前言

省略的代码：

```
import java.util.*;
```

```
public class Solution {
}
```

```
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        while (in.hasNext()) {
        }
    }
}
```

1. 小米-小米Git

- 重建多叉树
- 使用 LCA

```
private class TreeNode {
    int id;
    List<TreeNode> childs = new ArrayList<>();

    TreeNode(int id) {
        this.id = id;
    }
}

public int getSplitNode(String[] matrix, int indexA, int indexB) {
    int n = matrix.length;
    boolean[][] linked = new boolean[n][n]; // 重建邻接矩阵
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            linked[i][j] = matrix[i].charAt(j) == '1';
        }
    }
    TreeNode tree = constructTree(linked, 0);
    TreeNode ancestor = LCA(tree, new TreeNode(indexA), new TreeNode(indexB));
    return ancestor.id;
}

private TreeNode constructTree(boolean[][] linked, int root) {
    TreeNode tree = new TreeNode(root);
    for (int i = 0; i < linked[root].length; i++) {
        if (linked[root][i]) {
            linked[i][root] = false; //  因为题目给的邻接矩阵是双向的，在这里需要把它转为单向的
            tree.childs.add(constructTree(links, i));
        }
    }
    return tree;
}

private TreeNode LCA(TreeNode root, TreeNode p, TreeNode q) {
    if (root == null || root.id == p.id || root.id == q.id) return root;
    TreeNode ancestor = null;
    int cnt = 0;
    for (int i = 0; i < root.childs.size(); i++) {
        TreeNode tmp = LCA(root.childs.get(i), p, q);
        if (tmp != null) {
            ancestor = tmp;
            cnt++;
        }
    }
    return cnt == 2 ? root : ancestor;
}
```

2. 小米-懂二进制

对两个数进行异或，结果的二进制表示为 1 的那一位就是两个数不同的位。

```
public int countBitDiff(int m, int n) {
    return Integer.bitCount(m ^ n);
}
```

3. 小米-中国牛市

背包问题，可以设一个大小为 2 的背包。

状态转移方程如下：

```
dp[i, j] = max(dp[i, j-1], prices[j] - prices[jj] + dp[i-1, jj]) { jj in range of [0, j-1] } =
max(dp[i, j-1], prices[j] + max(dp[i-1, jj] - prices[jj]))
```

```
public int calculateMax(int[] prices) {
    int n = prices.length;
    int[][] dp = new int[3][n];
    for (int i = 1; i <= 2; i++) {
        int localMax = dp[i - 1][0] - prices[0];
        for (int j = 1; j < n; j++) {
            dp[i][j] = Math.max(dp[i][j - 1], prices[j] + localMax);
            localMax = Math.max(localMax, dp[i - 1][j] - prices[j]);
        }
    }
    return dp[2][n - 1];
}
```

4. 微软-LUCKY STRING

- 斐波那契数列可以预计算；
- 从头到尾遍历字符串的过程，每一轮循环都使用一个 Set 来保存从 i 到 j 出现的字符，并且 Set 保证了字符都不同，因此 Set 的大小就是不同字符的个数。

```
Set<Integer> fibSet = new HashSet<>(Arrays.asList(1, 2, 3, 5, 8, 13, 21, 34, 55, 89));
Scanner in = new Scanner(System.in);
String str = in.nextLine();
int n = str.length();
Set<String> ret = new HashSet<>();
for (int i = 0; i < n; i++) {
    Set<Character> set = new HashSet<>();
    for (int j = i; j < n; j++) {
        set.add(str.charAt(j));
        int cnt = set.size();
        if (fibSet.contains(cnt)) {
            ret.add(str.substring(i, j + 1));
        }
    }
}
```

```
String[] arr = ret.toArray(new String[ret.size()]);
Arrays.sort(arr);
for (String s : arr) {
    System.out.println(s);
}
```

