区间合并

给定区间

合并后区间

区间合并算法：快速地把有交集的区间进行合并

特殊规定：若两区间只有端点重合，也视作可合并

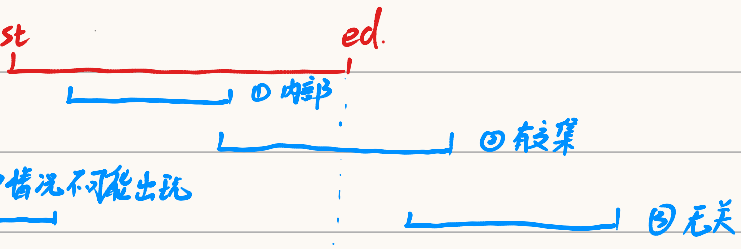
思路：

① 按区间左端点排序

② 把所有可能有交集的区间合并

维护一个区间 st ed

第i个区间与维护区间关系：



更新：① st ed 不变

② st不变 ed更新

③ st、ed都更新，且记录的区间数+1 当前区间是一个全新的区间

最开始我们维护的边界可看为 $(-\infty, -\infty)$

如数的范围为 $(-10^9, 10^9)$，那我可取 $(-2*10^9, 2*10^9)$

803. 区间合并

题目 | 提交记录 | 讨论 | 题解 | 视频讲解

给定 n 个区间 $[l_i, r_i]$，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：$[1, 3]$ 和 $[2, 6]$ 可以合并为一个区间 $[1, 6]$。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含两个整数 l 和 r。

输出格式

共一行，包含一个整数，表示合并区间完成后的区间个数。

数据范围

$1 \le n \le 100000$,
$-10^9 \le l_i \le r_i \le 10^9$

输入样例：

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

输出样例：

```
3
```

难度：简单

时/空限制：1s / 64MB

总通过数：112077

总尝试数：182990

来源：模板题

算法标签

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

typedef pair<int, int> PII;
const int N = 1e6 + 10;
int n;
vector<PII> segs;

void merge(vector<PII> & segs) {
    vector<PII> res; // 合并后结果
    sort(segs.begin(), segs.end(), [](PII &a, PII &b){return a.first < b.first;});
    int st = -2e9, ed = -2e9;

    for(int i = 0 ; i < segs.size(); i++) {
        //  这里不用单独判断st和segs[i],排序后一定满足 <= 关系的
        // 1 内部
        if( ed >= segs[i].second) continue;
        // 2 有交集但不在内部
        else if( ed < segs[i].second && ed >= segs[i].first) ed = segs[i].second;
        // 3 完全在外部
        else {
            if(st != -2e9) res.push_back({st, ed}); // 不能是初始的区间
            st = segs[i].first; ed = segs[i].second;
        }
    }

    if(st != -2e9) res.push_back({st, ed});
    // 最后的区间也要加入，但是为了防止segs为空，所以还要再判断一下
    segs = res;
    return ;
}

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

    merge(segs);
    cout << segs.size() << endl;
    return 0;
}
```

```cpp
#include <stdio.h>

const int N = 1e6 + 10;
typedef struct PII {
    int l, r;
} PII;

PII SEGS[N]; int segs = 0; // 存储区间
PII RES[N]; int res = 0;
int n;

PII tmp_p; //临时变量，方便使用

void qsort(PII *q, int l, int r) {
    if(l >= r) return ;
    int i = l - 1, j = r + 1;
    int x = q[l + r >> 1].l;
    while(i < j) {
        do i ++; while(q[i].l < x);
        do j --; while(q[j].l > x);
        if(i < j) {
            PII tmp = q[i]; q[i] = q[j]; q[j] = tmp;
        }
    }
    qsort(q, l, j); qsort(q, j + 1, r);
    return ;
}

void merge(PII *SEGS, int segs) {
    qsort(SEGS, 0, segs - 1); // 排序
    int st = -2e9, ed = -2e9; //  初始边界
    for(int i = 0; i < segs; i++) {
        int l = SEGS[i].l, r = SEGS[i].r;
        if(ed >= r) continue; // 情况1 i区间在当前区间内不
        else if(ed < r && ed >= l) ed = r; // 情况2 i区间和当前区间有重叠
        else { // 情况3 i区间和当前区间无重叠，当前区间可以收进RES中了
            if(st != -2e9) {// 最开始的区间是我们自己设置的，不要加进去
                tmp_p.l = st; tmp_p.r = ed;
                RES[res ++] = tmp_p;
            }
            st = l; ed = r; // 更新维护的区间
        }
    }
    if(st != -2e9) { // 最后一个区间，因为无后续区间，要手动判断
        tmp_p.l = st; tmp_p.r = ed; // 并且要防止SEGS为空而误加入我们虚拟出来的起始区间
        RES[res ++] = tmp_p;
    }
    return ;
}
```

```cpp
int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i ++) {
        int l, r;
        scanf("%d %d", &l, &r);
        tmp_p.l = l; tmp_p.r = r;
        SEGS[segs ++] = tmp_p;
    }
    merge(SEGS, segs);
    printf("%d", res);


    return 0;
}
```