```

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 题号 | 题目 | 知识点 | 难度 | 通过率 |
| PDD1 | 最大乘积 | 贪心模拟 | 中等 | 14.45% |
| PDD2 | 大整数相乘 | 模拟 | 中等 | 27.32% |
| PDD3 | 六一儿童节 | 贪心 | 中等 | 24.74% |
| PDD4 | 迷宫寻路 | 模拟 | 中等 | 14.77% |
| PDD5 | 列表补全 | 数组模拟 | 简单 | 21.09% |
| PDD6 | 拼多多周年庆 | 动态规划贪心树图 | 较难 | 23.93% |
| PDD7 | 数三角形 | 模拟排序穷举 | 中等 | 23.72% |
| PDD8 | 最大乘积 | 贪心模拟 | 中等 | 21.08% |
| PDD9 | 小熊吃糖 | 排序数组模拟贪心 | 简单 | 24.97% |
| PDD10 | 选靓号 | 字符串贪心 | 较难 | 15.29% |
| PDD11 | 种树 | 递归数组贪心 | 中等 | 15.95% |
| PDD12 | 两两配对差值最小 | 排序贪心数组 | 简单 | 38.52% |
| PDD13 | 回合制游戏 | 贪心 | 简单 | 24.09% |

```

### PDD1 最大乘积

#### 题目描述

```

给定一个无序数组，包含正数、负数和0，要求从中找出3个数的乘积，使得乘积最大，要求时间复杂度：O(n)，空间复杂度：O(1)

```

#### 输入描述:

```

输入共2行，第一行包括一个整数n，表示数组长度

第二行为n个以空格隔开的整数，分别为A1,A2, … ,An

```

#### 输出描述:

```

满足条件的最大乘积

```

#### 示例1

#### 输入

```

4

3 4 1 2

```

#### 输出

```

24

```

```java

import java.io.\*;

public class Main {

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

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

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

String[] data = in.readLine().split(" ");

long max1 = Integer.MIN\_VALUE;

long max2 = Integer.MIN\_VALUE;

long max3 = Integer.MIN\_VALUE;

long min1 = Integer.MAX\_VALUE;

long min2 = Integer.MAX\_VALUE;

for (String num : data) {

int temp = Integer.parseInt(num);

if (temp > max1) {

max3 = max2;

max2 = max1;

max1 = temp;

} else if (temp > max2) {

max3 = max2;

max2 = temp;

} else if (temp > max3) {

max3 = temp;

}

//找最小的两个数

if (temp < min1) {

min2 = min1;

min1 = temp;

} else if (temp < min2) {

min2 = temp;

}

}

long max = Integer.MIN\_VALUE;

max = Math.max(max, max1 \* max2 \* max3);

max = Math.max(max, max1 \* min1 \* min2);

System.out.println(max);

}

}

```

### PDD2 大整数相乘

#### 题目描述

```

有两个用字符串表示的非常大的大整数,算出他们的乘积，也是用字符串表示。不能用系统自带的大整数类型。

```

#### 输入描述:

```

空格分隔的两个字符串，代表输入的两个大整数

```

#### 输出描述:

```

输入的乘积，用字符串表示

```

#### 示例1

#### 输入

```

72106547548473106236 982161082972751393

```

#### 输出

```

70820244829634538040848656466105986748

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

public String mul(String num1, String num2) {

int[] nums1 = new int[num1.length()], nums2 = new int[num2.length()];

int[] res = new int[num1.length() + num2.length()];

for (int i = num1.length() - 1; i >= 0; i--) {

nums1[i] = num1.charAt(num1.length() - 1 - i) - '0';

}

for (int i = num2.length() - 1; i >= 0; i--) {

nums2[i] = num2.charAt(num2.length() - 1 - i) - '0';

}

for (int i = 0; i < num1.length(); i++) {

for (int j = 0; j < num2.length(); j++) {

res[i + j] += nums1[i] \* nums2[j];

}

}

//进位和留位

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

//进位

res[i] += res[i - 1] / 10;

//留位

res[i - 1] = res[i - 1] % 10;

}

StringBuilder buffer = new StringBuilder();

boolean start = false;

for (int i = res.length - 1; i >= 0; i--) {

if (!start && res[i] == 0) continue;

else start = true;

buffer.append(res[i]);

}

return buffer.toString();

}

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

Main c = new Main();

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

String input = bf.readLine();

String[] inputs = input.split(" ");

String res = c.mul(inputs[0], inputs[1]);

System.out.println(res);

}

}

