贪心-区间问题

一、AcWing 905. 区间选点

【题目描述】

给定N个闭区间$[a_i, b_i]$，请你在数轴上选择尽量少的点，使得每个区间内至少包含一个选出的点。

输出选择的点的最小数量。

位于区间端点上的点也算作区间内。

【输入格式】

第一行包含整数N，表示区间数。

接下来N行，每行包含两个整数a_i, b_i，表示一个区间的两个端点。

【输出格式】

输出一个整数，表示所需的点的最小数量。

【数据范围】

$1 \leq N \leq 10^5$

$-10^9 \leq a_i \leq b_i \leq 10^9$

【输入样例】

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

【输出样例】

```
2
```

【代码】

```
#include <iostream>

#include <algorithm>
```

```cpp
using namespace std;

const int N = 100010;
int n;

struct Edge
{
    int l, r;
    bool operator< (const Edge& t) const
    {
        return r < t.r;
    }
}e[N];

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> e[i].l >> e[i].r;
    sort(e, e + n);
    int cnt = 0, ed = -0x3f3f3f3f;
    for (int i = 0; i < n; i++)
        if (e[i].l > ed) { cnt++; ed = e[i].r; }
    cout << cnt << endl;
    return 0;
}
```

二、AcWing 908. 最大不相交区间数量

【题目描述】

给定N个闭区间$[a_i, b_i]$，请你在数轴上选择若干区间，使得选中的区间之间互不相交（包括端点）。

输出可选取区间的最大数量。

【输入格式】

第一行包含整数N，表示区间数。

接下来N行，每行包含两个整数a_i, b_i，表示一个区间的两个端点。

【输出格式】

输出一个整数，表示可选取区间的最大数量。

【数据范围】

$1 \leq N \leq 10^5$

$-10^9 \leq a_i \leq b_i \leq 10^9$

【输入样例】

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

【输出样例】

```
2
```

【代码】

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

const int N = 100010;
int n;

struct Edge
{
    int l, r;
    bool operator< (const Edge& t) const
    {
        return r < t.r;
    }
}e[N];

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> e[i].l >> e[i].r;
    sort(e, e + n);
    int cnt = 0, ed = -0x3f3f3f3f;
    for (int i = 0; i < n; i++)
        if (e[i].l > ed) { cnt++; ed = e[i].r; }
    cout << cnt << endl;
    return 0;
}
```

三、AcWing 906. 区间分组

【题目描述】

给定N个闭区间$[a_i, b_i]$，请你将这些区间分成若干组，使得每组内部的区间两两之间（包括端点）没有交集，并使得组数尽可能小。

输出最小组数。

【输入格式】

第一行包含整数N，表示区间数。

接下来N行，每行包含两个整数a_i, b_i，表示一个区间的两个端点。

【输出格式】

输出一个整数，表示最小组数。

【数据范围】

$1 \leq N \leq 10^5$

$-10^9 \leq a_i \leq b_i \leq 10^9$

【输入样例】

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

【输出样例】

```
2
```

【分析】

---

1. 先将所有区间按左端点从小到大排序
2. 从前往后处理每一个区间，判断能否将其放到某个现有的组中$(L[i] > Max_r)$

（1）如果不存在这样的组，则开新的组，然后将其放入

（2）如果存在这样的组，将其放进去，并更新当前组的Max_r

【代码】

```cpp
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;

typedef pair<int, int> PII;
const int N = 100010;
PII e[N];
int n;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> e[i].first >> e[i].second;
    sort(e, e + n);
    priority_queue<int, vector<int>, greater<int> > Q;
    for (int i = 0; i < n; i++)
        if (Q.empty() || e[i].first <= Q.top()) Q.push(e[i].second);
        else Q.pop(), Q.push(e[i].second);
    cout << Q.size() << endl;
    return 0;
}
```

四、AcWing 907. 区间覆盖

【题目描述】

给定N个闭区间$[a_i, b_i]$以及一个线段区间$[s, t]$，请你选择尽量少的区间，将指定线段区间完全覆盖。

输出最少区间数，如果无法完全覆盖则输出-1。

【输入格式】

第一行包含两个整数s和t，表示给定线段区间的两个端点。

第二行包含整数N，表示给定区间数。

接下来N行，每行包含两个整数a_i, b_i，表示一个区间的两个端点。

【输出格式】

输出一个整数，表示所需最少区间数。

如果无解，则输出-1。

【数据范围】

$1 \leq N \leq 10^5$

$-10^9 \leq a_i \leq b_i \leq 10^9$

$-10^9 \leq s \leq t \leq 10^9$

【输入样例】

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

【输出样例】

