第一题[编程题 25 分]： 闹钟叫醒去上课

时间限制：C/C++ 1 秒，其它语言 2 秒
空间限制：C/C++ 32768K，其它语言 65536K
64bit IO Format: %lld

题目描述：

小明总是睡过头，所以他定了很多闹钟，只有在闹钟响的时候他才会醒过来并且决定起不起来。
从他起床算起他需要 X 分钟到达教室，上课时间为当天的 A 时 B 分，请问他最晚可以什么时候起床。

输入描述：

每个输入包含一个测试用例
每个测试用例的第一行包含一个整数，表示闹钟的数量 N（N <= 100）
接下来的 N 行每行包含两个整数，表示这个闹钟响起的时间为 Hi（0 <= A < 24）时 Mi（0 <= B < 60）分
接下来的一行包含一个整数，表示从他起床算起他需要 X（0 <= X <= 100）分钟到达教室
接下来的一行包含两个整数，表示上课时间为 A（0 <= A < 24）时 B（0 <= B < 60）分
数据保证至少有一个闹钟可以让牛牛及时到达教室

输出描述：
输出两个整数表示牛牛最晚起床时间

示例 1：（输入输出示例仅供调试，后台判题数据一般不包含示例）

输入：

3
5 0
6 0
7 0
59
6 59
输出：

6 0
思路一：创建一个时间对象，包含 Hi 和 Mi，然后将闹钟时间封装成这个时间对象，然后将这些对象排序（快排时间复杂度为 O（nlog2n）O（nlog_2n）O（nlog

n)），然后就简单了，我开始是直接从结尾遍历，找到第一个满足条件的就输出（条件：闹钟时间 + X <= 上课时间），然后终止，然而，并没有通过全部样例，清醒一下，想了一下，有序的序列，查找数据明摆着用二分，就是查找一个序列中最右、最接近标杆的，比如一个序列{1, 2, 3, 3, 3, 4, 10, 10, 11}，当标杆为 3 时，我们应该要下标为 5 的那个 3，当标杆为 5 时，我们应该要下标为 6 的那个 4。

AC 代码：

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        run();
    }

    public static void run() {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt(); //  闹钟数量
        ArrayList<Time> list = new ArrayList<>();
        for (int i = 0; i < N; ++i) {
            list.add(new Time(scanner.nextInt(), scanner.nextInt())); //  封装成 Time 对象
        }
        list.sort(new Comparator<Time>() { //  将闹钟们排序
            @Override
            public int compare(Time o1, Time o2) {
                return o1.hi - o2.hi == 0 ? o1.mi - o2.mi : o1.hi - o2.hi;
            }
        });
        int X = scanner.nextInt(), A = scanner.nextInt(), B = scanner.nextInt();
        Time AB = new Time(A, B); //  上课时间
        AB.jian(X);
        int l = 0, r = list.size() - 1;
        while (l <= r) {    //  二分搜索
            int m = (l + r) >> 1;
            if (AB.compareTo(list.get(m)) >= 0) {
                l = m + 1;
            } else {
                r = m - 1;
            }
        }
        if (r < 0) // r 可能小于 0，因为题目说了保证有解，那就输出第一个闹钟就可以了
            System.out.println(list.get(0));
        else
            System.out.println(list.get(r));
        scanner.close();
    }

    static class Time implements Comparable<Time> { // Time 对象，实现 Comparable 接口，方便比较
```

```
        int hi;
        int mi;

        public Time(int hi, int mi) {
            this.hi = hi;
            this.mi = mi;
        }

        public Time jian(int X) { // 时间的减法，6 时 30 分 - 10 分 = 6 时 20 分
            int T = hi * 60 + mi;
            int cha = T - X;
            hi = cha / 60;
            mi = cha % 60;
            return this;
        }

        @Override
        public int compareTo(Time o) {
            return hi - o.hi == 0 ? mi - o.mi : hi - o.hi; // 按照小时升序，当小时相同时，按照分钟升序
        }