```

### PDD3 六一儿童节

#### 题目描述

```

六一儿童节，老师带了很多好吃的巧克力到幼儿园。每块巧克力j的重量为w[j]，对于每个小朋友i，当他分到的巧克力大小达到h[i] (即w[j]>=h[i])，他才会上去表演节目。老师的目标是将巧克力分发给孩子们，使得最多的小孩上台表演。可以保证每个w[i]> 0且不能将多块巧克力分给一个孩子或将一块分给多个孩子。

```

#### 输入描述:

```

第一行：n，表示h数组元素个数

第二行：n个h数组元素

第三行：m，表示w数组元素个数

第四行：m个w数组元素

```

#### 输出描述:

```

上台表演学生人数

```

#### 示例1

#### 输入

```

3 <br> 2 2 3<br> 2<br> 3 1

```

#### 输出

```

1

```

```java

import java.util.\*;

import java.io.\*;

import java.io.BufferedReader;

public class Main {

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

InputStreamReader ir = new InputStreamReader(System.in);

BufferedReader bf = new BufferedReader(ir);

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

String[] str1 = bf.readLine().split(" ");

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

String[] str2 = bf.readLine().split(" ");

int[] h = new int[n];

int[] w = new int[m];

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

h[i] = Integer.parseInt(str1[i]);

}

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

w[i] = Integer.parseInt(str2[i]);

}

Arrays.sort(w);

Arrays.sort(h);

int ans = 0;

int i = 0, j = 0;

while (i < n && j < m) {

if (h[i] <= w[j]) {

ans++;

i++;

}

j++;

}

System.out.println(ans);

}

}

```

### PDD4 迷宫寻路

#### 题目描述

```

假设一个探险家被困在了地底的迷宫之中，要从当前位置开始找到一条通往迷宫出口的路径。迷宫可以用一个二维矩阵组成，有的部分是墙，有的部分是路。迷宫之中有的路上还有门，每扇门都在迷宫的某个地方有与之匹配的钥匙，只有先拿到钥匙才能打开门。请设计一个算法，帮助探险家找到脱困的最短路径。如前所述，迷宫是通过一个二维矩阵表示的，每个元素的值的含义如下 0-墙，1-路，2-探险家的起始位置，3-迷宫的出口，大写字母-门，小写字母-对应大写字母所代表的门的钥匙

```

#### 输入描述:

```

迷宫的地图，用二维矩阵表示。第一行是表示矩阵的行数和列数M和N

后面的M行是矩阵的数据，每一行对应与矩阵的一行（中间没有空格）。M和N都不超过100, 门不超过10扇。

```

#### 输出描述:

```

路径的长度，是一个整数

```

#### 示例1

#### 输入

```

5 5

02111

01a0A

01003

01001

01111

```

#### 输出

```

7

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.LinkedList;

class Node {

int x;

int y;

int keys;

int step;

Node(int x, int y, int keys, int step) {

this.x = x;

this.y = y;

this.keys = keys;

this.step = step;

}

}

public class Main {

public static int[][] dir = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

public static int m;

public static int n;

public static char[][] maze;

public static boolean[][][] visit;

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

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

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

m = Integer.parseInt(strs[0]);

n = Integer.parseInt(strs[1]);

maze = new char[m][n];

visit = new boolean[m][n][1024];

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

maze[i] = br.readLine().toCharArray();

}

int res = 0;

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

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

if (maze[i][j] == '2') res = bfs(i, j);

}

}

System.out.println(res);

}

private static int bfs(int x, int y) {

LinkedList<Node> queue = new LinkedList<>();

queue.add(new Node(x, y, 0, 0));

visit[x][y][0] = true;

while (!queue.isEmpty()) {

Node node = queue.poll();

for (int i = 0; i < 4; i++) {

int nx = node.x + dir[i][0], ny = node.y + dir[i][1];

if (nx < 0 || nx >= m || ny < 0 || ny >= n || maze[nx][ny] == '0') continue;

if (maze[nx][ny] == '3') return node.step + 1;

if (maze[nx][ny] >= 'A' && maze[nx][ny] <= 'Z' && ((node.keys & (1 << (maze[nx][ny] - 'A'))) == 0))

continue;

int keys = node.keys;

if (maze[nx][ny] >= 'a' && maze[nx][ny] <= 'z') keys = node.keys | (1 << (maze[nx][ny] - 'a'));

if (!visit[nx][ny][keys]) {

visit[nx][ny][keys] = true;

queue.offer(new Node(nx, ny, keys, node.step + 1));

}

}

}

return -1;

}

}

