][n]!=0){
      return M[m][n];
    }
    int paths = helper(m-1,n,M) + helper(m,n-1,M);
    M[m][n] = paths;
    return paths;
  }
}
```

b) 有障碍

稍做改变的DP题。
标准的DP，一层一层扫描。空间复杂度为O(mn)，进一步优化可以到O(n)。
Base case: M[i][0] = M[0][i] = 1, M[i][j] = M[i-1][j] + M[j-1][1]。需要多做一步判断，如果该点为墙，则将该点的值设置为0。

```java
// 用for-for loop更简单，懒得写了
class Solution {
  public int uniquePathsWithObstacles(int[][] obstacleGrid) {
    int m = obstacleGrid.length, n = obstacleGrid[0].length;
    int[][] temp = new int[m][n];
    return solve(obstacleGrid, m - 1, n - 1, temp);
  }
  public int solve(int[][] ob, int m, int n, int[][] temp) {
    if(ob[m][n] == 1) return 0;
    if(temp[m][n] > 0) return temp[m][n];
    if(m != 0 && n != 0) {
      temp[m][n] = solve(ob, m, n - 1, temp) + solve(ob, m - 1, n, temp);
      return temp[m][n];
    }else if(m == 0) {
      for(int i = n; i >= 0; i--) if(ob[0][i] == 1) return 0;
      return 1;
    }else {
      for(int i = m; i >= 0; i--) if(ob[i][0] == 1) return 0;
      return 1;
    }
  }
}
```

5. Guess the Word(Leetcode 843)

先是随机选一个，然后对现有的剩余字符串进行匹配，在匹配度等于该字符串和ans匹配度一样时候，存入List。

```java
public void findSecretWord(String[] wordlist, Master master) {
  for (int i = 0, x = 0; i < 10 && x < 6; ++i) {
    String guess = wordlist[new Random().nextInt(wordlist.length)];
    x = master.guess(guess);
    List<String> wordlist2 = new ArrayList<>();
    for (String w : wordlist){
      if (match(guess, w) == x){
        wordlist2.add(w);
      }
    }
    wordlist = wordlist2.toArray(new String[wordlist2.size()]);
  }

public int match(String a, String b) {
  int matches = 0;
  for (int i = 0; i < a.length(); ++i){
    if (a.charAt(i) == b.charAt(i)){
      matches ++;
    }
  }
  return matches;
}
```

可以通过剪枝来缩小初始的搜索范围。对所有字符串进行两两匹配，如果得0就存入hashmap。最后选取匹配度为0的最少的字符串，然后在进行以上操作。

```java
public void findSecretWord(String[] wordlist, Master master) {
  for (int i = 0, x = 0; i < 10 && x < 6; ++i) {
    HashMap<String, Integer> count = new HashMap<>();
    for (String w1 : wordlist){
      for (String w2 : wordlist){
        if (match(w1, w2) == 0){
          count.put(w1, count.getOrDefault(w1 , 0) + 1);
        }
      }
    }
    Pair<String, Integer> minimax = new Pair<>("", 1000);
    for (String w : wordlist){
      if (count.getOrDefault(w, 0) < minimax.getValue()){
        minimax = new Pair<>(w, count.getOrDefault(w, 0));
      }
    }
    x = master.guess(minimax.getKey());
    List<String> wordlist2 = new ArrayList<String>();
    for (String w : wordlist){
      if (match(minimax.getKey(), w) == x){
        wordlist2.add(w);
      }
    }
    wordlist = wordlist2.toArray(new String[0]);
  }
}
```

6. Find and Replace Pattern

Input: words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"
Output: ["mee","aqq"]
Explanation: "mee" matches the pattern because there is a permutation {a -> m, b -> e, ...}.

重点是match方法，可以通过两个HashMap比较，这个比较简单，就是插入的时候判断一下两个hashmap中的值是否互相等于。
用一个HashMap的做法就是每次插入的时候检测key对应的value是否相等，最后在check一下value是否有重复。

```java
public boolean match(String word, String pattern) {
  Map<Character, Character> M = new HashMap();
  for (int i = 0; i < word.length(); ++i) {
    char w = word.charAt(i);
    char p = pattern.charAt(i);
    if (!M.containsKey(w)){
      M.put(w, p);
    }
    if (M.get(w) != p){
      return false;
    }
  }
  boolean[] seen = new boolean[26];
  for (char p: M.values()) {
    if (seen[p - 'a']){
      return false;
    }
    seen[p - 'a'] = true;
  }
  return true;
}
```

7. Robot Room Cleaner(Leetcode 489)

机器人的API有如下方法：turnleft(), turnright(),move()。这道题的重点是在做backtracking的时候，要注意机器人的方向。初始设定为向上。

