```

题号 题目 知识点 难度 通过率

DD1 连续最大和 动态规划贪心 简单 21.59%

DD2 餐馆 排序贪心 困难 13.77%

DD3 地下迷宫 简单 19.98%

DD4 末尾0的个数 贪心 入门 27.56%

DD5 进制转换 数学 简单 23.24%

DD6 数字和为sum的方法数 动态规划 较难 23.09%

DD7 整数无序数组求第K大数 堆排序 中等 22.97%

DD8 给定整数序列求连续子串最大和 动态规划贪心 中等 31.55%

DD9 寻找丑数 穷举 中等 22.37%

DD10 xor 贪心哈希 中等 19.62%

DD11 幂运算 模拟 中等 7.10%

DD12 几个岛 图数组模拟 中等 11.88%

DD13 最短字符编码 字符串模拟 中等 28.74%

DD14 CIDR去重 字符串模拟 中等 37.20%

```

### DD1 连续最大和

#### 题目描述

```

一个数组有 N 个元素，求连续子数组的最大和。 例如：[-1,2,1]，和最大的连续子数组为[2,1]，其和为 3

```

#### 输入描述:

```

输入为两行。 第一行一个整数n(1 <= n <= 100000)，表示一共有n个元素 第二行为n个数，即每个元素,每个整数都在32位int范围内。以空格分隔。

```

#### 输出描述:

```

所有连续子数组中和最大的值。

```

#### 示例1

#### 输入

```

3

-1 2 1

```

#### 输出

```

3

```

```java

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

while (in.hasNext()) {

int n = in.nextInt();

int[] arr = new int[n];

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

arr[i] = in.nextInt();

}

int[] dp = new int[n];

dp[0] = arr[0];

int max = arr[0];

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

dp[i] = Math.max(dp[i - 1] + arr[i], arr[i]);

if (dp[i] > max) max = dp[i];

}

System.out.println(max);

}

}

}

```

### DD2 餐馆

#### 题目描述

```

某餐馆有n张桌子，每张桌子有一个参数：a 可容纳的最大人数； 有m批客人，每批客人有两个参数:b人数，c预计消费金额。 在不允许拼桌的情况下，请实现一个算法选择其中一部分客人，使得总预计消费金额最大

```

#### 输入描述:

```

输入包括m+2行。 第一行两个整数n(1 <= n <= 50000),m(1 <= m <= 50000) 第二行为n个参数a,即每个桌子可容纳的最大人数,以空格分隔,范围均在32位int范围内。 接下来m行，每行两个参数b,c。分别表示第i批客人的人数和预计消费金额,以空格分隔,范围均在32位int范围内。

```

#### 输出描述:

```

输出一个整数,表示最大的总预计消费金额

```

#### 示例1

#### 输入

```

3 5 2 4 2 1 3 3 5 3 7 5 9 1 10

```

#### 输出

```

20

```

```java

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (sc.hasNext()) {

int n = sc.nextInt();

int m = sc.nextInt();

int[] table = new int[n];

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

table[i] = sc.nextInt();

}

int[][] cus = new int[m][2];

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

cus[i][0] = sc.nextInt();

cus[i][1] = sc.nextInt();

}

Arrays.sort(table);

Arrays.sort(cus, (a, b) -> b[1] - a[1]);

long res = 0L;

int index;

boolean[] tableb = new boolean[n];

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

if (cus[i][0] > table[n - 1]) continue;

index = bs(table, cus[i][0]);

while (index < n && tableb[index]) index++;

if (index < n) {

res += cus[i][1];

tableb[index] = true;

}

}

System.out.println(res);

}

sc.close();

}

private static int bs(int[] num, int tar) {

int low = 0;

int high = num.length - 1;

int mid;

while (low <= high) {

mid = (high + low) >> 1;

if (num[mid] >= tar) high = mid - 1;

else low = mid + 1;

}

return low;

}

}