5. 微软-Numeric Keypad

```
private static int[][] canReach = {
        {1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 0
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, // 1
        {1, 0, 1, 1, 0, 1, 1, 0, 1, 1}, // 2
        {0, 0, 0, 1, 0, 0, 1, 0, 0, 1}, // 3
        {1, 0, 0, 0, 1, 1, 1, 1, 1, 1}, // 4
        {1, 0, 0, 0, 0, 1, 1, 0, 1, 1}, // 5
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1}, // 6
        {1, 0, 0, 0, 0, 0, 0, 1, 1, 1}, // 7
        {1, 0, 0, 0, 0, 0, 0, 0, 1, 1}, // 8
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 1}  // 9
};

private static boolean isLegal(char[] chars, int idx) {
    if (idx >= chars.length || idx < 0) return true;
    int cur = chars[idx] - '0';
    int next = chars[idx + 1] - '0';
    return canReach[cur][next] == 1;
}

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int T = Integer.valueOf(in.nextLine());
    for (int i = 0; i < T; i++) {
        String line = in.nextLine();
        char[] chars = line.toCharArray();
        for (int j = 0; j < chars.length - 1; j++) {
            while (!isLegal(chars, j)) {
                if (--chars[j + 1] < '0') {
                    chars[j--]--;
                }
                for (int k = j + 2; k < chars.length; k++) {
                    chars[k] = '9';
                }
            }
        }
        System.out.println(new String(chars));
    }
}
```

6. 微软-Spring Outing

下面以 N = 3，K = 4 来进行讨论。

初始时，令第 0 个地方成为待定地点，也就是呆在家里。

从第 4 个地点开始投票，每个人只需要比较第 4 个地方和第 0 个地方的优先级，里，如果超过半数的人选择了第 4 个地方，那么更新第 4 个地方成为待定地点。

从后往前不断重复以上步骤，不断更新待定地点，直到所有地方都已经投票。

上面的讨论中，先令第 0 个地点成为待定地点，是因为这样的话第 4 个地点就只需要和这个地点进行比较，而不用考虑其它情况。如果最开始先令第 1 个地点成为待定地点，那么在对第 2 个地点进行投票时，每个人不仅要考虑第 2 个地点与第 1 个地点的优先级，也要考虑与其后投票地点的优先级。

```
int N = in.nextInt();
int K = in.nextInt();
int[][] votes = new int[N][K + 1];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < K + 1; j++) {
        int place = in.nextInt();
        votes[i][place] = j;
    }
}
int ret = 0;
for (int place = K; place > 0; place--) {
    int cnt = 0;
    for (int i = 0; i < N; i++) {
        if (votes[i][place] < votes[i][ret]) {
            cnt++;
        }
    }
    if (cnt > N / 2) {
        ret = place;
    }
}
System.out.println(ret == 0 ? "otaku" : ret);
```

7. 微软-S-expression

8. 华为-最高分是多少

```
int N = in.nextInt();
int M = in.nextInt();
int[] scores = new int[N];
for (int i = 0; i < N; i++) {
    scores[i] = in.nextInt();
}
for (int i = 0; i < M; i++) {
    String str = in.next();
    if (str.equals("U")) {
        int id = in.nextInt() - 1;
        int newScore = in.nextInt();
```

```
            scores[id] = newScore;
        } else {
            int idBegin = in.nextInt() - 1;
            int idEnd = in.nextInt() - 1;
            int ret = 0;
            if (idBegin > idEnd) {
                int t = idBegin;
                idBegin = idEnd;
                idEnd = t;
            }
            for (int j = idBegin; j <= idEnd; j++) {
                ret = Math.max(ret, scores[j]);
            }
            System.out.println(ret);
        }
    }
```

9. 华为-简单错误记录

```
HashMap<String, Integer> map = new LinkedHashMap<>();
while (in.hasNextLine()) {
    String s = in.nextLine();
    String key = s.substring(s.lastIndexOf('\\') + 1);
    map.put(key, map.containsKey(key) ? map.get(key) + 1 : 1);
}
List<Map.Entry<String, Integer>> list = new LinkedList<>(map.entrySet());
Collections.sort(list, (o1, o2) -> o2.getValue() - o1.getValue());
for (int i = 0; i < 8 && i < list.size(); i++) {
    String[] token = list.get(i).getKey().split(" ");
    String filename = token[0];
    String line = token[1];
    if (filename.length() > 16) filename = filename.substring(filename.length() - 16);
    System.out.println(filename + " " + line + " " + list.get(i).getValue());
}
```