```

### PDD5 列表补全

#### 题目描述

```

在商城的某个位置有一个商品列表，该列表是由L1、L2两个子列表拼接而成。当用户浏览并翻页时，需要从列表L1、L2中获取商品进行展示。展示规则如下：

1. 用户可以进行多次翻页，用offset表示用户在之前页面已经浏览的商品数量，比如offset为4，表示用户已经看了4个商品

2. n表示当前页面需要展示的商品数量

3. 展示商品时首先使用列表L1，如果列表L1长度不够，再从列表L2中选取商品

4. 从列表L2中补全商品时，也可能存在数量不足的情况

请根据上述规则，计算列表L1和L2中哪些商品在当前页面被展示了

```

#### 输入描述:

```

每个测试输入包含1个测试用例，包含四个整数，分别表示偏移量offset、元素数量n，列表L1的长度l1，列表L2的长度l2。

```

#### 输出描述:

```

在一行内输出四个整数分别表示L1和L2的区间start1，end1，start2，end2，每个数字之间有一个空格。

注意，区间段使用半开半闭区间表示，即包含起点，不包含终点。如果某个列表的区间为空，使用[0, 0)表示，如果某个列表被跳过，使用[len, len)表示，len表示列表的长度。

```

#### 示例1

#### 输入

```

2 4 4 4

1 2 4 4

4 1 3 3

```

#### 输出

```

2 4 0 2

1 3 0 0

3 3 1 2

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main {

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

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

String str;

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

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

int offset = Integer.parseInt(strings[0]);//用户已经看过的商品数量

int n = Integer.parseInt(strings[1]);//需要展示的商品数量

int l1 = Integer.parseInt(strings[2]);//列表L1的长度

int l2 = Integer.parseInt(strings[3]);//列表L2的长度

//分类讨论

//特判 偏移量达到或超过列表1的长度，此时只能使用列表2

if (offset >= l1) {

int temp = offset - l1;//列表2使用的起始行

//列表2也完全没用

if (temp >= l2) System.out.println(l1 + " " + l1 + " " + l2 + " " + l2);

//列表2也不够用

else if (temp + n >= l2) System.out.print(l1 + " " + l1 + " " + temp + " " + l2);

else System.out.println(l1 + " " + l1 + " " + temp + " " + (temp + n));

}

//特判 需要展示的商品数量加上偏移量未超过列表1的长度，只使用列表1

else if (offset + n <= l1) System.out.println(offset + " " + (offset + n) + " " + 0 + " " + 0);

//普通情况

else System.out.println(offset + " " + l1 + " " + 0 + " " + (n - (l1 - offset)));

}

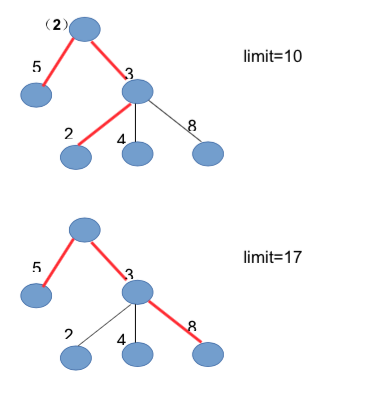
}

}

```

### PDD6 拼多多周年庆

#### 题目描述



```

拼多多王国的城市和道路的拓扑结构比较特别，是一个树状结构：

1. 每个城市是树的一个节点；

2. 城市之间的道路是树的一条边；

3. 树的根节点是首都。

拼多多周年庆马上就要到了，这是拼多多王国的一个大日子。为了活跃气氛，国王想在道路上布置花灯。花灯可是很贵的东西，尽管国王想要在所有道路上都布置花灯，但是如果要花太多钱的话，是过不了财政大臣那一关的。国王把这个计划告诉财政大臣，最后他们商讨出来这么一个方案：

1. 一条道路要么不布置花灯，要么整条布置花灯，不能选择其中的某一段布置；

2. 除非没有道路通向首都，否则至少为一条通向首都的道路布置花灯；

3. 所有布置花灯的道路构成的子图是连通的，这保证国王从首都出发，能通过只走布置了花灯的道路，把所有的花灯游览完；

4. 如果某个城市（包括首都）有大于等于2条道路通向子城市，为了防止铺张浪费，最多只能选择其中的两条路布置花灯；

5. 布置花灯的道路的总长度设定一个上限。

在上述方案下，国王想要使得布置花灯的道路长度越长越好，你帮国王想想办法。