```
2
```

【分析】

---

1. 将所有区间按左端点从小到大排序；
2. 从前往后一次枚举每个区间，在所有能覆盖$start$的区间中，选择右端点最大的区间，然后将$start$更新成右端点的最大值。

【代码】

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

const int N = 100010;
int n, st, ed;

struct Edge
{
    int l, r;
    bool operator< (const Edge& t) const
    {

        return l < t.l;
```

```
    }
}e[N];

int main()
{
    cin >> st >> ed >> n;
    for (int i = 0; i < n; i++) cin >> e[i].l >> e[i].r;
    sort(e, e + n);
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        int r = -0x3f3f3f3f, j = i;
        while (j < n && e[j].l <= st) r = max(r, e[j++].r);
        if (r < st) break;//st左边的所有线段都无法覆盖到st
        res++;
        st = r;
        i = j - 1;
        if (st >= ed) break;
    }
    if (st < ed) cout << -1 << endl;
    else cout << res << endl;
    return 0;
}
```

贪心-Huffman树

一、AcWing 148. 合并果子

【题目描述】

在一个果园里，达达已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。

达达决定把所有的果子合成一堆。

每一次合并，达达可以把两堆果子合并到一起，消耗的体力等于**两堆果子的重量之和**。

可以看出，所有的果子经过$n-1$次合并之后，就只剩下一堆了。

达达在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以达达在合并果子时要尽可能地节省体力。

假定每个果子重量都为1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使达达耗费的体力最少，并输出这个最小的体力耗费值。

例如有`3`种果子，数目依次为`1,2,9`。

可以先将`1,2`堆合并，新堆数目为`3`，耗费体力为`3`。

接着，将新堆与原先的第三堆合并，又得到新的堆，数目为`12`，耗费体力为`12`。

所以达达总共耗费体力为`3+12=15`。

可以证明`15`为最小的体力耗费值。

【输入格式】

输入包括两行，第一行是一个整数n，表示果子的种类数。

第二行包含n个整数，用空格分隔，第i个整数a_i是第i种果子的数目。

【输出格式】

输出包括一行，这一行只包含一个整数，也就是最小的体力耗费值。

输入数据保证这个值小于2^{31}。

【数据范围】

$1 \leq n \leq 10000$

$1 \leq a_i \leq 20000$

【输入样例】

```
3
1 2 9
```

【输出样例】

```
15
```

【代码】

```
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;


int main()
```

```
{
    int n;
    cin >> n;
    priority_queue<int, vector<int>, greater<int> > Q;
    while (n--)
    {
        int x;
        cin >> x;
        Q.push(x);
    }
    int res = 0;
    while (Q.size() > 1)
    {
        int a = Q.top(); Q.pop();
        int b = Q.top(); Q.pop();
        res += a + b;
        Q.push(a + b);
    }
    cout << res << endl;
    return 0;
}
```

贪心-排序不等式

一、AcWing 913. 排队打水

【题目描述】

有n个人排队到1个水龙头处打水，第i个人装满水桶所需的时间是t_i，请问如何安排他们的打水顺序才能使所有人的等待时间之和最小？

【输入格式】

第一行包含整数n。

第二行包含n个整数，其中第i个整数表示第i个人装满水桶所花费的时间t_i。

【输出格式】

输出一个整数，表示最小的等待时间之和。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq t_i \leq 10^4$

【输入样例】

```
7
3 6 1 4 2 5 7
```

【输出样例】

```
56
```

【代码】

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

typedef long long LL;
const int N = 100010;
int a[N];
int n;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    sort(a, a + n);
    LL res = 0;
    for (int i = 0; i < n; i++) res += a[i] * (n - 1 - i);
    cout << res << endl;
    return 0;
}
```

贪心-绝对值不等式

一、AcWing 104. 货仓选址

【题目描述】

在一条数轴上有N家商店，它们的坐标分别为$A_1 \sim A_N$。

现在需要在数轴上建立一家货仓，每天清晨，从货仓到每家商店都要运送一车商品。

为了提高效率，求把货仓建在何处，可以使得货仓到每家商店的距离之和最小。

【输入格式】

第一行输入整数N。

第二行N个整数$A_1 \sim A_N$。

【输出格式】

输出一个整数，表示距离之和的最小值。

【数据范围】

$1 \leq N \leq 100000$

$0 \leq A_i \leq 40000$

【输入样例】