```
public int[][] directions = {{-1,0},{0,1},{1,0},{0,-1}};
private swipe(Robot r, Set<Node> visited, int currentDirection, int row, int col){
  Node node = new Node(row, col);
  visited.add(node);
  robot.clean();
  for(int i = 0; i<4;i++){
    int direction = (currentDirection + i) % 4;
    int[] next = directions[i];
    int nextRow = row + next[0];
    int nextCol = col + next[1];
    node = new Node(nextRow, nextCol);
    if(!visited.contains(node) && r.move()){
      swipe(r, visited, direction, nextRow, nextCol);
      r.turnLeft();//结束上次的扫描，回头
      r.turnLeft();
      r.move();
      r.turnLeft();//恢复到上个循环的下次扫描
    }else{
      r.turnRight();//直接跳到下次扫描
    }
  }
}
```

8. 考试座位安排(Leetcode 855)

这个题目的目的是最大化每个两个相邻座位间的距离。
思路是用TreeSet。因为Treeset中的元素是排好序的。

```
Set<Integer> set = new TreeSet<>();
int N;
public int seat(){
  int student = 0;
  if(set.size()>0){
    int distance = set.first();
    Integer prev = null;
    for(Integer s: set){
      if(prev!=null){
        dist = (s - prev)/2;
        if(dist > distance){
          distance = dist;
```

```
            student = prev + dist;
          }
        }
        prev = s;
      }
      if(N-1 - student.last() > distance){
        student = N - 1;
      }
    }
    set.add(student);
    return student;
  }
```

9. 设计有Expiration的HashMap(Leetcode 146)

大概的思路是用双向链表、一个HasMap。设计双向链表的Node里面添加一个起始时间和一个Duration。每次读取的时候检查一下时间，如果超时，就删除掉，没有超时就返回。其实就是一个LRU。

10. Car Fleet(Leetcode 853)

这个题的思路就是按照他们的所在位置排序，如果后面的比前面的先到（时间更短，那就是一个Fleet）。

```
class Car {
  int position;
  double time;
  Car(int p, double t) {
    position = p;
    time = t;
  }
}
public int carFleet(int target, int[] position, int[] speed){
  int N = postion.length;
  if(N == 0){
    return 0;
  }
  Car[] cars = new Cars[];
  for(int i = 0; i < N; i++){
    cars[i] = new Car(position[i], (double)(target-position[i])/speed[i]);
  }
  Arrays.sort(cars, new Comparator<Cars>(){
    @Override
    public int compare(Car a, Car b){
      return Integer.compare(a.position, b.position);
    }
  });
  int ans = 0;
  int t = N;
  while(--t > 0){
    if(cars[t].time < cars[t-1].time){
      ans++;
    }else{
      car[t-1] = car[t];
    }
  }
  return ans+1;
}
```

11. Minimum Cost to Hire K Workers(Leetcode 857)

这个题目要关注员工的Price/Quality的Ratio，对这个Ratio进行从小到大的排序。越大的Ratio对于大工资来讲是不好的，所以在选择k个员工的时候，工资越大且Ratio越大的不受欢迎。

```
class Worker{
  int wage;
  int quality;
  public Worker(int q, int w) {
    quality = q;
    wage = w;
  }

  public double ratio() {
    return (double) wage / quality;
  }

  public int compareTo(Worker other) {
    return Double.compare(ratio(), other.ratio());
  }
}

