链滴

# 2023 携程春招笔试

作者：Hildaquan
原文链接：https://ld246.com/article/1678239807689
来源网站：链滴

昨天笔试录屏，做完后破防了。。。

今天上午回顾一下题目：

# 1. 简单dp

A了，就不写代码了

录屏截屏

```
    int n = nums.length;
    if (n <= 1) return 0;

    // dp[i]: i之前的最长连续稳定子数组
    int[] dp = new int[n];
    dp[0] = 0;

    int max = 0;
    for (int i = 1; i < n; i++) {
        if (Math.abs(nums[i] - nums[i - 1]) <= 1) {
            dp[i] = dp[i - 1] + 1;
        }
        if (dp[i] > max) max = dp[i];
    }

    return max + 1;
}
```

# 2. 字符串拼接

简单模拟题，直接字符串拼接搞定

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

        int n = in.nextInt(), q = in.nextInt();
        String str = in.next();
        int[][] ranges = new int[q][2];
        for (int i = 0; i < q; i++) {
            ranges[i][0] = in.nextInt() - 1;
            ranges[i][1] = in.nextInt() - 1;
        }

        System.out.println(solution(str, ranges));
    }
```

```java
private static String solution(String str, int[][] ranges) {
    int q = ranges.length;

    int l, r;
    for (int[] range : ranges) {
        l = range[0];
        r = range[1];

        String left = str.substring(0, l);
        String mid = str.substring(l, r + 1);
        String right = str.substring(r + 1);

        mid = change(mid);

        str = left + mid + right;
    }

    return str;
```

```java
private static String change(String str) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        sb.append(c + c);
    }
    return sb.toString();
}
```

# 3. 高数求导求极值。。。

想复杂了。。。耗费了不少时间。。。

**题目描述**

游游准备开车出游，她的车非常特殊，油越多则最高速度越快，即最高速度和油量是成正比的。另外，行驶过程中油是不会消耗的。
已知游游的车初始的最高速度为$v_0$，当游游花费了$t$时间加油时，车的最高速度会变成$v_0 + t * x$。
游游开车的总里程为$y$，假设游游始终以最高速度行驶（即忽略加速时间），游游想知道，自己最少花费多少时间可以完成出游？

**输入描述:**

> 三个整数$v_0, x, y$，用空格隔开。
> $0 \le v_0 \le 10^9$
> $1 \le x, y \le 10^9$

**输出描述:**

> 一个浮点数，代表最终花费的总时间。如果你的答案和标准答案的相对误差不超过$10^{-6}$，则认为答案正确。

# 解方程

$$T = \frac{y}{v_0 + xt} + t \longrightarrow$$ 不一开始就加油

$$T' = 1 - \frac{yx}{(v_0 + xt)^2} = 0$$

$$\therefore (v_0 + xt)^2 = yx \quad \therefore t = \frac{\sqrt{xy} - v_0}{x}$$

代入t回原方程

$$T = \frac{y}{\sqrt{yx}} + \frac{\sqrt{yx} - v_0}{x} = 2\sqrt{\frac{y}{x}} - \frac{v_0}{x}$$

```
import java.util.Scanner;

/**
 * Description:
 * date: 2023/03/08 上午 9:25
 *
 * @author Quan
 */
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        int v0 = in.nextInt(), x = in.nextInt(), y = in.nextInt();

        double ans =  2 * Math.sqrt(y / x) - (v0 / x * 1.0);
        System.out.println(ans);
    }
}

// 0 1 2
// 2.8284271247461903
```

# 4. 状态 dp + 01 背包

游游正在逛超市，有$n$个商品摆成一排，第$i$个商品的价格为$a_i$，游游对它的喜爱度为$b_i$。所有商品的价格都是偶数。
超市开展了一个活动，当游游**花费原价**买了一件商品时，她可以用半价买下一件右边相邻的商品（也可以用原价购买，这样该商品右边的商品就有一次享受半价的机会）。但如果游游半价购买了一件商品，那么下一件右边相邻的商品只能原价购买。
换言之，如果游游想要半价买某一件商品，必须先用原价买下它相邻的左边的那个商品。
游游初始的钱为$x$，她想要买的商品的喜爱度总和尽可能大，但总价格不能超过$x$。你能帮帮她计算最大的喜爱度总和吗？

**输入描述:**

> 第一行输入两个正整数$n$和$x$，分别代表商品的数量，以及游游初始的金额数。
> 第二行输入$n$个正整数$a_i$，分别代表每个商品的价格。
> 第三行输入$n$个正整数$b_i$，分别代表每个商品可以给游游带来的喜爱度。
> $1 \le n, x, a_i \le 1000$
> $1 \le b_i \le 10^9$

耗费了 1 小时，早知道不去做这道题，专心攻克第三道算了。。。

```
import java.util.Scanner;

/**
 * Description:
 * date: 2023/03/07 下午 10:09
 *
 * @author Quan
 */
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        int n = in.nextInt(), x = in.nextInt();
        int[] A = new int[n + 1];
        int[] B = new int[n + 1];
        for (int i = 1; i <= n; i++) A[i] = in.nextInt();
        for (int i = 1; i <= n; i++) B[i] = in.nextInt();

        long[][][] dp = new long[n + 1][x + 1][3];
        // dp[i][j][0] - 不买
        // dp[i][j][1] - 原价买
        // dp[i][j][2] - 半价买

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

```
            // 当前半价买，前一个必须原价买
            if (i > 1 && j - A[i] / 2 >= A[i - 1]) {
                // j - A[i] / 2 >= A[i - 1]：剩下的钱要能够买前一个
                dp[i][j][2] = dp[i - 1][j - A[i] / 2][1] + B[i];
            }

            // 原价买
            if (j >= A[i]) {
                // 要么不买，要么在原价和半价中选择最大的
                dp[i][j][1] = Math.max(dp[i - 1][j - A[i]][0],
                        Math.max(dp[i - 1][j - A[i]][1], dp[i - 1][j - A[i]][2]) + B[i]
                );
            }

            // 上面的是01背包逻辑，下面则是状态转移，记录最大的数值
            dp[i][j][0] = Math.max(dp[i - 1][j][0], Math.max(dp[i - 1][j][1], dp[i - 1][j][2]));
            dp[i][j][1] = Math.max(dp[i - 1][j][1], dp[i][j][1]);
            dp[i][j][2] = Math.max(dp[i - 1][j][2], dp[i][j][2]);
        }
    }

    System.out.println(max(dp[n][x][0], dp[n][x][1], dp[n][x][2]));
  }

  private static long max(long a, long b, long c) {
    return Math.max(a, Math.max(b, c));
  }
}

// 输入
4 7
2 2 6 2
3 4 5 1

// 输出
12
```