        @Override
        public String toString() {
            return hi + " " + mi;
        }
    }
}
```

思路二：由于上面创建了很多闹钟对象，比较费空间，所以我们可以将输入的时间全部转换成分钟，比如 6 时 10 分=6*60+10=370 分，然后将输入的时间排序（快排时间复杂度为 O（nlog2n）O（nlog_2n）O（nlog 2

n）），然后二分，思路和上面一样。

AC 代码：

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        run();
    }
```

```
    public static void run() {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 0; i < N; ++i) {
            list.add(scanner.nextInt() * 60 + scanner.nextInt());
        }
        list.sort(new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o1 - o2;
            }
        });
        int X = scanner.nextInt();
        int A = scanner.nextInt();
        int B = scanner.nextInt();
        int AB = A * 60 + B; // 上课时间，转换成了分钟
        int t = AB - X; // 这就是标杆
        int l = 0, r = list.size() - 1;
        while (l <= r) {
            int m = (l + r) >> 1;
            if (t >= list.get(m)) {
                l = m + 1;
            } else {
                r = m - 1;
            }
        }
        if (r < 0)
            System.out.println(list.get(0) / 60 + " " + list.get(0) % 60);
        else
            System.out.println(list.get(r) / 60 + " " + list.get(r) % 60);
        scanner.close();
    }
}
```

第二题[编程题 25 分]： 秘密通信

时间限制：C/C++ 1 秒，其它语言 2 秒
空间限制：C/C++ 32768K，其它语言 65536K
64bit IO Format: %lld

题目描述：

小明和安琪是好朋友。最近，他们的谈话被一家侦探机构监控，所以他们想将他们的谈话内容进行加密处

理。
于是，他们发明了一种新的加密方式。每条信息都被编译成二进制数 B（明文），其长度为 N。
然后该信息被写下 K 次，每次向右移动 0，1，…，K-1 位。
例如：B = 1001010，K=4
1001010
 1001010
  1001010
   1001010
然后对每一列进行异或操作，并且把最终所得的结果记录下来，我们将该数称为 S（密文）。
例如上述例子的结果为：1110100110。
最后，将编码的信息 S 和 K 发送给安琪。
小明已经实现了这种编码的加密过程，但他要求安琪写一个程序去实现这种编码的解密过程，你能帮助安琪实现解密过程吗？

输入描述：
第一行输入两个整数 N 和 K
第二行输入一个二进制字符串 S，长度是 N + K - 1

输出描述：
输出明文 B

示例 1：（输入输出示例仅供调试，后台判题数据一般不包含示例）

输入：
7 4
1110100110
输出：
1001010

示例 2：（输入输出示例仅供调试，后台判题数据一般不包含示例）

输入：
6 2
1110001

输出：
101111

备注：

1 <= N <= 10^6
1 <= K <= 10^6

这个题目没啥算法，就是纯粹找规律，先来看示例一：1110100110，它是这样来的：

```
  1001010
   1001010
    1001010
^     1001010
-----------
 1110100110
```

我们把它抽象出来：

```
  abcdefg
   abcdefg
    abcdefg
^     abcdefg
-----------
 hijklmnopq
```

这样就很容易看出规律了（opq 用不上）：

```
h = a
i = a ^ b
j = a ^ b ^ c
k = a ^ b ^ c ^ d
l = b ^ c ^ d ^ e
m = c ^ d ^ e ^ f
n = d ^ e ^ f ^ g
```

我们知道，1 ^ 1 = 0 ^ 0 = 0, 1 ^ 0 = 0 ^ 1 = 1，即相同异或值为 0，不同就为 1，0 与 n 异或结果为 n，而上面的等式我们是已知左边的（hjiklmn），我们可以反过来，根据异或的性质，i = a ^ b 可以推出 b = i ^ a（两边同时异或 a 即可得到）：

```
h = a                   =>          a = h
i = a ^ b               =>          b = i ^ a
j = a ^ b ^ c           =>          c = j ^ a ^ b
k = a ^ b ^ c ^ d       =>          d = k ^ a ^ b ^ c
l = b ^ c ^ d ^ e       =>          e = l ^ b ^ c ^ d
m = c ^ d ^ e ^ f       =>          f = m ^ c ^ d ^ e
n = d ^ e ^ f ^ g       =>          g = n ^ d ^ e ^ f
```