public double minCost(int[] wage, int[] quality, int k){
  Worker[] workers = new Worker[];
  for(int i = 0; i < wage.length; i++){
    workers[i] = new Worker(quality[i], wage[i]);
  }
  Arrays.sort(workers);
  ProrityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
  int sumOfQuality = 0;
  double ans = Double.MAX_VALUE;
  for(Worker worker: workers){
    double ratio = worker.ratio();
    maxHeap.offer(worker.wage);
    sumOfQuality += worker.quality;
    if(maxHeap.size() > k){
      sumOfQuality -= maxHeap.poll();
    }
    if(maxHeap.size() == k){
      ans = Math.min(ans,ratio * sumOfQuality);
    }
  }
  return ans;
}
```

12. Split Array into Consecutive Subsequences(Leetcode 659)

这道题问你能不能valid split， 不split就满足条件，也算个Split。
You are given an integer array sorted in ascending order (may contain duplicates), you need to split them into several subsequences, where each subsequences consist of at least 3 consecutive integers. Return whether you can make such a split.

这道题用Greedy算法。遇到一个数字，考虑从它开始能不能有一个Valid字串。从左向右扫描，一个数字被加到当前字串后面是比开启一个新的字串要好的。因为，如果我们能从x开启一个新串，是等同于我们把它接到之前的子串的。所以，对于一个数字x，我们检测有没有结束在它前一个的串，如果有就加上去。没有，就开启一个新的字串。

```
class Counter extends HashMap<Integer, Integer> {
  public int get(int k) {
    return containsKey(k) ? super.get(k) : 0;
  }

  public void add(int k, int v) {
    put(k, get(k) + v);
  }
}

private boolean checkSplit(int nums){
  Counter count = new Counter();
  Counter tail = new Counter();
  for(int x: nums){
    count.add(x,1);
  }
  for(int x: nums){
    if(count.get(x) == 0){
      continue;
    }else if(tail.get(x) > 0){
      tail.add(x,-1);
      tail.add(x+1,1);
    }else if(count.get(x+1) > 0 && count.get(x+2) > 0){
      count.add(x+1,-1);
      count.add(x+2,-1);
      tail.add(x+3,1);
    }else{
      return false;
    }
    count.add(x,-1);
  }
  return true;
}
```

13. Backspace String Compare(Leetcode 844)

思路是从后向前分别扫描，直到跳过的字串数量=0。

```
public boolean backspaceCompare(String S, String T) {
  int i = S.length() - 1, j = T.length() - 1;
  int skipS = 0, skipT = 0;
  while(i >= 0 || j >= 0){
    while(i >= 0){
      if(S.charAt(i) == '#'){
        skipS ++;
        i--;
      }else if(skipS > 0){
        skipS --;
        i--;
      }else{
        break;
      }
    }
    while(j >= 0){
      if(T.charAt(j) == '#'){
        skipT++;
        j--;
      }else if(skipT > 0){
        skipT --;
        j--;
      }else{
        break;
      }
    }
    if(i >= 0 && j >= 0 && S.charAt(i) != T.charAt(j)){
      return false;
    }
    if(i >= 0 ^ j >=0){
      return false;
    }
    i--;
    j--;
  }
  return true;
}
```

14. Number Of Corner Rectangles(Leetcode 750)

这个问题，就是逐行遍历，然后把所有在一行的点两两编码，存到HashMap中。之后比对HashMap，再加上频率就好。

```java
public int countCornerRectangles(int[][] grid) {
  Map<Integer, Integer> map = new HashMap<>();
  int ans = 0;
  for(int[] row: grid){
    for(int i = 0; i < row.length-1; i++){
      if(row[i] == 1){
        for(int j = i + 1; j < row.length; j++){
          if(row[j] == 1){
            int hash = 200 * i + j;
            if(map.containsKey(hash)){
              ans+=map.get(hash);
              map.put(hash, map.get(hash)+1);
            }else{
              map.put(hash,1);
            }
          }
        }
      }
    }
  }
  return ans;
}
```

还可以更近一步优化。思路是，对于点少的行继续按照上面的方法做，对于点多的行，通过其他点少的行和这个行找重叠，square = found*(found-1)/2。

针对这种密集的行（记该行为r，一种改进措施是不再遍历其列表中每一对元素，而是直接寻找表中每一行（如行a）有多少个1，它们在r行对应列上也是1。假设有f个，那么行a与行f所组成的矩形就有(f - 1) * f / 2个。
我们可以用把行r的列表转换为集合Set，然后只需线性遍历每一行，计算出f及其产生的矩形数。
要注意如果如果行r和行a都是密集行，那么行a只有在行r的下方有效。否则会重复计算，也即遍历到行r计算了一次，遍历到行a又计算了一次。

```java
public int countCornerRectangles(int[][] grid) {
  List<List<Integer>> rows = new ArrayList();
  int N = 0;
  for (int r = 0; r < grid.length; ++r) {
    rows.add(new ArrayList());
    for (int c = 0; c < grid[r].length; ++c){
      if (grid[r][c] == 1) {
        rows.get(r).add(c);
```