```

### DD3 地下迷宫

#### 题目描述

```

小青蛙有一天不小心落入了一个地下迷宫,小青蛙希望用自己仅剩的体力值P跳出这个地下迷宫。为了让问题简单,假设这是一个n\*m的格子迷宫,迷宫每个位置为0或者1,0代表这个位置有障碍物,小青蛙达到不了这个位置;1代表小青蛙可以达到的位置。小青蛙初始在(0,0)位置,地下迷宫的出口在(0,m-1)(保证这两个位置都是1,并且保证一定有起点到终点可达的路径),小青蛙在迷宫中水平移动一个单位距离需要消耗1点体力值,向上爬一个单位距离需要消耗3个单位的体力值,向下移动不消耗体力值,当小青蛙的体力值等于0的时候还没有到达出口,小青蛙将无法逃离迷宫。现在需要你帮助小青蛙计算出能否用仅剩的体力值跳出迷宫(即达到(0,m-1)位置)。

```

#### 输入描述:

```

输入包括n+1行:

第一行为三个整数n,m(3 <= m,n <= 10),P(1 <= P <= 100)

接下来的n行:

每行m个0或者1,以空格分隔

```

#### 输出描述:

```

如果能逃离迷宫,则输出一行体力消耗最小的路径,输出格式见样例所示;如果不能逃离迷宫,则输出"Can not escape!"。 测试数据保证答案唯一

```

#### 示例1

#### 输入

```

4 4 10 1 0 0 1 1 1 0 1 0 1 1 1 0 0 1 1

```

#### 输出

```

[0,0],[1,0],[1,1],[2,1],[2,2],[2,3],[1,3],[0,3]

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String s;

while ((s = br.readLine()) != null) {

int n = Integer.parseInt(s.split(" ")[0]);

int m = Integer.parseInt(s.split(" ")[1]);

int p = Integer.parseInt(s.split(" ")[2]);

int[][] maze = new int[n][m];

String[] arr;

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

arr = br.readLine().split(" ");

for (int j = 0; j < m; j++) maze[i][j] = Integer.parseInt(arr[j]);

}

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

boolean bool = solve(maze, p, 0, 0, list);

if (bool) {

for (int i = 0; i < list.size() - 1; i++) System.out.print(list.get(i) + ",");

System.out.println(list.get(list.size() - 1));

} else System.out.println("Can not escape!");

}

}

private static boolean solve(int[][] maze, int p, int x, int y, List<String> list) {

maze[x][y] = 0;

list.add("[" + x + "," + y + "]");

if (x == 0 && y == maze[0].length - 1) return true;

//向上

if (x > 0 && maze[x - 1][y] == 1 && p >= 3) if (solve(maze, p - 3, x - 1, y, list)) return true;

//向右

if (y < maze[0].length - 1 && maze[x][y + 1] == 1 && p >= 1)

if (solve(maze, p - 1, x, y + 1, list)) return true;

//向下

if (x < maze.length - 1 && maze[x + 1][y] == 1) if (solve(maze, p, x + 1, y, list)) return true;

//向左

if (y > 0 && maze[x][y - 1] == 1 && p >= 1) if (solve(maze, p - 1, x, y - 1, list)) return true;

maze[x][y] = 1;

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

return false;

}

}

```

### DD4 末尾0的个数

#### 题目描述

```

输入一个正整数n,求n!(即阶乘)末尾有多少个0？ 比如: n = 10; n! = 3628800,所以答案为2

```

#### 输入描述:

```

输入为一行，n(1 ≤ n ≤ 1000)

```

#### 输出描述:

```

输出一个整数,即题目所求

```

#### 示例1

#### 输入

```

10

```

#### 输出

```

2

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

static int[] coins = {1, 4, 16, 64};

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String s;

while ((s = reader.readLine()) != null) {

int n = Integer.parseInt(s);

int res = 0;

while (n > 0) {

res += n / 5;

n /= 5;

}

System.out.println(res);

}

}

}