```

#### 输入描述:

```

每个测试输入包含1个测试用例。

输入的第一行是一个正整数m，0<m<=9900，表示布置花灯的道路的总长度的上限。

输入的第二行是一个正整数n，n<=100，表示城市的个数。

紧接着是n-1行输入，每行三个正整数u、v、d，表示下标为u的城市有一条长度为d的道路通向它的一个子城市v，其中0<=u<n，0<=v<n，0<d<=100。

```

#### 输出描述:

```

输出一个正整数，表示能布置花灯的道路长度的最大值

```

#### 示例1

#### 输入

```

5

5

0 1 1

0 2 2

0 3 3

0 4 4

```

#### 输出

```

5

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

static class TreeNode {

int val;

int parent = -1;

//把子节点和权值分开存储

ArrayList<TreeNode> child = new ArrayList<>();

ArrayList<Integer> dist = new ArrayList<>();

public TreeNode(int val) {

this.val = val;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int limit = scanner.nextInt();

int n = scanner.nextInt();

TreeNode[] treeNodes = new TreeNode[n];

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

treeNodes[i] = new TreeNode(i);

}

for (int i = 0; i < n - 1; i++) {

int parent = scanner.nextInt();

int child = scanner.nextInt();

int dist = scanner.nextInt();

treeNodes[parent].child.add(treeNodes[child]);

treeNodes[child].parent = parent;

treeNodes[parent].dist.add(dist);

}

//找到根节点

TreeNode root = null;

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

if (treeNodes[i].parent == -1) {

root = treeNodes[i];

break;

}

}

HashSet<Integer> set = getPath(root, limit);

int max = Integer.MIN\_VALUE;

for (int len : set) {

max = Math.max(max, len);

}

System.out.println(max);

}

private static HashSet<Integer> getPath(TreeNode root, int limit) { //limit不变

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

set.add(0);

if (root == null) return set;

//保持连通是通过从子树往上递归

ArrayList<HashSet<Integer>> arr = new ArrayList<>();

for (int i = 0; i < root.child.size(); i++) {

HashSet<Integer> childSet = getPath(root.child.get(i), limit); //以第一个参数为根节点的处理方法

arr.add(childSet);

}

// one path

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

int d = root.dist.get(i);

//最少选一条路

for (int path : arr.get(i)) {

if (path + d <= limit) set.add(path + d);

}

//最多选两条

for (int j = i + 1; j < arr.size(); j++) {

int d2 = root.dist.get(j);

for (int path1 : arr.get(i)) {

for (int path2 : arr.get(j)) {

if (path1 + path2 + d + d2 <= limit) set.add(path1 + path2 + d + d2);

}

}

}

}

return set;

}

}

```

### PDD7 数三角形

#### 题目描述

```

给出平面上的n个点，现在需要你求出，在这n个点里选3个点能构成一个三角形的方案有几种。

```

#### 输入描述:

```

第一行包含一个正整数n，表示平面上有n个点（n <= 100)

第2行到第n + 1行，每行有两个整数，表示这个点的x坐标和y坐标。(所有坐标的绝对值小于等于100，且保证所有坐标不同）

```

#### 输出描述:

```

输出一个数，表示能构成三角形的方案数。

```

#### 示例1

#### 输入

```

4

0 0

0 1

1 0

1 1

```

#### 输出

```

4

说明

4个点中任意选择3个都能构成三角形

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.Comparator;

public class Main {

static class Node {

int x, y;

Node(int x, int y) {

this.x = x;

this.y = y;

}

}

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

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

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

Node[] nodes = new Node[n];

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

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

nodes[i] = new Node(Integer.parseInt(line[0]), Integer.parseInt(line[1]));

}

int res = 0;

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

for (int j = i + 1; j < nodes.length - 1; j++) {

for (int k = j + 1; k < nodes.length; k++) {

if ((nodes[j].y - nodes[i].y) \* (nodes[k].x - nodes[i].x) != (nodes[k].y - nodes[i].y) \* (nodes[j].x - nodes[i].x)) {

res++;

}

}

}

}

System.out.println(res);

}

}