```
            N++;
          }
        }
      }
      int sqrtN = (int) Math.sqrt(N);
      int ans = 0;
      Map<Integer, Integer> count = new HashMap();
      for (int r = 0; r < grid.length; ++r) {
        if (rows.get(r).size() >= sqrtN) {
          Set<Integer> target = new HashSet(rows.get(r));
          for (int r2 = 0; r2 < grid.length; ++r2) {
            if (r2 <= r && rows.get(r2).size() >= sqrtN)
              continue;
            int found = 0;
            for (int c2: rows.get(r2))
              if (target.contains(c2))
                found++;
            ans += found * (found - 1) / 2;
          }
        } else {
          for (int i1 = 0; i1 < rows.get(r).size(); ++i1) {
            int c1 = rows.get(r).get(i1);
            for (int i2 = i1 + 1; i2 < rows.get(r).size(); ++i2) {
              int c2 = rows.get(r).get(i2);
              int ct = count.getOrDefault(200*c1 + c2, 0);
              ans += ct;
              count.put(200*c1 + c2, ct + 1);
            }
          }
        }
      }
      return ans;
    }
```

15. Bus Routes(Leetcode 815)

这道题的思路是BFS。(如果队列里存储的是站的序号的话，会超时，因为是O(n^3))。

```
class Solution {
    public int numBusesToDestination(int[][] routes, int S, int T) {
        if(S == T){
            return 0;
        }
        Map<Integer, List<Integer>> map = new HashMap<>();
        for(int i = 0; i < routes.length; i++){
            for(int j = 0; j < routes[i].length;j++){
                if(!map.containsKey(routes[i][j])){
                    map.put(routes[i][j], new ArrayList<>());
                }
```

```java
                map.get(routes[i][j]).add(i);
            }
        }

        Set<Integer> visitedRoute = new HashSet<>();
        Set<Integer> visitedStop = new HashSet<>();
        Queue<Integer> queue = new LinkedList<>();
        int transfer = 0;
        visitedStop.add(S);
        for(Integer i: map.get(S)){
            queue.add(i);
            visitedRoute.add(i);
        }
        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i = 0; i < size; i++){
                int route = queue.poll();
                for(int j = 0; j < routes[route].length; j++){
                    int stop = routes[route][j];
                    if(visitedStop.contains(stop)){
                        continue;
                    }
                    if(stop == T){
                        return transfer+1;
                    }
                    for(Integer k: map.get(stop)){
                        if(visitedRoute.contains(k)){
                            continue;
                        }
                        queue.add(k);
                        visitedRoute.add(k);
                    }
                }
            }
            transfer ++;
        }
        return -1;
    }
}
```

所以应该在里面存储的是在这在这个线路上能换乘的线路。所以是<线路1 - [线路2、线路4]>，<线路2,[线路1]>，<线路4，[线路1]>。就能简化运算。

```java
import java.awt.Point;

class Solution {
  public int numBusesToDestination(int[][] routes, int S, int T) {
    if (S==T) return 0;
    int N = routes.length;

    List<List<Integer>> graph = new ArrayList();
    for (int i = 0; i < N; ++i) {
      Arrays.sort(routes[i]);
      graph.add(new ArrayList());
    }
    Set<Integer> seen = new HashSet();
    Set<Integer> targets = new HashSet();
    Queue<Point> queue = new ArrayDeque();

    // Build the graph.  Two buses are connected if
    // they share at least one bus stop.
    for (int i = 0; i < N; ++i)
      for (int j = i+1; j < N; ++j)
        if (intersect(routes[i], routes[j])) {
          graph.get(i).add(j);
          graph.get(j).add(i);
        }

    // Initialize seen, queue, targets.
    // seen represents whether a node has ever been enqueued to queue.
    // queue handles our breadth first search.
    // targets is the set of goal states we have.
    for (int i = 0; i < N; ++i) {
      if (Arrays.binarySearch(routes[i], S) >= 0) {
        seen.add(i);
        queue.offer(new Point(i, 0));
      }
      if (Arrays.binarySearch(routes[i], T) >= 0)
        targets.add(i);
    }

    while (!queue.isEmpty()) {
      Point info = queue.poll();
      int node = info.x, depth = info.y;
      if (targets.contains(node)) return depth+1;
      for (Integer nei: graph.get(node)) {
        if (!seen.contains(nei)) {
          seen.add(nei);
          queue.offer(new Point(nei, depth+1));
        }
      }
    }
    return -1;
  }

  public boolean intersect(int[] A, int[] B) {
    int i = 0, j = 0;
    while (i < A.length && j < B.length) {
      if (A[i] == B[j]) return true;
      if (A[i] < B[j]) i++; else j++;
    }
    return false;
  }
}
```

16. Tree Isomorphism Problem

树的变形。有的节点反转了，有的没有。

