区间问题汇总

区间合并

Description

给定 n 个区间 $[l_i, r_i][l_i, r_i]$，要求合并所有有交集的区间。

注意如果在端点处相交，也算有交集。

输出合并完成后的区间个数。

例如：$[1, 3][1, 3]$和$[2, 6][2, 6]$可以合并为一个区间$[1, 6][1, 6]$。

Input

第一行包含整数 n。

接下来 n 行，每行包含两个整数 l 和 r。

```
5
1 2
2 4
5 6
7 8
7 9
```

Output

共一行，包含一个整数，表示合并区间完成后的区间个数。

```
3
```

思路

1. 使用`pair<int, int>`类型的`vector`存储所有集合的左右端点
2. 将所有集合按**左端点**排序
3. 创建`vector`用于存储合并后的集合
4. 若当前集合的左端点大于当前合并集合的右端点，则二者不能合并。
    - 若不是第一个区间：将上一个合并后的区间放入vector，并更新l, r为当前值
    - 若是第一个区间：只更新l, r为当前值
5. 若当前集合的左端点小于等于当前合并集合的右端点,则二者可以合并。
    - 取当前集合和当前合并集合的最大值，作为当前合并集合的最大值。

代码

```cpp
#include<bits/stdc++.h>
using namespace std;

typedef pair<int,int> pii;
vector<pii> v;
int n;

void merge(vector<pii> &v){
    vector<pii> opt;
    int st = -1e9, ed = -1e9;
    for(auto item : v){
        if(ed < item.first){
            if(st != -1e9) opt.push_back({st, ed});
            st = item.first, ed = item.second;
        }
        else{
            ed = max(ed, item.second);
        }
    }
    if(st != -1e9) opt.push_back({st, ed});
    v = opt;
}

int main(){
    cin >> n;
    for(int i = 0; i < n; i++){
        int l, r;
        cin >> l >> r;
        v.push_back({l, r});
    }
    sort(v.begin(), v.end());

    merge(v);

    cout << v.size();

    return 0;
}
```

区间选点

Description

给定 N 个闭区间 $[a_i, b_i][a_i, b_i]$，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。

Input

第一行包含整数 n。

接下来 n 行，每行包含两个整数 l 和 r。

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

Output

```
2
```

思路

其实就是找到相互之间没有任何交集的区间有几个？

1. 所有区间按**右端点**从小到大排序
2. 遍历所有区间：
    - 初始区间为第一个区间
    - 若该区间左端点小于选定区间右端点，`ans++`，更新该区间为选定区间

代码

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int maxn = 0x3f3f3f3f;

int n, ans;
vector<pii> v;

int main(){
    cin >> n;
    for(int i = 0; i < n; i++){
        int l, r; cin >> l >> r;
        v.push_back({r, l});
    }

    sort(v.begin(), v.end());

    int st = -maxn, ed = -maxn;
    for(auto x : v){
        if(x.second > ed){
            ans ++;
            ed = x.first;
        }
    }

    cout << ans;

    return 0;
}
```

区间分组

Description

给定 N 个闭区间 $[a_i, b_i][a_i, b_i]$，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。

Input

第一行包含整数 n。

接下来 n 行，每行包含两个整数 l 和 r。

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

Output

```
2
```

思路1

1. 将所有区间按**左端点**从小到大排序
2. 从前往后枚举每个区间，判断能否将其放到某个现有的组中，即是否存在当前区间的的左端点 L > 任意组中右端点的最小值的组
    - 如果不存在这样的组，则开新组，然后再将其放进组中
    - 如果存在这样的组，则将其放在符合条件的组中，并更新当前组的右端点的值
3. 为了不用每次选择组时都遍历所有组，可以通过小根堆来维护所有组中的尾端

为什么不按右端点排序？（@小豆冰果Acwing）

比如，有n个人需要用教室，每个人占用教室的起始时间和终止时间是不一样的。

1. 如果想知道只有一间教室，能安排下的最多不冲突人数（不是所有的人都有机会，有的会被舍掉）是多少（区间选点和最大不相交问题），那么当然是最先结束的人排在前面，这样后面的人才有更多机会。如果是按左端点排序，那么如过一个人0点开始用，那么肯定他排在最前面，但是如果他自己就占用了24小时，那么只能给他一个人用了，这样就达不到最大的效果。所以按右端点排序。
2. 如果想知道这些人都必须安排，没有人被舍弃，至少需要多少个教室能安排下（区间分组问题）。那么肯定是按照开始时间排序，开始时间越早越优先。这样每间教室都能得到最充分的利用。

代码1

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int maxn = 0x3f3f3f3f;

```

```cpp
int n, ans;
vector<pii> v, opt;

int main(){
    cin >> n;
    for(int i = 0; i < n; i++){
        int l, r; cin >> l >> r;
        v.push_back({l, r});
    }

    sort(v.begin(), v.end());

    priority_queue<int, vector<int>, greater<int>> heap;
    heap.push(maxn);

    for(auto x : v){
        if(x.first > heap.top()){
            heap.pop();
            heap.push(x.second);
        }
        else{
            ans ++;
            heap.push(x.second);
        }
    }

    cout << ans;

    return 0;
}
```

思路2

我们可以把所有开始时间和结束时间排序，遇到开始时间就把需要的教室加1，遇到结束时间就把需要的教室减1,在一系列需要的教室个数变化的过程中，峰值就是多同时进行的活动数，也是我们至少需要的教室数。

代码2

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100100;

int n;
int b[2 * N], idx;

int main()
{
    cin >> n;
    for(int i = 0; i < n ; i ++)
    {
        int l, r;
        scanf("%d %d", &l, &r);
        b[idx ++] = l * 2;//标记左端点为偶数。
```

```
            b[idx ++] = r * 2 + 1;// 标记右端点为奇数。
        }

        sort(b, b + idx);

        int res = 1, t = 0;
        for(int i = 0; i < idx ; i ++)
        {
            if(b[i] % 2 == 0) t ++;
            else t --;
            res = max(res, t);
        }
        cout << res << endl;
        return 0;
}

作者：未来i
链接：https://www.acwing.com/solution/content/8902/
来源：AcWing
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

区间覆盖

Description

给定 N 个闭区间 $[a_i,b_i][a_i,b_i]$ 以及一个线段区间 $[s,t][s,t]$，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。

Input

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。

Output

输出一个整数，表示所需最少区间数。

如果无解，则输出 -1。

思路

1. 将所有区间按左端点从小到大进行排序
2. 从前往后枚举每个区间，在所有能覆盖s的区间中，选择右端点最大的区间，然后将s更新成右端点的最大值

代码

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 0x3f3f3f3f;
typedef pair<int, int> pii;

int n, s, t, ans;
```

```cpp
bool success;
vector<pii> v;

int main(){
    cin >> s >> t;
    cin >> n;
    for(int i = 0; i < n; i++){
        int l, r; cin >> l >> r;
        v.push_back({l, r});
    }

    sort(v.begin(), v.end());

    for(int i = 0; i < v.size(); i++){
        int j = i,tmp = s;
        while(v[j].first <= s && j < n){
            tmp = max(tmp, v[j].second);
            j++;
        }
        if(s == tmp || tmp < s){
            ans = -1;
            break;
        }
        ans++;
        if(tmp >= t){
            success = true;
            break;
        }
        s = tmp;
        i = j - 1;
    }

    if(success)cout<<ans;
    else cout <<-1;

    return 0;
}
```