10. 华为-扑克牌大小

```
public class Main {

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

    public Main() {
        map.put("3", 0);
        map.put("4", 1);
        map.put("5", 2);
        map.put("6", 3);
        map.put("7", 4);
        map.put("8", 5);

        map.put("9", 6);
```

```
        map.put("10", 7);
        map.put("J", 8);
        map.put("Q", 9);
        map.put("K", 10);
        map.put("A", 11);
        map.put("2", 12);
        map.put("joker", 13);
        map.put("JOKER ", 14);
    }

    private String play(String s1, String s2) {
        String[] token1 = s1.split(" ");
        String[] token2 = s2.split(" ");
        CardType type1 = computeCardType(token1);
        CardType type2 = computeCardType(token2);
        if (type1 == CardType.DoubleJoker) return s1;
        if (type2 == CardType.DoubleJoker) return s2;
        if (type1 == CardType.Bomb && type2 != CardType.Bomb) return s1;
        if (type2 == CardType.Bomb && type1 != CardType.Bomb) return s2;
        if (type1 != type2 || token1.length != token2.length) return "ERROR";
        for (int i = 0; i < token1.length; i++) {
            int val1 = map.get(token1[i]);
            int val2 = map.get(token2[i]);
            if (val1 != val2) return val1 > val2 ? s1 : s2;
        }
        return "ERROR";
    }

    private CardType computeCardType(String[] token) {
        boolean hasjoker = false, hasJOKER = false;
        for (int i = 0; i < token.length; i++) {
            if (token[i].equals("joker")) hasjoker = true;
            else if (token[i].equals("JOKER")) hasJOKER = true;
        }
        if (hasjoker && hasJOKER) return CardType.DoubleJoker;
        int maxContinueLen = 1;
        int curContinueLen = 1;
        String curValue = token[0];
        for (int i = 1; i < token.length; i++) {
            if (token[i].equals(curValue)) curContinueLen++;
            else {
                curContinueLen = 1;
                curValue = token[i];
            }
            maxContinueLen = Math.max(maxContinueLen, curContinueLen);
        }
        if (maxContinueLen == 4) return CardType.Bomb;
        if (maxContinueLen == 3) return CardType.Triple;
        if (maxContinueLen == 2) return CardType.Double;
        boolean isStraight = true;
        for (int i = 1; i < token.length; i++) {
            if (map.get(token[i]) - map.get(token[i - 1]) != 1) {

                isStraight = false;
```

```
                    break;
                }
            }
            if (isStraight && token.length == 5) return CardType.Straight;
            return CardType.Sigal;
        }

        private enum CardType {
            DoubleJoker, Bomb, Sigal, Double, Triple, Straight;
        }

        public static void main(String[] args) {
            Main main = new Main();
            Scanner in = new Scanner(System.in);
            while (in.hasNextLine()) {
                String s = in.nextLine();
                String[] token = s.split("-");
                System.out.println(main.play(token[0], token[1]));
            }
        }
    }
```

11. 去哪儿-二分查找

对于有重复元素的有序数组，二分查找需要注意以下要点：

- if (val <= A[m]) h = m;
- 因为 h 的赋值为 m 而不是 m - 1，因此 while 循环的条件也就为 l < h。（如果是 m - 1 循环条件为 l <= h）

```
public int getPos(int[] A, int n, int val) {
    int l = 0, h = n - 1;
    while (l < h) {
        int m = l + (h - l) / 2;
        if (val <= A[m]) h = m;
        else l = m + 1;
    }
    return A[h] == val ? h : -1;
}
```

12. 去哪儿-首个重复字符

```java
public char findFirstRepeat(String A, int n) {
    boolean[] hasAppear = new boolean[256];
    for (int i = 0; i < n; i++) {
        char c = A.charAt(i);
        if(hasAppear[c]) return c;
        hasAppear[c] = true;
    }
    return ' ';
}
```

13. 去哪儿-寻找Coder

```java
public String[] findCoder(String[] A, int n) {
    List<Pair<String, Integer>> list = new ArrayList<>();
    for (String s : A) {
        int cnt = 0;
        String t = s.toLowerCase();
        int idx = -1;
        while (true) {
            idx = t.indexOf("coder", idx + 1);
            if (idx == -1) break;
            cnt++;
        }
        if (cnt != 0) {
            list.add(new Pair<>(s, cnt));
        }
    }
    Collections.sort(list, (o1, o2) -> (o2.getValue() - o1.getValue()));
    String[] ret = new String[list.size()];
    for (int i = 0; i < list.size(); i++) {
        ret[i] = list.get(i).getKey();
    }
    return ret;
}

// 牛客网无法导入 javafx.util.Pair，这里就自己实现一下 Pair 类
private class Pair<T, K> {
    T t;
    K k;

    Pair(T t, K k) {
        this.t = t;
        this.k = k;
    }

    T getKey() {
        return t;
    }

    K getValue() {
        return k;
```

```
    }
}
```

