区间贪心

TEACHING COURSEWARE POWPOINT

授课时间：2025.07.30

目录

01

概念引入

TEACHING
COURSEWARE

TEACH

概念引入

区间类的贪心问题是一类围绕“区间”展开的优化问题，核心是通过对区间的合理选择、排序或调整，找到满足特定条件的最优解（如最多数量、最少资源等）。这类问题的共同特征是涉及区间的起点和终点，且贪心策略往往依赖于对区间的排序（按起点或终点）。

02

题型分析

TEACHING
COURSEWARE

TEACH

题型分析

1. 区间选点
问题描述：给定n个区间，要求在数轴上放置最少的点，使得每个区间内至少包含一个点（点可以在区间的端点上）。
核心目标：最小化点的数量。
关键特征：用最少的“标记点”覆盖所有区间，点的位置需尽可能覆盖多个区间。

题型分析

题目描述

给定 `n` 个闭区间 `[a_i, b_i]`，请你在数轴上选择最少的点，使得每个区间内至少包含一个选出的点。

请问最少需要选择多少个点?

输入格式

第一行包含整数 `n`，表示区间数量。

接下来 `n` 行，每行包含两个整数 `a_i` 和 `b_i`，表示一个区间的两个端点。

输出格式

输出一个整数，表示所需的最少点的数量。

数据范围

1 ≤ `n` ≤ 10^5

-10^9 ≤ `a_i` ≤ `b_i` ≤ 10^9

输入样例

Copy

```
3
-1 1
2 4
3 5
```

输出样例

Copy

```
2
```

题型分析

贪心策略
按右端点排序：将所有区间按照右端点从小到大排序

贪心选点：从第一个区间开始，每次选择当前区间的右端点作为选点位置

跳过覆盖区间：如果后续区间的左端点 ≤ 当前选点位置，说明该区间已被覆盖，跳过

新增选点：遇到未被覆盖的区间时，选择其右端点作为新选点，计数增加

题型分析


```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 100010;
struct node {
    int l, r;
} a[N];
bool cmp(node a, node b) {
    return a.r < b.r;
}
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i].l >> a[i].r;
    }
    sort(a, a + n, cmp);  // 对数组a排序
    int cnt = 0;          // 记录选点数量
    int ed = -2e9;        // 当前最后一个选点位置（初始化为负无穷）
    for (int i = 0; i < n; i++) {
        // 如果当前区间左端点大于上一个选点位置，需要新增选点
        if (a[i].l > ed) {
            cnt++;
            ed = a[i].r;  // 选择当前区间的右端点作为新点
        }
    }
    cout << cnt << endl;
    return 0;
}
```

题型分析

2. 区间不重叠最大化
问题描述：给定n个区间，要求选择尽可能多的区间，使得选中的区间两两不重叠（即任意两个区间没有交集）。
核心目标：最大化选中的区间数量。
关键：区间之间存在重叠，在其中筛选出最多的无重叠区间。

题型分析

题目描述

现在各大 oj 上有 n 个比赛，每个比赛的开始、结束的时间点是知道的。

yyy 认为，参加越多的比赛，noip 就能考的越好（假的）。

所以，他想知道他最多能参加几个比赛。

由于 yyy 是蒟蒻，如果要参加一个比赛必须善始善终，而且不能同时参加 2 个及以上的比赛。

输入格式

第一行是一个整数 n，接下来 n 行每行是 2 个整数 a_i, b_i $(a_i < b_i)$，表示比赛开始、结束的时间。

输出格式

一个整数最多参加的比赛数目。

输入输出样例

输入 #1 复制

```
3
0 2
2 4
1 3
```

输出 #1 复制

```
2
```

说明/提示

- 对于 20% 的数据，$n \leq 10$；
- 对于 50% 的数据，$n \leq 10^3$；
- 对于 70% 的数据，$n \leq 10^5$；
- 对于 100% 的数据，$1 \leq n \leq 10^6$，$0 \leq a_i < b_i \leq 10^6$。

题型分析

1：感觉
2：猜

策略1：按开始时间升序排序
反例：
活动1：[1，10]，活动2：[2，3]，活动3：[3，4]
➡ 选开始最早的 活动1 后，其他活动无法选择（仅1场），但最优解是选 活动2+3（2场）。

