2025年暑假竞赛集训

习题课

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.02

目录

01

题目描述

TEACHING
COURSEWARE

TEACH

题目描述

TEACHING COURSEWARE

启航舱

题目背景

在步行街中，有 n 间商店买卖同一种商品，第i间商店一件商品的收购价和出售价均为 a_i 元。为了防 止过度交易，步行街有一个规定：您在第i间商店最多进行b_i 次交易（一次买或一次卖均计为一次交 易），且每次只能交易一件商品。

您准备通过在步行街中买卖这种商品来赚钱。假如初始时有无限的金钱（也就是说，不会因为钱不够而买不了一件商品），您最多能在步行街中赚到多少总利润？具体来说，“利润”指的是卖出商品获得的金钱总额，减去购买商品花费的金钱总额。

输入格式：

第一行输入一个整数n （$1 \leq n \leq 10^5$），表示商店的数量。对于接下来 n 行，第 i 行输入两个整数 a_i 和b_i （$1 \leq a_i, b_i \leq 10^6$），分别表示第$i$间商店的商品价格，以及该商店可以交易的最大次数。

输出格式

输出一行一个整数，表示在步行街中赚到的最大总利润。

题目描述

样例1

输入

```
4
10 2
30 7
20 4
50 1
```

输出

```
100
```

样例2

输入

```
2
1 100
1 1000
```

输出

```
0
```

样例解释

对于第一组样例数据，最优方案是在第 1 间商店买入2件商品，在第3间商店买入4件商品，在第 2 间商店卖出 5 件商品，在第 4 间商店卖出 1 件商品。总利润为 $30 * 5 + 50 * 1 - 10 * 2 - 20 * 4 = 100$。对于第二组样例数据，由于所有商店的商品价格都相同，因此无法获得利润。

数据分布

40% 的数据，$1 \leq n \leq 100$，$1 \leq a_i, b_i \leq 100$

100% 的数据，$1 \leq n \leq 10^5$，$1 \leq a_i, b_i \leq 10^6$

02

知识回归

TEACHING
COURSEWARE

TEACH

知识回归

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

题目解答

TEACHING
COURSEWARE

TEACH

题目解答

TEACHING COURSEWARE

注意到一定是购买按照排序后前一半，以后一半售出最佳，如果一共有偶数件，直接按照一半一半的购买即可，但是如果是奇数件，需要从前往后买 n/2 向下取整，从后往前卖 n/2 向下取整，会空出中间一个。

04

标准程序

TEACHING
COURSEWARE

TEACH

```cpp
#include<bits/stdc++.h>
#define int long long
const int N = 300010;
using namespace std;
struct node {
    int a;  // 商品价格
    int b;  // 该商店最多最大交易次数
} e[N];  // 存储所有商店信息的数组（使用1-based索引）
// 自定义比较函数：按商品价格升序排序
bool cmp(const node& x, const node& y) {
    return x.a < y.a;
}
```

```
signed main() {
    int n;  // 商店数量
    cin >> n;

    for (int i = 1; i <= n; i++) {
        cin >> e[i].a >> e[i].b;
    }
    sort(e + 1, e + n + 1, cmp);
    // 计算所有商店的总交易次数
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += e[i].b;
    }
    // 最大可完成的交易对数为总交易次数的一半（一次买+一次卖为一对）
    int cnt = sum / 2;
    int sum1 = 0;  // 买入商品的总成本
    int sum2 = 0;  // 卖出商品的总收入
```

```cpp
// 正序遍历（从价格最低的商店开始），计算买入总成本
for (int i = 1; i <= n; i++) {
    // 如果当前商店的交易次数足够满足剩余需求
    if (e[i].b <= cnt) {
        sum1 += e[i].a * e[i].b;  // 累加当前商店所有商品的买入成本
        cnt -= e[i].b;            // 减少剩余需要买入的数量
    }
    // 如果当前商店的交易次数多于剩余需求
    else if (cnt > 0) {
        sum1 += cnt * e[i].a;  // 只买入所需剩余数量
        cnt = 0;               // 买入需求已满足，退出循环
        break;
    }
}
```

```cpp
    // 重置计数器，准备计算卖出收入
    cnt = sum / 2;
    // 逆序遍历（从价格最高的商店开始）
    for (int i = n; i >= 1; i--) {
        // 如果当前商店的交易次数足够满足剩余需求
        if (e[i].b <= cnt) {
            sum2 += e[i].a * e[i].b;  // 累加当前商店所有商品的卖出收入
            cnt -= e[i].b;            // 减少剩余需要卖出的数量
        }
        // 如果当前商店的交易次数多于剩余需求
        else if (cnt > 0) {
            sum2 += cnt * e[i].a;  // 只卖出所需剩余数量
            cnt = 0;               // 卖出需求已满足，退出循环
            break;
        }
    }
    // 总利润 = 卖出总收入 - 买入总成本
    cout << sum2 - sum1 << '\n';
    return 0;
}
```