```

### PDD8 最大乘积

#### 题目描述

```

给定一个无序数组，包含正数、负数和0，要求从中找出3个数的乘积，使得乘积最大，要求时间复杂度：O(n)，空间复杂度：O(1)。

n<=1e5。

```

#### 输入描述:

```

第一行是数组大小n，第二行是无序整数数组A[n]

```

#### 输出描述:

```

满足条件的最大乘积

```

#### 示例1

#### 输入

```

4

3 4 1 2

```

#### 输出

```

24

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

//一组数字取三个，使其乘积最大

public class Main {

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

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

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

String[] data = in.readLine().split(" ");

long max1 = Integer.MIN\_VALUE;

long max2 = Integer.MIN\_VALUE;

long max3 = Integer.MIN\_VALUE;

long min1 = Integer.MAX\_VALUE;

long min2 = Integer.MAX\_VALUE;

for (String num : data) {

int temp = Integer.parseInt(num);

// 找最大的三个数

if (temp > max1) {

max3 = max2;

max2 = max1;

max1 = temp;

} else if (temp > max2) {

max3 = max2;

max2 = temp;

} else if (temp > max3) {

max3 = temp;

}

//找最小的两个数

if (temp < min1) {

min2 = min1;

min1 = temp;

} else if (temp < min2) {

min2 = temp;

}

}

long max = Integer.MIN\_VALUE;

max = Math.max(max, max1 \* max2 \* max3);

max = Math.max(max, max1 \* min1 \* min2);

System.out.println(max);

}

}

```

### PDD9 小熊吃糖

#### 题目描述

```

有n只小熊，他们有着各不相同的战斗力。每次他们吃糖时，会按照战斗力来排，战斗力高的小熊拥有优先选择权。前面的小熊吃饱了，后面的小熊才能吃。每只小熊有一个饥饿值，每次进食的时候，小熊们会选择最大的能填饱自己当前饥饿值的那颗糖来吃，可能吃完没饱会重复上述过程，但不会选择吃撑。

现在给出n只小熊的战斗力和饥饿值，并且给出m颗糖能填饱的饥饿值。

求所有小熊进食完之后，每只小熊剩余的饥饿值。

```

#### 输入描述:

```

第一行两个正整数n和m，分别表示小熊数量和糖的数量。（n <= 10, m <= 100）

第二行m个正整数，每个表示着颗糖能填充的饥饿值。

接下来的n行，每行2个正整数，分别代表每只小熊的战斗力和当前饥饿值。

题目中所有输入的数值小于等于100。

```

#### 输出描述:

```

输出n行，每行一个整数，代表每只小熊剩余的饥饿值。

```

#### 示例1

#### 输入

```

2 5

5 6 10 20 30

4 34

3 35

```

#### 输出

```

4

0

说明

第一只小熊吃了第5颗糖

第二只小熊吃了第4颗糖

第二只小熊吃了第3颗糖

第二只小熊吃了第1颗糖

```

```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[] line1 = br.readLine().trim().split(" ");

int n = Integer.parseInt(line1[0]); // 小熊数量

int m = Integer.parseInt(line1[1]); // 糖的数量

String[] line2 = br.readLine().trim().split(" ");

int[] sweets = new int[m]; // 每颗糖能填充的饥饿值

int[] eated = new int[m]; // 标识糖是否被吃掉

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

sweets[i] = Integer.parseInt(line2[i]);

}

// 糖按大小递增排序

Arrays.sort(sweets);

int[] energy = new int[n]; // 每只熊的战斗力

int[] hungry = new int[n]; // 每只熊的当前饥饿值

int[] full = new int[n]; // 标识该熊是否吃过了

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

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

energy[i] = Integer.parseInt(arr[0]);

hungry[i] = Integer.parseInt(arr[1]);

}

// 每次找剩下战斗力最大的熊

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

int max = Integer.MIN\_VALUE;

int p = -1;

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

if (full[j] == 0 && energy[j] > max) {

p = j;

max = energy[j];

}

}

full[p] = 1;

// 吃糖

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

if (hungry[p] >= sweets[j] && eated[j] == 0) {

hungry[p] -= sweets[j];

eated[j] = 1;

}

}

}

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

System.out.println(hungry[i]);

}

}

}