```

### DD5 进制转换

#### 题目描述

```

给定一个十进制数M，以及需要转换的进制数N。将十进制数M转化为N进制数

```

#### 输入描述:

```

输入为一行，M(32位整数)、N(2 ≤ N ≤ 16)，以空格隔开。

```

#### 输出描述:

```

为每个测试实例输出转换后的数，每个输出占一行。如果N大于9，则对应的数字规则参考16进制（比如，10用A表示，等等）

```

#### 示例1

#### 输入

```

7 2

```

#### 输出

```

111

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

//进制转换

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] nums = br.readLine().split(" ");

int m = Integer.parseInt(nums[0]);

int n = Integer.parseInt(nums[1]);

StringBuilder sb = new StringBuilder();

char[] arr = {'A', 'B', 'C', 'D', 'E', 'F'};

int temp;

boolean fs = false;

if (m < 0) {

fs = true;

m = -m;

}

while (m != 0) {

temp = m % n;

if (temp > 9) sb.append(arr[temp - 9 - 1]);

else sb.append(temp);

m = m / n;

}

if (fs) sb.append("-");

System.out.println(sb.reverse());

}

}

```

### DD6 数字和为sum的方法数

#### 题目描述

```

给定一个有n个正整数的数组A和一个整数sum,求选择数组A中部分数字和为sum的方案数。

当两种选取方案有一个数字的下标不一样,我们就认为是不同的组成方案。

```

#### 输入描述:

```

输入为两行:

第一行为两个正整数n(1 ≤ n ≤ 1000)，sum(1 ≤ sum ≤ 1000)

第二行为n个正整数A[i](32位整数)，以空格隔开。

```

#### 输出描述:

```

输出所求的方案数

```

#### 示例1

#### 输入

```

5 15 5 5 10 2 3

```

#### 输出

```

4

```

```java

import java.math.BigInteger;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

int n;

int sum;

int[] A;

Scanner cin = new Scanner(System.in);

n = cin.nextInt();

sum = cin.nextInt();

A = new int[n];

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

A[i] = cin.nextInt();

}

System.out.println(getTypeCount1(n, sum, A));

}

public static BigInteger getTypeCount1(int n, int sum, int[] A) {

BigInteger[] c = new BigInteger[1005];

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

c[i] = BigInteger.ZERO;

}

c[0] = BigInteger.ONE;

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

int index = A[i];

for (int j = sum; j >= 0; j--) {

if (j >= index && !c[j - index].equals(BigInteger.ZERO)) c[j] = c[j].add(c[j - index]);

}

}

return c[sum];

}

}

```

### DD7 整数无序数组求第K大数

#### 题目描述

```

给定无序整数序列，求其中第K大的数，例如{45，67，33，21}，第2大数为45

```

#### 输入描述:

```

输入第一行为整数序列，数字用空格分隔，如：45 67 33 21

输入第二行一个整数K，K在数组长度范围内，如：2

```

#### 输出描述:

```

输出第K大的数，本例为第2大数：45

```

#### 示例1

#### 输入

```

45 67 33 21

2

```

#### 输出

```

45

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));

String[] S = bf.readLine().split(" ");

int[] a = new int[S.length];

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

a[i] = Integer.parseInt(S[i]);

}

int k = Integer.parseInt(bf.readLine());

Arrays.sort(a);

System.out.println(a[a.length - k]);

}

}

```

### DD8 给定整数序列求连续子串最大和

#### 题目描述

```

给定无序整数序列，求连续非空子串最大和，例如{-23 17 -7 11 -2 1 -34}，子串为{17,-7,11}，最大和为21

```

#### 输入描述:

```

输入为整数序列，数字用空格分隔，如：-23 17 -7 11 -2 1 -34

```

#### 输出描述:

```

输出为子序列的最大和：21

```

#### 示例1

#### 输入

```

-23 17 -7 11 -2 1 -34

```

#### 输出

```

21

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] strs = br.readLine().split(" ");

int[] a = new int[strs.length];

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

a[i] = Integer.parseInt(strs[i]);

}

int[] dp = new int[a.length];

dp[0] = a[0];

int res = dp[0];

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

dp[i] = (dp[i - 1] >= 0) ? dp[i - 1] + a[i] : a[i];

res = Math.max(res, dp[i]);

}

System.out.println(res);

}

}