到这里就很简单了，我们把 abcdfg 分成三部分来看，0，1 ~ k-1，K ~ N-1，，因为这三段执行过程是不一样的，我们设 int[] S 为输入的 01 数字（对应 hijklmnopq），int[] res 为我们要输出的答案（对应 abcdefg）：

```
0：res[0] = S[0]
1 ~ k-1：res[i] = S[i] ^ res[0] ^  •••  ^ res[i - 1]
```

K ~ N-1：res[i] = S[i] ^ res[i + 1 - K] ^ ••• ^ res[i - 1]
实现代码：

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Scanner;

public class Main {
	public static void main(String[] args) {
		run();
	}
	private static void run1() {
		Scanner scanner = new Scanner(System.in);
		int N = scanner.nextInt();
		int K = scanner.nextInt();
		scanner.nextLine();
		String S = scanner.nextLine();
		int[] S_int = new int[N + K - 1]; // 将输入的 01 字符串转化为 01 整数数组，方便下面进行异或操作
		int[] res = new int[N]; // 输出的结果
		for (int i = 0; i < N + K - 1; ++i) { // 将输入的 01 字符串转化为 01 整数数组，方便下面进行异或操作
			S_int[i] = S.charAt(i) - '0';
		}
		res[0] = S_int[0]; // 第一段
		for (int i = 1; i < K; ++i) { // 第二段
			res[i] = S_int[i] ^ yihuo(0, i - 1, res);
		}
		for (int i = K; i < N; i++) { // 第三段
			res[i] = S_int[i] ^ yihuo(i + 1 - K, i - 1, res);
		}
		for (int i = 0, len = res.length; i < len; ++i) {
			System.out.print(res[i]);
		}
		System.out.println();
		scanner.close();
	}

	private static int yihuo(int i, int j, int[] res) { // 返回从 i 异或到 j 的值
		int result = res[i];
		for (int t = i + 1; t <= j; t++) {
			result ^= res[t];
		}
		return result;
	}
}
```

好了，代码写到这里，jj 了，只通过了 83.33%测试用例，想了一会，发现求当前异或的时候不必再循环求，而是与上一个异或值有关的，发现规律，还可以优化，比如求 c = j ^ a ^ b 的时候，b 是上一步的结果（b = i ^ a），我们带入这个式子，可得 c = j ^ a ^ i ^ a，再化简，可得 c = i ^ j，求 e = l ^ b ^ c ^ d 的时候，将 d = k ^ a ^ b ^ c 带入：

0：res[0] = S[0]
1 ~ k-1：res[i] = S[i] ^ res[0] ^ ••• ^ res[i - 1]
= S[i] ^ res[i - 1] ^ S[i - 1] ^ res[i - 1]
= S[i] ^ S[i - 1]
K ~ N-1：res[i] = S[i] ^ res[i + 1 - K] ^ ••• ^ res[i - 1]
= S[i] ^ res[i - 1] ^ res[i - K] ^ S[i - 1] ^ res[i - 1]
= S[i] ^ res[i - K] ^ S[i - 1]
AC 代码：

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Scanner;

public class Main {
	public static void main(String[] args) {
		run();
	}
	private static void run() {
		Scanner scanner = new Scanner(System.in);
		int N = scanner.nextInt();
		int K = scanner.nextInt();
		scanner.nextLine();
		String S = scanner.nextLine();
		int[] S_int = new int[N + K - 1]; // 将输入的 01 字符串转化为 01 整数数组，方便下面进行异或操作
		int[] res = new int[N]; // 输出的结果
		for (int i = 0; i < N + K - 1; ++i) { // 将输入的 01 字符串转化为 01 整数数组，方便下面进行异或操作
			S_int[i] = S.charAt(i) - '0';
		}
		res[0] = S_int[0]; // 第一段
		for (int i = 1; i < K; ++i) { // 第二段
			res[i] = S_int[i - 1] ^ S_int[i];
		}
		for (int i = K; i < N; i++) { // 第三段
			res[i] = res[i - K] ^ S_int[i - 1] ^ S_int[i];
		}
		for (int i = 0, len = res.length; i < len; ++i) {
			System.out.print(res[i]);
		}
```

```
        System.out.println();
        scanner.close();
    }
```