```

### PDD10 选靓号

#### 题目描述

```

A 国的手机号码由且仅由 N 位十进制数字(0-9)组成。一个手机号码中有至少 K 位数字相同则被定义为靓号。A 国的手机号可以有前导零，比如 000123456 是一个合法的手机号。

小多想花钱将自己的手机号码修改为一个靓号。修改号码中的一个数字需要花费的金额为新数字与旧数字之间的差值。比如将 1 修改为 6 或 6 修改为 1 都需要花 5 块钱。

给出小多现在的手机号码，问将其修改成一个靓号，最少需要多少钱？

```

#### 输入描述:

```

第一行包含2个整数 N、K，分别表示手机号码数字个数以及靓号至少有 K 个数字相同。

第二行包含 N 个字符，每个字符都是一个数字('0'-'9')，数字之间没有任何其他空白符。表示小多的手机号码。

数据范围：

2 <= K <= N <= 10000

```

#### 输出描述:

```

第一行包含一个整数，表示修改成一个靓号，最少需要的金额。

第二行包含 N 个数字字符，表示最少花费修改的新手机号。若有多个靓号花费都最少，则输出字典序最小的靓号。

```

#### 示例1

#### 输入

```

6 5

787585

```

#### 输出

```

4

777577

说明

花费为4的方案有两种：777577与777775，前者字典序更小。

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.Arrays;

public class Main {

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

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

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

int n = Integer.parseInt(strArr[0]), k = Integer.parseInt(strArr[1]);

char[] phoneNum = br.readLine().trim().toCharArray();

solve(n, k, phoneNum);

}

private static void solve(int n, int k, char[] phoneNum) {

int minCost = Integer.MAX\_VALUE;

char[] newPhoneNum = new char[n];

// 先得到原本号码中各数字出现的次数

int[] counter = getCounter(phoneNum);

for (int curNum = 0; curNum < 10; curNum++) {

// 依次计算将数字curNum改到k次需要多大的代价

if (counter[curNum] >= k) {

// 已经满足靓号要求

minCost = 0;

newPhoneNum = phoneNum;

break;

}

int curNumCost = 0;

int upperLimitNum = curNum + 1; // 将upperLimitNum改成curNum

int lowerLimitNum = curNum - 1; // 将lowerLimitNum改成curNum

char[] tempPhoneNum = Arrays.copyOfRange(phoneNum, 0, phoneNum.length);

// 还需要的重复次数

int remain = k - counter[curNum];

// 循环直至不再需要数字重复

while (remain > 0) {

// 比curNum大的数字从前往后改(尽量把前面的数改小，以保证字典序小)

if (upperLimitNum < 10) {

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

if (phoneNum[i] - '0' == upperLimitNum) {

curNumCost += upperLimitNum - curNum;

tempPhoneNum[i] = (char) (curNum + '0');

remain--;

}

}

}

// 比curNum小的数字从后往前改(尽量把后面的数改大，以保证字典序小)

if (lowerLimitNum >= 0) {

for (int i = n - 1; i >= 0 && remain > 0; i--) {

if (phoneNum[i] - '0' == lowerLimitNum) {

curNumCost += curNum - lowerLimitNum;

tempPhoneNum[i] = (char) (curNum + '0');

remain--;

}

}

}

// 如果改完了upperLimitNum和lowerLimitNum还没达到k次重复，就扩大上下限继续改

upperLimitNum++;

lowerLimitNum--;

}

if (curNumCost < minCost) {

minCost = curNumCost;

newPhoneNum = tempPhoneNum;

}

}

System.out.println(minCost);

System.out.println(newPhoneNum);

}

private static int[] getCounter(char[] phoneNum) {

int[] counter = new int[10];

for (char c : phoneNum) counter[c - '0']++;

return counter;

}

}

```

### PDD11 种树

#### 题目描述

```

小多想在美化一下自己的庄园。他的庄园毗邻一条小河，他希望在河边种一排树，共 M 棵。小多采购了 N 个品种的树，每个品种的数量是 Ai (树的总数量恰好为 M)。但是他希望任意两棵相邻的树不是同一品种的。小多请你帮忙设计一种满足要求的种树方案。

