习题课

T E A C H I N G C O U R S E W A R E P O W P O I N T

授课时间：2025.08.02

排序：

首先将每天的柠檬水制作消耗和天数按照消耗从小到大排序，如果消耗相同则按天数从大到小排序。

按排序后的顺序遍历每一天，维护两个变量：当前处理到的最大天数和当前拥有的柠檬数量。

对于每一天，如果其天数大于当前处理到的最大天数，则更新为当前天数，并计算可以制作的柠檬水杯数。通过贪心策略，优先选择消耗低的天数制作柠檬水，从而最大化柠檬水的总制作数量。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// 存储每天的柠檬消耗和对应日期
struct node {
    int cost; // 制作1杯柠檬水的消耗
    int day; // 对应的天数 (1-based)
}e[1000005];
bool cmp(node a, node b)
{
    if (a.cost == b.cost) {
        return a.day > b.day; // 消耗相同则日期大的排前面
    }
    return a.cost < b.cost; // 消耗小的排前面
}
```

柠檬水

```
int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> e[i].cost;
        e[i].day = i;
    }
    sort(e + 1, e + n + 1, cmp);
    int last_day = 0; // 上一次处理的最晚日期
    int total_used = 0; // 累计使用的柠檬数
    int result = 0; // 总共制作的柠檬水
    // 遍历排序后的日期, 计算最大制作量
    for (int i = 1; i <= n; ++i) {
        int current_cost = e[i].cost;
        int current_day = e[i].day;
        // 只处理未覆盖的日期区间
        if (current_day > last_day) {
            last_day = current_day;
            // 计算当前区间可制作的杯数: (当前日期累计柠檬数 - 已使用柠檬数) / 消耗
            int can_make = (current_day - total_used) / current_cost;
            result += can_make;
            // 更新累计使用的柠檬数
            total_used += can_make * current_cost;
        }
    }

    cout << result << endl;
    return 0;
}
```

目录

- PART-01 题目描述 TEACHING COURSEWARE
- PART-02 知识回归 TEACHING COURSEWARE
- PART-03 解题思路 TEACHING COURSEWARE
- PART-04 标准程序 TEACHING COURSEWARE

01

题目描述

TEACHING
COURSEWARE

TEACH

题目描述

题目描述

你有 n 个赌场，编号从 1 到 n 。每个赌场由三个整数描述： l_i 、 r_i 和 $real_i$ ($l_i \leq real_i \leq r_i$)。你最初有 k 枚硬币。你只有在当前硬币数 x 满足 $l_i \leq x \leq r_i$ 时，才能在第 i 个赌场玩。玩完后，你的硬币数会变为 $real_i$ 。你可以以任意顺序访问这些赌场，并且不要求必须访问所有赌场。每个赌场最多只能访问一次。你的任务是通过合理选择访问赌场的顺序，使最终获得的硬币数最大。

输入格式

只有一组测试用例。每个测试用例的第一行包含两个整数 n 和 k ($1 \leq n \leq 10^5$, $0 \leq k \leq 10^9$)，分别表示赌场数量和初始硬币数。接下来的 n 行，每行包含三个整数 l_i 、 r_i 、 $real_i$ ($0 \leq l_i \leq real_i \leq r_i \leq 10^9$)，表示第 i 个赌场的参数。保证所有测试用例中 n 的总和不超过 10^5 。

输出格式

对于每个测试用例，输出一个整数，表示在最优访问顺序下你最终能获得的最大硬币数。

输入输出样例 #1

输入 #1

```
3 1
2 3 3
1 2 2
3 10 10
```

Copy

输出 #1

```
10
```

Copy

说明/提示

你可以先去第 2 个赌场，此时你有 2 枚硬币。然后可以去第 1 个赌场，硬币数增加到 3。最后去第 3 个赌场，最终硬币数为 10，这是最大可能的数量。

02

知识回归

TEACHING
COURSEWARE

TEACH

贪心算法是一种在**每一步**决策时，都选择**当前状态下最优**（即最有利）的选项，且**不回溯**修改之前决策的算法。

它的核心思路可以概括为：“走一步看一步，每一步都选当下最好的，**不考虑未来**可能的变化，也**不后悔之前**的选择”。

重要性质：**“无后效性”**

关键性质：

贪心选择性质：局部最优能推出全局最优。

局部和全局

03

解题思路

TEACHING
COURSEWARE

TEACH

通过合理选择访问赌场顺序以最大化最终硬币数的问题。

核心思路是贪心策略结合区间扩展：首先将每个赌场的有效区间定义为左边界 l_i 到右边界

$\min(r_i, \text{real}_i)$ （仅保留能使硬币数不减少的有效范围），并按左边界升序排序；

然后从初始硬币数 k 出发，寻找第一个包含 k 的有效区间，再沿排序后的区间依次扩展——每次将当前硬币数更新为区间的最大右边界，直到无法覆盖下一个区间的左边界为止，最终得到的就是能达到的最大硬币数。这种方法通过优先扩展可衔接的区间，确保每次更新都能最大化后续访问的可能性，高效找到最优解。

04

标准程序

TEACHING
COURSEWARE

TEACH

标准程序

```
#include <iostream>
#include <algorithm>
using namespace std;
// 定义结构体存储赌场信息
struct node {
    int l, s; // l为左边界, s为有效右边界(min(r_i, real_i))
};
// 排序比较函数: 按l升序, l相同则按s升序
bool cmp(node p1, node p2) {
    if (p1.l == p2.l) {
        return p1.s < p2.s;
    }
    return p1.l < p2.l;
}
const int MAX_N = 1e5 + 10; // 最大赌场数量
node a[MAX_N]; // 存储赌场数组
```

```
int main() {  
  
    int n, k;  
    cin >> n >> k; // 读入赌场数量和初始硬币数  
  
    // 读入每个赌场的参数并计算有效右边界  
    for (int i = 1; i <= n; i++) {  
        int x, y, z;  
        cin >> x >> y >> z;  
        a[i].l = x;  
        a[i].s = min(y, z); // 有效右边界为min(r_i, real_i)  
    }  
}
```

```
// 按自定义规则排序赌场
sort(a + 1, a + n + 1, cmp);
int ans = k; // 初始硬币数
// 寻找可扩展的区间并更新最大硬币数
for (int i = 1; i <= n; i++) {
    // 当前硬币数在有效区间内时, 尝试扩展
    if (ans >= a[i].l && ans <= a[i].s) {
        for (int j = i; j <= n; j++) {
            ans = max(ans, a[j].s); // 扩展到当前区间的最大右边界
            // 无法继续扩展时输出结果
            if (ans < a[j + 1].l || j == n) {
                cout << ans << endl;
                return 0;
            }
        }
    }
}
// 若没有可扩展的区间, 输出初始值
cout << ans << endl;
return 0;
}
```