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
>
> ```
> static int N = 100010;
>
> static int n;
> static int[] a = new int[N], s = new int[N];
>
> public static void main(String[] args) throws Exception {
>     ins.nextToken(); n = (int)ins.nval;
>
>     for (int i=0; i<n; i++) { ins.nextToken(); a[i] = (int)ins.nval; }
>
>     int res = 0;
>     for (int i=0, j=0; i<n; i++) {
>         s[a[i]]++;
>
>         while (j < i && s[a[i]] > 1) {
>             s[a[j]]--;
>             j++;
>         }
>
>         res = Math.max(res, i-j+1);
>     }
>
>     out.print(res);
>
>     out.flush();
> }
> ```

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**

```cpp
#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

```java
static int n;

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

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    while (n-- > 0) {
        ins.nextToken(); int x = (int)ins.nval;

        int res = 0;
        while (x > 0) { x -= lowbit(x); res++; }
        out.print(res + " ");
    }

    out.flush();
}
```

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

```java
static int N = 300010;

static int n, m;
static int[] a = new int[N], s = new int[N];
static List<PII> ad = new ArrayList<PII>(), query = new ArrayList<PII>();
static List<Integer> all = new ArrayList<Integer>();

static int unique(List<Integer> a) {
    int j = 0;
    for (int i=0; i<a.size(); i++) {
        if (i==0 || a.get(i) != a.get(i-1))
            a.set(j++, a.get(i));
    }
    return j;
}

static int find(int x) {
    int l = 0, r = all.size()-1;
    while (l < r) {
        int mid = l+r>>1;
```

```
        if (all.get(mid) >= x) r = mid;
        else l = mid+1;
    }
    return l+1;     //前缀和从1开始
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=0; i<n; i++) {
        ins.nextToken(); int x = (int)ins.nval;
        ins.nextToken(); int c = (int)ins.nval;
        ad.add(new PII(x, c));
        all.add(x);
    }

    for (int i=0; i<m; i++) {
        ins.nextToken(); int l = (int)ins.nval;
        ins.nextToken(); int r = (int)ins.nval;
        query.add(new PII(l, r));
        all.add(l); all.add(r);
    }

    Collections.sort(all);
    all = all.subList(0, unique(all));

    for (PII p: ad) {
        int x = find(p.first);
        a[x] += p.second;
    }

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

    for (PII p: query) {
        int l = find(p.first), r = find(p.second);
        out.println(s[r]-s[l-1]);
    }

    out.flush();
}

static class PII {
    int first, second;

    PII(int f, int s) {
        first = f; second = s;
    }
}
```

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）

```
static int N = 100010;

static int n;
static List<PII> segs = new ArrayList<PII>();

static int merge(List<PII> segs) {
    List<PII> res = new ArrayList<PII>();

    segs.sort((o1, o2) -> o1.first-o2.first);   //sort参数>0交换，<0不交换

    int st = (int)-2e9, ed = (int)-2e9;
```

```
    for (PII seg: segs) {
        if (ed < seg.first) {
            if (ed != -2e9) res.add(new PII(st, ed));
            st = seg.first; ed = seg.second;
        }
        ed = Math.max(ed, seg.second);
    }

    if (st != -2e9) res.add(new PII(st, ed));

    return res.size();
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;

    while (n-- > 0) {
        ins.nextToken(); int l = (int)ins.nval;
        ins.nextToken(); int r = (int)ins.nval;
        segs.add(new PII(l, r));
    }

    out.print(merge(segs));

    out.flush();
}

static class PII {
    int first, second;

    PII(int f, int s) {
        first = f; second = s;
    }
}
```