```

#### 输入描述:

```

第一行包含一个正整数 N，表示树的品种数量。

第二行包含 N 个正整数，第 i (1 <= i <= N) 个数表示第 i 个品种的树的数量。

数据范围：

1 <= N <= 1000

1 <= M <= 2000

```

#### 输出描述:

```

输出一行，包含 M 个正整数，分别表示第 i 棵树的品种编号 (品种编号从1到 N)。若存在多种可行方案，则输出字典序最小的方案。若不存在满足条件的方案，则输出"-"。

```

#### 示例1

#### 输入

```

3

4 2 1

```

#### 输出

```

1 2 1 2 1 3 1

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.Comparator;

public class Main {

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

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

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

int[] arr = new int[n];

int cnt = 0, max = 0;

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

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

arr[i] = Integer.parseInt(s[i]);

cnt += arr[i];

max = Math.max(max, arr[i]);

}

//如果有树的个数>(总个数+1)/2 就不存在序列

if (max > (cnt + 1) >> 1) {

System.out.print("-");

return;

}

StringBuilder sb = new StringBuilder();

int pre = -1;

while (cnt > 0) {

int preCnt = cnt;

//如果有树的个数大于剩余总数的1/2，选这种树

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

if (arr[i] > cnt >> 1) {

arr[i]--;

pre = i;

sb.append(i + 1);

sb.append(" ");

cnt--;

break;

}

}

if (preCnt != cnt) continue;

//如果没有，选择序号最小的 还有剩余的 上次没选的树

int j = 0;

while (pre == j || arr[j] == 0) j++;

arr[j]--;

pre = j;

sb.append(j + 1);

sb.append(" ");

cnt--;

}

sb.deleteCharAt(sb.length() - 1);

System.out.print(sb);

}

}

```

### PDD12 两两配对差值最小

#### 题目描述

```

给定一个长度为偶数的数组arr，将该数组中的数字两两配对并求和，在这些和中选出最大和最小值，请问该如何两两配对，才能让最大值和最小值的差值最小？

```

#### 输入描述:

```

一共2行输入。

第一行为一个整数n，2<=n<=10000, 第二行为n个数，组成目标数组，每个数大于等于2，小于等于100。

```

#### 输出描述:

```

输出最小的差值。

```

#### 示例1

#### 输入

```

4

2 6 4 3

```

#### 输出

```

1

示例2

输入

6

11 4 3 5 7 1

```

#### 输出

```

3

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

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

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

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

int[] nums = new int[len];

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

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

nums[i] = Integer.parseInt(p[i]);

}

Arrays.sort(nums);

int left = 0;

int right = len - 1;

int min = 200;

int max = 4;

while (left < right) {

min = Math.min(nums[left] + nums[right], min);

max = Math.max(nums[left] + nums[right], max);

left++;

right--;

}

System.out.println(max - min);

}

}

```

### PDD13 回合制游戏

#### 题目描述

```

你在玩一个回合制角色扮演的游戏。现在你在准备一个策略，以便在最短的回合内击败敌方角色。在战斗开始时，敌人拥有HP格血量。当血量小于等于0时，敌人死去。一个缺乏经验的玩家可能简单地尝试每个回合都攻击。但是你知道辅助技能的重要性。

在你的每个回合开始时你可以选择以下两个动作之一：聚力或者攻击。

聚力会提高你下个回合攻击的伤害。

攻击会对敌人造成一定量的伤害。如果你上个回合使用了聚力，那这次攻击会对敌人造成buffedAttack点伤害。否则，会造成normalAttack点伤害。

给出血量HP和不同攻击的伤害，buffedAttack和normalAttack，返回你能杀死敌人的最小回合数。

```

#### 输入描述:

```

第一行是一个数字HP

第二行是一个数字normalAttack

第三行是一个数字buffedAttack

1 <= HP,buffedAttack,normalAttack <= 10^9

```

#### 输出描述:

```

输出一个数字表示最小回合数

```

#### 示例1

#### 输入

```

13

3

5

```

#### 输出

```

5

```

```java

import java.io.\*;

public class Main {

public static int res = 0;

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

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

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

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

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

int r1 = hp / na + ((hp % na == 0) ? 0 : 1);

int r2 = hp / ba \* 2;

if (hp % ba != 0) {

if (hp % ba <= na) r2++;

else r2 += 2;

}

System.out.println(Math.min(r1, r2));

}

}

```