策略2：按结束时间升序排序
示例：
活动1：[1,4]，活动2：[3,5]，活动3：[4,7]，活动4：[5,9]
➡ 排序后：活动1(end=4)，活动2(end=5)，活动3(end=7)，活动4(end=9)
➡ 选 活动1 ➡ 跳过 活动2（与 活动1 重叠）➡ 选 活动3 ➡ 跳过 活动4（与 活动3 重叠），共2场。
最优解验证：选 活动2 + 活动4 也是2场，无更优解。

题型分析

贪心的本质是 为未来留出更多时间。结束越早的活动，对后续活动的‘阻碍’越小。

假设有两个活动：
A: [1, 4]（结束早）
B: [2, 5]（结束晚）
选 A 后，后续活动只需满足开始时间 ≥4；
选 B 后，后续活动需满足开始时间 ≥5。
→ A 为后续留出更多机会！

题型分析

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
struct node {
    int start;
    int end;
}act[1000005];
bool cmp(node x, node y) {
    return x.end < y.end; // 按结束时间升序排序
}
int main() {
    int n;
    cin >> n;
    node act[n];
    // 读入所有活动的时间区间
    for (int i = 0; i < n; i++) {
        cin >> act[i].start >> act[i].end;
    }
    // 按结束时间升序排序
    sort(act, act + n, cmp);
    int count = 0;      // 记录选择的活动数量
    int end = -1;       // 记录当前选择的活动的结束时间
    // 贪心选择不冲突的活动
    for (int i = 0; i < n; i++) {
        if (act[i].start >= end) { // 当前活动开始时间晚于上一个活动的结束时间
            count++;               // 选择该活动
            end = act[i].end;      // 更新结束时间
        }
    }
    cout << count; // 输出最多能参加的活动数量
    return 0;
}
```

题型分析

3. 区间划分

区间划分问题是指：给定一组连续的元素序列，将其划分为最少的连续子区间，使得每个子区间满足特定的约束条件（如区间和不超过阈值）。

题型分析

题目描述

复制 Markdown 展开 进入 IDE 模式

对于给定的一个长度为 N 的正整数数列 A_i，现要将其分成连续的若干段，并且每段和不超过 M（可以等于 M），问最少能将其分成多少段使得满足要求。

输入格式

第1行包含两个正整数 N, M，表示了数列 A_i 的长度与每段和的最大值，第 2 行包含 N 个空格隔开的非负整数 A_i，如题目所述。

输出格式

一个正整数，输出最少划分的段数。

输入输出样例

输入 #1 复制	输出 #1 复制
5 6 4 2 4 5 1	3

说明/提示

对于20%的数据，有$N \leq 10$；

对于40%的数据，有$N \leq 1000$；

对于100%的数据，有$N \leq 100000, M \leq 10^9$，$M$大于所有数的最大值，$A_i$之和不超过$10^9$。

将数列如下划分：

$[4][24][51]$

第一段和为4，第2段和为6，第3段和为6均满足和不超过$M = 6$，并可以证明3是最少划分的段数。

题型分析

局部最优蕴含全局最优：

每次尽可能填满当前分段，相当于为后续分段"节省"了容量空间

提前结束分段只会导致更多不必要的分段

无后效性：

决策只依赖于当前分段状态

当前决策不影响后续元素的处理方式

必要性原则：

当加入新元素会导致分段超限时，必须创建新分段

这是唯一可行的选择，没有其他可能性

题型分析

顺序遍历数列，尽可能将当前元素加入已有分段，只在加入后会导致分段和超过 M 时才创建新分段。

题型分析


```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, M;
    cin >> n >> M;  // 读取序列长度和每段最大和

    vector<int> nums(n);  // 存储序列的数组
    for (int i = 0; i < n; i++) {
        cin >> nums[i];  // 读取序列元素
    }

    int cnt = 1;    // 分段计数（至少1段）
    int cur = 0;    // 当前分段的和

    for (int i = 0; i < n; i++) {
        // 尝试加入当前分段
        if (cur + nums[i] <= M) {
            cur += nums[i];  // 加入当前分段
        } else {
            cnt++;    // 需要新分段
            cur = nums[i];  // 新分段从当前元素开始
        }
    }

    cout << cnt << endl;  // 输出最少分段数
    return 0;
}
```

03

总结展望

TEACHING
COURSEWARE

TEACH

排序

找到贪心策略

04

Q&A

TEACHING
COURSEWARE

TEACH