```
public boolean isIsomorphic(Node n1, Node n2){
  if(n1 == null && n2 == null){
    return true;
  }
  if(n1 == null || n2 == null){
    return false;
  }
  if(n1.val != n2.val){
    return false;
  }
  return isIsomorphic(n1.left,n2.right) && isIsomorphic(n1.right, n2.left) || isIsomorphic(n1.left, n2.left) && isIsomorphic(n1.right, n2.right);
}
```

17. 围棋判断是否被包围

思路是DFS，一定要问清楚各种条件以及各种算包围的方式。

从一个符合条件的点开始做DFS/BFS，遇到边界返回false，在不允许留白的情况下，可以直接遇到白色或者边界返回false，即没有被包围。

```
//待解决
```

18. Random Generate a Maze(Laicode 218)

这个题目的思路依旧是DFS，但是区别是要两步两步地走，为了防止产生闭合回路，要两步两步地走。

```
public class Solution {
  public int[][] maze(int n) {
    // Write your solution here.
    int[][] maze = new int[n][n];
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
        if(i == 0 && j == 0){
          maze[i][j] = 0;
        }else{
          maze[i][j] = 1;
        }
        }
    }
    generate(maze, 0, 0);
    return maze;
  }

  private void generate(int[][] maze, int x, int y){
    int [] dirX = {1,0,-1,0};
    int [] dirY = {0,1,0,-1};
    shuffle(dirX,dirY);
    for(int i = 0; i < dirX.length; i++){
      int dx = dirX[i];
      int dy = dirY[i];
      if(validWall(maze, x + 2*dx, y +2*dy)){
        maze[x+dx][y+dy] = 0;
        maze[x+2*dx][y+2*dy] = 0;
        generate(maze, x+2*dx, y+2*dy);
      }
    }
  }

  private boolean validWall(int [][] maze, int x, int y){
    if(x >= 0 && x < maze.length && y >= 0 && y < maze.length && maze[x][y] == 1){
      return true;
    }else{
      return false;
    }
  }

  private void shuffle(int [] dirX, int [] dirY){
    for(int i = 0; i < dirX.length; i++){
      int rand = (int)(Math.random()*(dirX.length - i));
      swap(dirX,i,i+rand);
      swap(dirY,i,i+rand);
    }
  }

  private void swap(int [] num, int i, int j){
    int tmp = num[i];
    num[i] = num[j];
    num[j] = tmp;
  }
}
```

19. Falling Bricks When Hit(Leetcode 859)

最简单的办法就是，在被击碎的砖块的四个相邻砖块进行连通块的遍历。如果能够遍历到最上面一行，则返回false（不会掉落），如果不能够遍历到最上面的一行，则返回true（即掉落）。在运行的时候，加入一个大小为1的数组进行答案的存储。

还有一种办法是并查集。思路是：将所有hit的部分敲掉，然后再从后向前插入被敲掉的砖块，新建立起来的连通分量就是数量就是会掉落的砖的数量。

```java
class Solution {
    public static int[] hitBricks(int[][] grid, int[][] hits) {
        int m = grid.length;
        int n = grid[0].length;
        int[][] mat = new int[m][n];
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                mat[i][j] = grid[i][j];
            }
        }
        for (int[] hit : hits) {
            mat[hit[0]][hit[1]] = 0;
        }
        UnionFind uf = new UnionFind(m * n + 1);
        for (int i = 0; i < n; i++) {
            if (mat[0][i] == 1) {
                uf.union(i, m * n);
            }
        }