```

### DD9 寻找丑数

#### 题目描述

```

把只包含因子2、3和5的数称作丑数（Ugly Number）。例如6、8都是丑数，但14不是，因为它包含因子7。 习惯上我们把1当做是第一个丑数。求按从小到大的顺序的第N个丑数。

```

#### 输入描述:

```

整数N

```

#### 输出描述:

```

第N个丑数

```

#### 示例1

#### 输入

```

6

```

#### 输出

```

6

```

```java

import java.io.InputStreamReader;

import java.io.BufferedReader;

import java.io.IOException;

public class Main {

private static boolean mm(int i) {

while (i % 2 == 0) i = i / 2;

while (i % 3 == 0) i = i / 3;

while (i % 5 == 0) i = i / 5;

return i == 1;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

br.close();

if (n == 1) {

System.out.println(1);

return;

}

int i = 2;

int j = 1;

while (true) {

if (mm(i)) {

j++;

if (n == j) {

System.out.println(i);

return;

}

}

i++;

}

}

}

```

### DD10 xor

#### 题目描述

```

给出n个数字 a\_1,...,a\_n，问最多有多少不重叠的非空区间，使得每个区间内数字的xor都等于0。

```

#### 输入描述:

```

第一行一个整数ｎ； 第二行ｎ个整数　a\_1,...,a\_n； 对于30%的数据，n<=20； 对于100%的数据，n<=100000, a\_i<=100000；

```

#### 输出描述:

```

一个整数表示最多的区间个数；

```

#### 示例1

#### 输入

```

4

3 0 2 2

```

#### 输出

```

2

```

```java

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

int index = 0, res = 0;

for (String tmp : br.readLine().split(" ")) {

arr[index++] = Integer.parseInt(tmp);

}

br.close();

index = -1;

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

int tmp = 0;

for (int j = i; j > index; j--) { // 从后向前

tmp ^= arr[j];

if (tmp == 0) {

res++;

index = i; // 记录该区间的end 边界

break;

}

}

}

System.out.println(res);

}

}

```

### DD11 幂运算

#### 题目描述

```

给定两个数R和n，输出R的n次方，其中0.0<R<99.999, 0<n<=25

```

#### 输入描述:

```

多组测试用例，请参考例题的输入处理 输入每行一个浮点数 R 其中0.0 < R <99.999， 一个整数 n 其中0 < n <=25

```

#### 输出描述:

```

输出R的n次方

```

#### 示例1

#### 输入

```

95.123 12 0.1 1

```

#### 输出

```

548815620517731830194541.899025343415715973535967221869852721 0.1

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.math.BigDecimal;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str;

while ((str = br.readLine()) != null) {

if (str.equals("")) continue;

String[] params = str.split(" ");

StringBuilder res = new StringBuilder();

for (int i = 0; i < params.length; i += 2) {

String R = params[i];

if (params[i + 1].equals("")) i++;

int n = Integer.parseInt(params[i + 1]);

res.append(power(R, n)).append(" ");

}

System.out.println(res.toString().trim());

}

}

// 计算乘方

private static String power(String R, int n) {

String res = "1";

for (int i = 0; i < n; i++) res = multiply(R, res);

return res;

}

// 计算乘法

private static String multiply(String num1, String num2) {

BigDecimal float1 = new BigDecimal(num1);

BigDecimal float2 = new BigDecimal(num2);

// 去掉后面的0，并取消科学计数法

return float1.multiply(float2).stripTrailingZeros().toPlainString();

}

}

```

### DD12 几个岛

#### 题目描述

```

给定一个m行n列的二维地图, 初始化每个单元都是水.

操作addLand 把单元格(row,col)变成陆地.

岛屿定义为一系列相连的被水单元包围的陆地单元, 横向或纵向相邻的陆地称为相连(斜对角不算).

在一系列addLand的操作过程中, 给出每次addLand操作后岛屿的个数.

二维地图的每条边界外侧假定都是水.