```
4
6 2 9 1
```

【输出样例】

```
12
```

【分析】

货仓的坐标为各商店坐标的中位数~

【代码】

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

typedef long long LL;
const int N = 100010;
int a[N];
int n;

int main()
{
    cin >> n;
```

```
    for (int i = 0; i < n; i++) cin >> a[i];
    nth_element(a, a + n / 2, a + n);
    LL res = 0;
    for (int i = 0; i < n; i++) res += abs(a[i] - a[n / 2]);
    cout << res << endl;
    return 0;
}
```

贪心-推公式

一、AcWing 125. 耍杂技的牛

【题目描述】

农民约翰的N头奶牛（编号为$1 \sim N$）计划逃跑并加入马戏团，为此它们决定练习表演杂技。

奶牛们不是非常有创意，只提出了一个杂技表演：

叠罗汉，表演时，奶牛们站在彼此的身上，形成一个高高的垂直堆叠。

奶牛们正在试图找到自己在这个堆叠中应该所处的位置顺序。

这N头奶牛中的每一头都有着自己的重量W_i以及自己的强壮程度S_i。

一头牛支撑不住的可能性取决于它头上所有牛的总重量（不包括它自己）减去它的身体强壮程度的值，现在称该数值为风险值，风险值越大，这只牛撑不住的可能性越高。

您的任务是确定奶牛的排序，使得所有奶牛的风险值中的最大值尽可能的小。

【输入格式】

第一行输入整数N，表示奶牛数量。

接下来N行，每行输入两个整数，表示牛的重量和强壮程度，第i行表示第i头牛的重量W_i以及它的强壮程度S_i。

【输出格式】

输出一个整数，表示最大风险值的最小可能值。

【数据范围】

$1 \leq N \leq 50000$

$1 \leq W_i \leq 10,000$

$1 \leq S_i \leq 1,000,000,000$

【输入样例】

```
3
10 3
2 5
3 3
```

【输出样例】

```
2
```

【分析】

首先给出结论：按$w+s$的值从小到大排序，值最大的在最下面，最小的在最上面。

证明：

假设第$i+1$头牛在第i头牛的下面，记风险值为v，则在将两头牛交换前有：

- $v_i = \sum_{j=1}^{i-1} w_j - s_i$
- $v_{i+1} = \sum_{j=1}^{i} w_j - s_{i+1}$

将两头牛交换位置后有：

- $v_i = \sum_{j=1}^{i-1} w_j + w_{i+1} - s_i$
- $v_{i+1} = \sum_{j=1}^{i-1} w_j - s_{i+1}$

其他牛的危险值显然不变，所以分析交换前后这两头牛中最大的危险值即可。

将上述式子进行化简，每个式子减去$\sum_{j=1}^{i-1} w_j$得到如下式子：

- 交换前：$v_i = -s_i$，交换后：$v_i = w_{i+1} - s_i$
- 交换前：$v_{i+1} = w_i - s_{i+1}$，交换后：$v_{i+1} = -s_{i+1}$

由于s, w都是正数，$w_i - s_{i+1} > -s_{i+1}, w_{i+1} - s_i > -s_i$。

因此比较$w_i - s_{i+1}$和$w_{i+1} - s_i$即可。

- 当$w_i - s_{i+1} \geq w_{i+1} - s_i$，即$w_i + s_i \geq w_{i+1} + s_{i+1}$时，交换后更优；
- 当$w_i - s_{i+1} < w_{i+1} - s_i$，即$w_i + s_i < w_{i+1} + s_{i+1}$时，交换前更优。

所以得到做法：按每头牛的$w+s$进行排序，当存在逆序时就进行交换（即升序排序），然后根据题意算出每头牛的危险值并记录其中的最大值即可。

【代码】

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

typedef pair<int, int> PII;
const int N = 50010;
PII cow[N];
int n;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> cow[i].first >> cow[i].second;
        cow[i].first += cow[i].second;
    }
    sort(cow, cow + n);//按w+s的大小从小到大排序
    int res = -0x3f3f3f3f, sum = 0;
    for (int i = 0; i < n; i++)
    {
        int s = cow[i].second, w = cow[i].first - s;
        res = max(res, sum - s);
        sum += w;
    }
    cout << res << endl;
    return 0;
}
```