        for (int i = 1; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (mat[i][j] == 1) {
                    if (mat[i - 1][j] == 1) {
                        uf.union(i * n + j, (i - 1) * n + j);
                    }
                    if (j - 1 >= 0 && mat[i][j - 1] == 1) {
                        uf.union(i * n + j, i * n + j - 1);
                    }
                }
            }
        }
        int[] ans = new int[hits.length];
        for (int i = hits.length - 1; i >= 0; i--) {
            int x = hits[i][0];
            int y = hits[i][1];
            if (grid[x][y] == 1) {
                int prev = uf.getTopRank(m * n);
                mat[x][y] = 1;
                if (x + 1 < m && mat[x + 1][y] == 1) {
                    uf.union(x * n + y, (x + 1) * n + y);
                }
                if (x - 1 >= 0 && mat[x - 1][y] == 1) {
                    uf.union(x * n + y, (x - 1) * n + y);
                }
                if (y + 1 < n && mat[x][y + 1] == 1) {
                    uf.union(x * n + y, x * n + y + 1);
                }
                if (y - 1 >= 0 && mat[x][y - 1] == 1) {
                    uf.union(x * n + y, x * n + y - 1);
                }
                if (x == 0) {
                    uf.union(x * n + y, m * n);
                }
                int cur = uf.getTopRank(m * n);
                ans[i] = Math.max(cur - prev-1,0);
            } else {
                ans[i] = 0;
            }
        }
        return ans;
    }

    private static void reverse(int[] ans) {
        int i = 0;
        int j = ans.length - 1;
        while (i < j) {
            int tmp = ans[i];
            ans[i] = ans[j];
            ans[j] = tmp;
            i++;
            j--;
        }
    }
}

class UnionFind {
    int[] parent;

    public UnionFind(int N) {
        parent = new int[N];
        Arrays.fill(parent, -1);
    }

    public int find(int i) {
        if (parent[i] < 0) {
            return i;
        } else {
            return parent[i] = find(parent[i]);
        }
    }

    public void union(int i, int j) {
        int parentI = find(i);
        int parentJ = find(j);
        if (parentI == parentJ) {
            return;
        }
        int size = parent[parentI] + parent[parentJ];
        if (parent[parentI] < parent[parentJ]) {
            parent[parentJ] = parentI;
            parent[parentI] = size;
        } else {
            parent[parentI] = parentJ;
            parent[parentJ] = size;
        }
    }

    public int getTopRank(int i) {
        return -parent[find(i)];
    }
}
```

20. Meeting Rooms(Leetcode 253)

这个题的思路是PriorityQueue，Heap里面的是这个会议的结束时间。外面每个都循环比较外面的interval的start和peek的end。如果start > end，则不冲突。

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */

class Solution {
  public int minMeetingRooms(Interval[] intervals) {
    if(intervals == null || intervals.length == 0){
      return 0;
    }
    PriorityQueue<Integer> heap = new PriorityQueue<>();
    Arrays.sort(intervals, new Comparator<Interval>(){
      @Override
      public int compare(Interval a, Interval b){
        return a.start - b.start;
      }
    });
    heap.offer(intervals[0].end);
    for(int i = 1; i < intervals.length; i++){
      if(intervals[i].start > heap.peek(){
        heap.poll();
      }
      heap.offer(intervals[i].end);
    }
    return heap.size();
  }
}
```

21. Implement Magic Dictionary(Leetcode 676)

思路1: 对字典里面每个词做one edit distance，如果满足，就直接返回。但是缺点在于，每次比较是O(n),有k个单词，那就是O(n*k)。如果词库非常大，复杂度就非常高。

思路2: 在存储的时候，生成所有可能的词汇存在一个表里面，每次查询生成所有可能的，查询他是否在词汇表中。这道题一定要注意Corner Case。(比如：「hello，hallo，leetcode」，hello)

```java
package implementMagicDictionary;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class Solution {
    Set<String> words;
    Map<String, Integer> count;

    public Solution() {
        words = new HashSet();
        count = new HashMap();
    }

    private ArrayList<String> generalizedNeighbors(String word) {
        ArrayList<String> ans = new ArrayList();
        char[] ca = word.toCharArray();
        for (int i = 0; i < word.length(); ++i) {
            char letter = ca[i];
            ca[i] = '*';
            String magic = new String(ca);
            ans.add(magic);
            ca[i] = letter;
        }
        return ans;
    }

    public void buildDict(String[] words) {
        for (String word : words) {
            this.words.add(word);
            for (String nei : generalizedNeighbors(word)) {
                count.put(nei, count.getOrDefault(nei, 0) + 1);
            }
        }
    }

    public boolean search(String word) {
        for (String nei : generalizedNeighbors(word)) {
            int c = count.getOrDefault(nei, 0);
            if (c > 1 || c == 1 && !words.contains(word))
                return true;
        }
        return false;
    }
}
```