```

#### 输入描述:

```

多组测试数据，请参考例题处理 每组数据k+3行, k表示addLand操作次数 第一行:表示行数m 第二行:表示列数n 第三行:表示addLand操作次数k 第4~k+3行:row col 表示addLand的坐标。注意超过边界的坐标是无效的。

```

#### 输出描述:

```

一行,k个整数, 表示每次addLand操作后岛屿的个数, 用空格隔开，结尾无空格

```

#### 示例1

#### 输入

```

3

3

4

0 0

0 1

1 2

2 1

```

#### 输出

```

1 1 2 3

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

static int[][] map;

static int n;

static int m;

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String s;

while ((s = br.readLine()) != null) {

n = Integer.parseInt(s.trim());

s = br.readLine();

m = Integer.parseInt(s.trim());

map = new int[n][m];

s = br.readLine();

int k = Integer.parseInt(s.trim());

StringBuilder sb = new StringBuilder();

int res = 0;

for (int a = 0; a < k; a++) {

s = br.readLine();

String[] strs = s.split(" ");

int map\_n = Integer.parseInt(strs[0]);

int map\_m = Integer.parseInt(strs[1]);

if (map\_n < 0 || map\_n >= n || map\_m < 0 || map\_m >= m) {

sb.append(res).append(" ");

continue;

}

res = 0;

map[map\_n][map\_m] = 1;

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

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

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

res++;

dfs(i, j);

}

}

}

sb.append(res).append(" ");

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

for (int j = 0; j < m; j++) if (map[i][j] == 2) map[i][j] = 1;

}

}

System.out.println(sb.toString().trim());

}

}

static void dfs(int i, int j) {

if (i < 0 || j < 0 || i >= n || j >= m || map[i][j] != 1) return;

map[i][j] = 2;

dfs(i - 1, j);

dfs(i + 1, j);

dfs(i, j - 1);

dfs(i, j + 1);

}

}

```

### DD13 最短字符编码

#### 题目描述

```

给定一个非空字符串, 按照如下方式编码, 使得编码后长度最小, 返回编码后的长度:

编码规则为: k[encoding\_string], 表示重复k次encoding\_strng,

例如'abcdefabcdefabc'可表示为'2[abcdef]abc', 但是'aaa'仅能编码成'aaa',

因为len('3[a]')>len('aaa').

补充:

1. k为正整数, []内的encoding\_string不得含有空格不得为空;

2. []内的encoding\_string 本身可以为编码过的字符串, 例如'abcdabcdeabcdabcde' 可以编码为 '2[abcdabcde]'(编码后长度从18减少到12), []内的'abcdabcde'又可以编码为 '2[abcd]e', 最终编码为 '2[2[abcd]e]', 编码后长度为11, 应返回11; 这个编码路径也能是: 'abcdabcdeabcdabcde' -> '2[abcd]e2[abcd]e' -> '2[2[abcd]e]';

2. 输入字符串为全小写英文字母, 长度<=160;

3. 如果编码后长度没有更小, 则保留原有字符串;