14. 美团-最大差值

贪心策略。

```
public int getDis(int[] A, int n) {
    int max = 0;
    int soFarMin = A[0];
    for (int i = 1; i < n; i++) {
        if(soFarMin > A[i]) soFarMin = A[i];
        else max = Math.max(max, A[i]- soFarMin);
    }
    return max;
}
```

15. 美团-棋子翻转

```
public int[][] flipChess(int[][] A, int[][] f) {
    int[][] direction = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
    for (int[] ff : f) {
        for (int[] dd : direction) {
            int r = ff[0] + dd[0] - 1, c = ff[1] + dd[1] - 1;
            if(r < 0 || r > 3 || c < 0 || c > 3) continue;
            A[r][c] ^= 1;
        }
    }
    return A;
}
```

16. 美团-拜访

```
private Set<String> paths;
private List<Integer> curPath;

public int countPath(int[][] map, int n, int m) {
    paths = new HashSet<>();
    curPath = new ArrayList<>();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (map[i][j] == 1) {
                map[i][j] = -1;
                int[][] leftRightDirection = {{1, 0}, {-1, 0}};
                int[][] topDownDirection = {{0, 1}, {0, -1}};
                for (int[] lr : leftRightDirection) {
                    for (int[] td : topDownDirection) {
                        int[][] directions = {lr, td};
```

```
                        backtracking(map, n, m, i, j, directions);
                    }
                }
                return paths.size();
            }
        }
    }
    return 0;
}

private void backtracking(int[][] map, int n, int m, int r, int c, int[][] directions) {
    if (map[r][c] == 2) {
        String path = "";
        for (int num : curPath) {
            path += num;
        }
        paths.add(path);
        return;
    }
    for (int i = 0; i < directions.length; i++) {
        int nextR = r + directions[i][0];
        int nextC = c + directions[i][1];
        if (nextR < 0 || nextR >= n || nextC < 0 || nextC >= m || map[nextR][nextC] == -1)
continue;
        map[nextR][nextC] = map[nextR][nextC] == 2 ? 2 : -1;
        curPath.add(nextR);
        curPath.add(nextC);
        backtracking(map, n, m, nextR, nextC, directions);
        curPath.remove(curPath.size() - 1);
        curPath.remove(curPath.size() - 1);
        map[nextR][nextC] = map[nextR][nextC] == 2 ? 2 : 0;
    }
}
```

17. 美团-直方图内最大矩形

```
public int countArea(int[] A, int n) {
    int max = 0;
    for (int i = 0; i < n; i++) {
        int min = A[i];
        for (int j = i; j < n; j++) {
            min = Math.min(min, A[j]);
            max = Math.max(max, min * (j - i + 1));
        }
    }
    return max;
}
```

18. 美团-字符串计数

字符串都是小写字符，可以把字符串当成是 26 进制。但是字典序的比较和普通的整数比较不同，是从左往右进行比较，例如 "ac" 和 "abc"，字典序的比较结果为 "ac" > "abc"，如果按照整数方法比较，因为 "abc" 是三位数，显然更大。

由于两个字符串的长度可能不想等，在 s1 空白部分和 s2 对应部分进行比较时，应该把 s1 的空白部分看成是 'a' 字符进行填充的。

还有一点要注意的是，s1 到 s2 长度为 len_i 的字符串个数只比较前面 i 个字符。例如 'aaa' 和 'bbb'，长度为 2 的个数为 'aa' 到 'bb' 的字符串个数，不需要考虑后面部分的字符。

在统计个数时，从 len1 开始一直遍历到最大合法长度，每次循环都统计长度为 i 的子字符串个数。

```
String s1 = in.next();
String s2 = in.next();
int len1 = in.nextInt();
int len2 = in.nextInt();
int len = Math.min(s2.length(), len2);
int[] subtractArr = new int[len];
for (int i = 0; i < len; i++) {
    char c1 = i < s1.length() ? s1.charAt(i) : 'a';
    char c2 = s2.charAt(i);
    subtractArr[i] = c2 - c1;
}
int ret = 0;
for (int i = len1; i <= len; i++) {
    for (int j = 0; j < i; j++) {
        ret += subtractArr[j] * Math.pow(26, i - j - 1);
    }
}
System.out.println(ret - 1);
```