第三题[编程题 25 分]： 万万没想到之抠门的老板

时间限制：C/C++ 1 秒，其它语言 2 秒
空间限制：C/C++ 32768K，其它语言 65536K
64bit IO Format: %lld

题目描述：

我叫王大锤，是一家互联网公司的老板，快到年底了，要给员工发奖金。
真头疼，大环境这么差，怎么才能尽可能的少发点、同时还能让大家怨气少一点呢？
公司的座位是排成一排的，每个人都最多打听的到和自己相邻左右两个人的奖金数，我决定这样发：

1. 每个人都至少发 100 块。
2. 论资排辈：每个人加入公司的年限是公开的，如果一个员工 A 加入公司的时间比领座的同事 B 早，那 A 至少比 B 多拿 100 块。这样，他的心理会平衡一些。

我特喵是个天才！将人性理解的如此透彻，做一个小公司的老板真是屈才了。
……
万万没想到，发完奖金，所有员工都离职了，都跳槽去了一家叫字节跳动的公司，他们都说这家公司一不论资排辈，二不吃大锅饭，
奖罚分明，激励到位，老板还特大方，说的我都想去应聘了.....

请听题：给定大锤公司员工的座位表，以及每个员工的入职时间，计算大锤最少需要发多少奖金。

输入描述：

第一行只有一个正整数 N （1 <= N <= 1000），表示员工人数
第二行有 N 个正整数，代表每个员工的入职年限。排列顺序即为员工的座位顺序。

输出描述：

一个数字，代表大锤最少需要发的奖金总数。

示例 1：（输入输出示例仅供调试，后台判题数据一般不包含示例）

输入：
4
3 9 2 7
输出：

600

说明：
每人奖金数为（100，200，100，200）

示例 2：（输入输出示例仅供调试，后台判题数据一般不包含示例）

输入：
5
1 2 3 4 5

输出：
1500

说明：

每人奖金数为（100，200，300，400，500）

这个题目的话，直接从左往右遍历，遍历到每个人的时候，再向左向右遍历，向左遍历的时候，如果当前值比右边一个大，则当前值需要取 max（后一个值+100，当前值），如果小于右边的，则停止向左试探；向右也是同理。

当然，还有更好的解法，移步 leetcode 135：https://leetcode-cn.com/problems/candy/

AC 代码：

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        run();
    }

    private static void run() {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        int[] people = new int[N];
        for (int i = 0; i < N; ++i)
            people[i] = scanner.nextInt();
        int[] money = new int[N];
        Arrays.fill(money, 100);
        for (int i = 0; i < N; ++i) {
            for (int j = i - 1; j >= 0; --j) {
```

```
                if (people[j] < people[j + 1])
                    break;
                money[j] = Math.max(money[j + 1] + 100, money[j]);
            }
            for (int j = i + 1; j < N; ++j) {
                if (people[j] < people[j - 1])
                    break;
                money[j] = Math.max(money[j - 1] + 100, money[j]);
            }
        }
        int sum = 0;
        for (int i = 0; i < N; ++i)
            sum += money[i];
        System.out.println(sum);
        scanner.close();
    }
}
```

第四题[编程题 25 分]： 跑步

时间限制：C/C++ 1 秒，其它语言 2 秒
空间限制：C/C++ 32768K，其它语言 65536K
64bit IO Format: %lld

题目描述：

小明练习跑步，他家附近的街道是棵树，这棵树上的点按 1 到 n 标号，任意两点互相可达，并且有且仅有一条路，每条路的距离都是 1，
需要在树上找一条路来跑，小明对 3 很感兴趣，所以他想知道所有跑道距离和%3=0，1，2 的道路总长度一共各有多长。
即树上任意两点间距离%3=k 的距离和。

输入描述：

第一行一个 n，点数 n <=1e5
接下来 n - 1 行每行 u，v 一条无向边

输出描述：
一行 3 个整数，分别代表%3=0，1，2 的两点距离的距离和
结果取模 1e9 + 7

示例 1：（输入输出示例仅供调试，后台判题数据一般不包含示例）

输入：
3

1 2
2 3
输出：
0 2 2

说明：

长度%3=0 的距离不存在，=1 的有两条 1-2，2-3 总长度是 2，=2 的有 1 条，1-3，总长度是 2

备注：
前 4 个 case 小数据点数 3，10，100，10000
之后数据全部 100000 个点
——————————————————————