```

#### 输入描述:

```

一行数据, 表示输入字符串

```

#### 输出描述:

```

输出一个字符串表示编码后长度

```

#### 示例1

#### 输入

```

aaa

```

#### 输出

```

3

说明

aaa，长度3

```

#### 示例2

#### 输入

```

aaaaa

```

#### 输出

```

4

说明

5[a]，长度4

```

#### 示例3

#### 输入

```

aabcaabcd

```

#### 输出

```

8

说明

2[aabc]d，长度8

```

```java

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String s = br.readLine();

br.close();

System.out.println(solution(s).length());

}

private static String solution(String s) {

if (s.length() <= 4) return s;

int len = s.length();

int rptLen = len >> 1; // 当前尝试的重复的长度

int bestRptTime = 0;

int bestCompressLen = len;

String seg1 = "", seg2 = "", seg3 = "";

while (rptLen >= 1) {

for (int k = 0; k <= len - (rptLen << 1); k++) { // 从k位置开始寻找，至少留两段

int count = 1;

String s2 = s.substring(k, k + rptLen);

for (int j = 1; k + j \* rptLen + rptLen <= len; j++) { // 判断最多重复几次

String s3 = s.substring(k + j \* rptLen, k + (j + 1) \* rptLen);

if (s2.equals(s3)) count++;

else break;

}

int newLen = len - count \* rptLen + 3 + rptLen;

if (newLen < len && newLen < bestCompressLen) { // 新长度合适

bestCompressLen = newLen;

bestRptTime = count;

seg1 = s.substring(0, k);

seg2 = s.substring(k, k + rptLen);

seg3 = s.substring(k + count \* rptLen);

}

}

rptLen--;

}

if (bestRptTime == 0) return s;

return solution(seg1) + bestRptTime + "[" + solution(seg2) + "]" + solution(seg3);

}

}

```

### DD14 CIDR去重

#### 题目描述

```

无类别域间路由（CIDR）是一个用于对IPV4地址进行分类表述的方法。CIDR 路由描述的IP地址组的子网mask长度是可变长度, 例如10.0.0.0/22 表示前22位和10.0.0.0相同的网络地址都被覆盖, 22包含了10.0这前两个字段(0-7位,8-15位)和第三个字段的前6位(16-21,即0b000000\*\*), 涵盖了 10.0.0.\*, 10.0.1.\*, 10.0.2.\*, 10.0.3.\* 四组ip地址. 在此前提下请实现IP网络中的一个常用的去重操作: 给定一系列 CIDR 路由地址, 其中没有完全等价的路由, 去掉被重复表示的 CIDR 路由, 即去掉已经被其他CIDR路由表示覆盖的路由地址. 例如 10.0.1.1/32 已经被 10.0.0.0/22覆盖了, 如果路由列表中已经有了后者, 就可以去掉前者.

```

#### 输入描述:

```

k+1行, k表示输入的CIDR路由个数

第1行:表示路由个数k

第2~k+1行: 表示一个CIDR路由, 形如 x.x.x.x/x

```

#### 输出描述:

```

n+1行, n表示去重后剩下的CIDR路由个数

第1行:n

第2~n+1行: 表示一个去重后的CIDR路由, 输出按照输入顺序

```

#### 示例1

#### 输入

```

13

192.168.0.0/16

172.24.96.17/32

172.50.137.225/32

202.139.219.192/32

172.24.68.0/24

192.183.125.71/32

201.45.111.138/32

192.168.59.211/32

192.168.26.13/32

172.24.0.0/17

172.24.5.1/32

172.24.68.37/32

172.24.168.32/32

```

#### 输出

```

7

192.168.0.0/16

172.50.137.225/32

202.139.219.192/32

192.183.125.71/32

201.45.111.138/32

172.24.0.0/17

172.24.168.32/32

```

```java

import java.io.\*;

import java.util.\*;

import java.lang.String;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str;

while ((str = br.readLine()) != null) {

if (str.equals("")) continue;

int k = Integer.parseInt(str);

int[][] address = new int[k][2];

String[] res = new String[k];

boolean[] drop = new boolean[k];

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

res[i] = br.readLine();

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

address[i][1] = Integer.parseInt(ip[1]);

String[] segments = ip[0].split("\\.");

for (int j = 0; j < 4; j++) address[i][0] = (address[i][0] << 8) + Integer.parseInt(segments[j]);

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

if (drop[j]) continue;

if (address[j][1] <= address[i][1] && (address[j][0] >> (32 - address[j][1])) == (address[i][0] >> (32 - address[j][1]))) {

drop[i] = true;

break;

} else if (address[j][1] > address[i][1] && (address[j][0] >> (32 - address[i][1])) == (address[i][0] >> (32 - address[i][1]))) {

drop[j] = true;

}

}

}

StringBuilder ans = new StringBuilder();

int count = 0;

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

if (!drop[i]) {

count++;

ans.append(res[i]).append("\n");

}

}

System.out.print(count + "\n" + ans);

}

br.close();

}

}

```