19. 美团-平均年龄

```
int W = in.nextInt();
double Y = in.nextDouble();
double x = in.nextDouble();
int N = in.nextInt();
while (N-- > 0) {
    Y++; // 老员工每年年龄都要加 1
    Y += (21 - Y) * x;
}
System.out.println((int) Math.ceil(Y));
```

20. 百度-罪犯转移

部分和问题，将每次求的部分和缓存起来。

```
int n = in.nextInt();
int t = in.nextInt();
int c = in.nextInt();
```

```
int[] values = new int[n];
for (int i = 0; i < n; i++) {
    values[i] = in.nextInt();
}
int cnt = 0;
int totalValue = 0;
for (int s = 0, e = c - 1; e < n; s++, e++) {
    if (s == 0) {
        for (int j = 0; j < c; j++) totalValue += values[j];
    } else {
        totalValue = totalValue - values[s - 1] + values[e];
    }
    if (totalValue <= t) cnt++;
}
System.out.println(cnt);
```

22. 百度-裁减网格纸

```
int n = in.nextInt();
int minX, minY, maxX, maxY;
minX = minY = Integer.MAX_VALUE;
maxX = maxY = Integer.MIN_VALUE;
for (int i = 0; i < n; i++) {
    int x = in.nextInt();
    int y = in.nextInt();
    minX = Math.min(minX, x);
    minY = Math.min(minY, y);
    maxX = Math.max(maxX, x);
    maxY = Math.max(maxY, y);
}
System.out.println((int) Math.pow(Math.max(maxX - minX, maxY - minY), 2));
```

23. 百度-钓鱼比赛

P (至少钓一条鱼) = 1 - P (一条也钓不到)

坑：读取概率矩阵的时候，需要一行一行进行读取，而不能直接用 in.nextDouble()。

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    while (in.hasNext()) {
        int n = in.nextInt();
        int m = in.nextInt();
        int x = in.nextInt();
        int y = in.nextInt();
        int t = in.nextInt();
        in.nextLine(); // 坑
        double pcc = 0.0;
        double sum = 0.0;

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

```
            String[] token = in.nextLine().split(" "); // 坑
            for (int j = 1; j <= m; j++) {
                double p = Double.parseDouble(token[j - 1]);
                //  double p = in.nextDouble();
                sum += p;
                if (i == x && j == y) {
                    pcc = p;
                }
            }
        }
        double pss = sum / (n * m);
        pcc = computePOfIRT(pcc, t);
        pss = computePOfIRT(pss, t);
        System.out.println(pcc > pss ? "cc" : pss > pcc ? "ss" : "equal");
        System.out.printf("%.2f\n", Math.max(pcc, pss));
    }
}

// compute probability of independent repeated trials
private static double computePOfIRT(double p, int t) {
    return 1 - Math.pow((1 - p), t);
}
```

24. 百度-蘑菇阵

这题用回溯会超时，需要用 DP。

dp[i][j] 表示到达 (i,j) 位置不会触碰蘑菇的概率。对于 N*M 矩阵，如果 i == N || j == M，那么 (i,j) 只能有一个移动方向；其它情况下能有两个移动方向。

考虑以下矩阵，其中第 3 行和第 3 列只能往一个方向移动，而其它位置可以有两个方向移动。

```
int N = in.nextInt();
int M = in.nextInt();
int K = in.nextInt();
boolean[][] mushroom = new boolean[N][M];
while (K-- > 0) {
    int x = in.nextInt();
    int y = in.nextInt();
    mushroom[x - 1][y - 1] = true;
}
double[][] dp = new double[N][M];
dp[0][0] = 1;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        if (mushroom[i][j]) dp[i][j] = 0;
        else {
            double cur = dp[i][j];
            if (i == N - 1 && j == M - 1) break;
            if (i == N - 1) dp[i][j + 1] += cur;
            else if (j == M - 1) dp[i + 1][j] += cur;

            else {
```

```
                dp[i][j + 1] += cur / 2;
                dp[i + 1][j] += cur / 2;
            }
        }
    }
}
System.out.printf("%.2f\n", dp[N - 1][M - 1]);
```