

## 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

```
1  for (i=0, j=0; i<n; i++) {
2      while (j < i && check(i, j)) j++;
3      //每道题目具体逻辑
4  }
```

- 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， $O(n^2) \rightarrow O(n)$

- e1: 输出字符串中每个单词

```
1  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  int main(void) {
7      char str[1000];
8      cin.getline(str, 1010);
9
10     for (int i=0; str[i]; i++) {
11         if (str[i] == ' ') continue;
12
13         int j = i;
14         while (j<strlen(str) && str[j]!=' ') j++;
15
16         for (int k=i; k<j; k++) cout << str[k];
17         puts("");
18         i = j;
19     }
20
21     return 0;
22 }
```

- e2: 最长不重复子序列

```
1  static int N = 100010;
2
3  static int n;
4  static int[] a = new int[N], s = new int[N];
5
6  public static void main(String[] args) throws Exception {
7      ins.nextToken(); n = (int)ins.nval;
8
9      for (int i=0; i<n; i++) { ins.nextToken(); a[i] = (int)ins.nval; }
10
11     int res = 0;
12     for (int i=0, j=0; i<n; i++) {
13         s[a[i]]++;
14
15         while (j < i && s[a[i]] > 1) {
16             s[a[j]]--;
17             j++;
18         }
19
20         res = Math.max(res, i-j+1);
21     }
22
23     out.print(res);
24
25     out.flush();
26 }
```

## 位运算

- **n的二进制表示中第k位是什么:  $n \gg k \& 1$**
- 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  $n \gg k$ ）
- 求个位的值
- 结合1, 2步，得公式  **$n \gg k \& 1$**

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(void) {
6      int a = 10;
7
8      for (int i=31; i>=0; i--) cout << (a>>i&1);
9
10     return 0;
11 }
```

- **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**
  - $x=1010$ ,  $\text{lowbit}(x) = 10$
  - $x=101000$ ,  $\text{lowbit}(x) = 1000$
  - $\text{lowbit}(x) = x \& -x = x \& (\sim x + 1)$
  - 应用：统计x中1的个数

```
1  static int n;
2
3  static int lowbit(int x) {
4      return x&-x;
5  }
6
7  public static void main(String[] args) throws Exception {
8      ins.nextToken(); n = (int)ins.nval;
9
10     while (n-- > 0) {
11         ins.nextToken(); int x = (int)ins.nval;
12
13         int res = 0;
14         while (x > 0) { x -= lowbit(x); res++; }
15         out.print(res + " ");
16     }
17
18     out.flush();
19 }
```

## 离散化（整数离散化）

- 适用于值域大，个数少的序列，如值域  $0 \sim 10^9$ ，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]中离散化后的值是多少（二分）
- 对数组**下标**进行映射

```
1  static int N = 300010;
2
3  static int n, m;
4  static int[] a = new int[N], s = new int[N];
5  static List<PII> ad = new ArrayList<PII>(), query = new ArrayList<PII>();
6  static List<Integer> all = new ArrayList<Integer>();
7
8  static int unique(List<Integer> a) {
9      int j = 0;
10     for (int i=0; i<a.size(); i++) {
11         if (i==0 || a.get(i) != a.get(i-1))
12             a.set(j++, a.get(i));
13     }
14     return j;
15 }
16
17 static int find(int x) {
18     int l = 0, r = all.size()-1;
19     while (l < r) {
20         int mid = l+r>>1;
```

```

21         if (all.get(mid) >= x) r = mid;
22         else l = mid+1;
23     }
24     return l+1;    //前缀和从1开始
25 }
26
27 public static void main(String[] args) throws Exception {
28     ins.nextToken(); n = (int)ins.nval;
29     ins.nextToken(); m = (int)ins.nval;
30
31     for (int i=0; i<n; i++) {
32         ins.nextToken(); int x = (int)ins.nval;
33         ins.nextToken(); int c = (int)ins.nval;
34         ad.add(new PII(x, c));
35         all.add(x);
36     }
37
38     for (int i=0; i<m; i++) {
39         ins.nextToken(); int l = (int)ins.nval;
40         ins.nextToken(); int r = (int)ins.nval;
41         query.add(new PII(l, r));
42         all.add(l); all.add(r);
43     }
44
45     Collections.sort(all);
46     all = all.subList(0, unique(all));
47
48     for (PII p: ad) {
49         int x = find(p.first);
50         a[x] += p.second;
51     }
52
53     for (int i=1; i<=all.size(); i++) s[i] = s[i-1]+a[i];    //预处理前缀和
54
55     for (PII p: query) {
56         int l = find(p.first), r = find(p.second);
57         out.println(s[r]-s[l-1]);
58     }
59
60     out.flush();
61 }
62
63 static class PII {
64     int first, second;
65
66     PII(int f, int s) {
67         first = f; second = s;
68     }
69 }

```

## 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)

```

1  static int N = 100010;
2
3  static int n;
4  static List<PII> segs = new ArrayList<PII>();
5
6  static int merge(List<PII> segs) {
7      List<PII> res = new ArrayList<PII>();
8
9      segs.sort((o1, o2) -> o1.first-o2.first);    //sort参数>0交换, <0不交换
10
11     int st = (int)-2e9, ed = (int)-2e9;

```

```
12     for (PII seg: segs) {
13         if (ed < seg.first) {
14             if (ed != -2e9) res.add(new PII(st, ed));
15             st = seg.first; ed = seg.second;
16         }
17         ed = Math.max(ed, seg.second);
18     }
19
20     if (st != -2e9) res.add(new PII(st, ed));
21
22     return res.size();
23 }
24
25 public static void main(String[] args) throws Exception {
26     ins.nextToken(); n = (int)ins.nval;
27
28     while (n-- > 0) {
29         ins.nextToken(); int l = (int)ins.nval;
30         ins.nextToken(); int r = (int)ins.nval;
31         segs.add(new PII(l, r));
32     }
33
34     out.print(merge(segs));
35
36     out.flush();
37 }
38
39 static class PII {
40     int first, second;
41
42     PII(int f, int s) {
43         first = f; second = s;
44     }
45 }
```