Lesson3

> **双指针算法**
>
> - 两类
>   - 两个指针指向两个序列
>   - 两个指针指向一个序列
> - 一般写法
>
> ```
> for (i=0, j=0; i<n; i++) {
>     while (j < i && check(i, j)) j++;
>     //每道题目具体逻辑
> }
> ```
>
> - 核心思想
>
>   对朴素算法进行优化（单调性），时间复杂度优化为O(n)，常数为2，最坏情况下O(2n)
>
>   先想暴力算法，再通过单调性进行优化，O(n^2)->O(n)
> - e1: 输出字符串中每个单词
>
> ```
> #include <iostream>
> #include <cstring>
>
> using namespace std;
>
> int main(void) {
>     char str[1000];
>     cin.getline(str, 1010);
>
>     for (int i=0; str[i]; i++) {
>         if (str[i] == ' ') continue;
>
>         int j = i;
>         while (j<strlen(str) && str[j]!=' ') j++;
>
>         for (int k=i; k<j; k++) cout << str[k];
>         puts("");
>         i = j;
>     }
>
>     return 0;
> }
> ```
>
> - e2: 最长不重复子序列
>
> ```
> #include <iostream>
>
> using namespace std;
>
> const int N = 1e5+10;
>
> int a[N], s[N], n;
> int res;
>
> int main(void) {
>     scanf("%d", &n);
>
>     for (int i=0; i<n; i++) scanf("%d", &a[i]);
>
>     for (int i=0, j=0; i<n; i++) {
>         s[a[i]]++;
>
>         while (j<=i && s[a[i]]>1) {
>             s[a[j]]--;
>             j++;
>         }
>
>         res = max(res, i-j+1);
>     }
>
>     cout << res;
>
>     return 0;
> }
> ```

位运算

- **n的二进制表示中第k位是什么: n>>k&1**
- 个位（最后一位）是第0位，从个位开始
- 先把第k位移至最后一位（个位）（右移运算 n>>k）
- 求个位的值
- 结合1，2步，得公式**n>>k&1**

```
#include <iostream>

using namespace std;

int main(void) {
    int a = 10;

    for (int i=31; i>=0; i--) cout << (a>>i&1);

    return 0;
}
```

- **lowbit（x）返回x的最后一位（最右边）1的位置，主要用于树状数组**
  - x=1010，lowbit（x）=10
  - x=101000，lowbit（x）=1000
  - lowbit（x） = x&-x = x&(~x+1)
  - 应用：统计x中1的个数

```
#include <iostream>

using namespace std;

int lowbit(int x) {
    return x & -x;
}

int main(void) {
    int n;
    scanf("%d", &n);

    while (n--) {
        int x;
        scanf("%d", &x);

        int res = 0;
        while (x) x -= lowbit(x), res++;

        printf("%d ", res);
    }

    return 0;
}
```

离散化（整数离散化）

- 适用于值域大，个数少的序列，如值域0~10^9，个数10^5
- 重复元素的处理：**去重，库函数all.erase(unique(all.begin(), all.end()), all.end())**
- 如何算出a[i]中i离散化后的值是多少（**二分**）
- 对数组**下标**进行映射

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef pair<int, int> PII;

const int N = 300010;

int n, m;
int a[N], s[N];

```

```cpp
vector<int> all;
vector<PII> add, query;

//去重函数
vector<int>::iterator unique(vector<int> &a) {
    int j = 0;
    for (int i=0; i<a.size(); i++)
        if (!i || a[i]!=a[i-1])
            a[j++] = a[i];

    return a.begin()+j;
}

//二分
int find(int x) {
    int l = 0, r = all.size()-1;
    while (l < r) {
        int mid = l+r>>1;
        if (all[mid] >= x) r = mid;
        else l = mid+1;
    }

    return r+1;
}

int main(void) {
    scanf("%d%d", &n, &m);

    while (n--) {
        int x, c;
        scanf("%d%d", &x, &c);
        add.push_back({x, c});
        all.push_back(x);
    }

    while (m--) {
        int l, r;
        scanf("%d%d", &l, &r);
        query.push_back({l, r});
        all.push_back(l), all.push_back(r);
    }

    sort(all.begin(), all.end());
    // all.erase(unique(all.begin(), all.end()), all.end());
    all.erase(unique(all), all.end());

    //处理插入
    for (auto item: add) {
        int x = find(item.first);
        a[x] += item.second;
    }

    //预处理前缀和
    for (int i=1; i<=all.size(); i++) s[i] = s[i-1]+a[i];

    //处理查询
    for (auto item: query) {
        int l = find(item.first), r = find(item.second);
        printf("%d\n", s[r]-s[l-1]);
    }

    return 0;
}
```

区间（大多数贪心）合并

- 按区间左端点排序
- 扫描所有区间，把所有可能有交集的区间进行合并
    - 维护两个端点st（start），ed(end)
    - 3种情况
        - 包含

          st，ed不变
        - 有交

          更新ed
        - 不包含

更新st，ed（新区间）

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

using namespace std;

typedef pair<int, int> PII;

const int N = 100010;

int n;
vector<PII> segs;

void merge(vector<PII> &segs) {
    vector<PII> res;

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

    int st = -2e9, ed = -2e9;
    for (auto seg: segs)
        if (ed < seg.first) {
            if (ed != -2e9) res.push_back({st, ed});
            st = seg.first, ed = seg.second;
        }
        else ed = max(ed, seg.second);

    if (st != -2e9) res.push_back({st, ed});

    segs = res;
}

int main(void) {
    scanf("%d", &n);

    for (int i=0; i<n; i++) {
        int l, r;
        scanf("%d%d", &l, &r);
        segs.push_back({l, r});
    }

    merge(segs);

    cout << segs.size();

    return 0;
}
```