# 线段树解决离线询问

主讲人：邓哲也

# HDU 3333 - Turing Tree

有一个长度为 n 的序列 A[1], A[2], …, A[n]

每次给出一个询问 (x, y):

请你求出 A[x .. y] 中出现过的数字之和。（出现多次的数只计算一次）

n <= 30000, Q <= 100000

**Sample Input**

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

**Sample Output**

```
1
5
6
3
6
```

# HDU 3333 - Turing Tree

思考直接用线段树来维护。

无法合并左右两个子节点。

因为不能统计每个节点出现了哪几种数。

# HDU 3333 - Turing Tree

换一个思路，思考暴力做法。

枚举 x 到 y 中间的每个数，如果是重复的，那么就不加进答案，否则加入。

如何判断是否重复？

我们可以用一个 flag 数组，表示这个数已经出现过了。

# HDU 3333 - Turing Tree

```
for (int i = x;i <= y;i ++){
    if (!flag[a[i]]) ans += a[i];
    flag[a[i]] = 1;
}
```

# HDU 3333 - Turing Tree

还有一种思路，记录一个数组 left[i]，表示左边第一个出现的相同数字 a[i] 的下标。

这样如果 left[i] < l，就说明 a[i] 是 [x, y] 中第一个出现的 a[i].

如果 left[i] >= l，就说明在 [x, i - 1] 中已经出现过一次 a[i]了，不用累计进答案。

# HDU 3333 - Turing Tree

```
for (int i = x;i <= y;i ++)
    if (left[i] < x) ans += a[i];
```

预处理 left 数组：

STL中的map 或

离散化

# HDU 3333 - Turing Tree

现在问题变成了给定 left[1..n], a[1..n]

每次求 [x, y] 中 left < x 的 a 的和。

这个时候可以考虑离线求解。

我们对 left 值排序，然后顺序枚举 x，依次把 a 插入线段树。

就可以做到消除一维限制。

# HDU 3333 - Turing Tree

从小到大枚举 x，把当前 left[i] < x 的 a[i] 插入线段树中的第 i 个位置。

表明：对于任意的 y，询问 [x, y] 的话，如果 i 在 [x, y]中，那一定有 left[i] < x，所以 a[i] 是要被统计进答案的。

这样就做到了 O((n + q) log n) 的时间复杂度，离线解决了所有询问。

# 代码实现

```
#define N 1000
#define ls (x << 1)
#define rs (x << 1 | 1)
double xbin[N], ybin[N];
int xcnt, ycnt, n;
double sum[N];
int val[N], tag[N];
struct rect{
    double x1, y1, x2, y2;
}r[N];
struct event{
    int x1, x2, v;
};
vector <event> g[N];
```

# 代码实现

```
#define N 100010

struct node{
    int l, r, id;
}q[N], p[N];
int a[N], bin[N], n, m, last[N], le[N], cnt;
typedef long long ll;
ll sum[N << 2], ans[N];
#define ls (x << 1)
#define rs (x << 1 | 1)
bool cmp(const node &a, const node &b){
    return a.l < b.l;
}
```

# 代码实现

```
void upd(int x){
    sum[x] = sum[ls] + sum[rs];
}
void add(int pos, int v, int l, int r, int x){
    if (l == r){
        sum[x] += v;
        return;
    }
    int mid = (l + r) >> 1;
    if (pos <= mid) add(pos, v, l, mid, ls);
    else add(pos, v, mid + 1, r, rs);
    upd(x);
}
```

# 代码实现

```
ll query(int A, int B, int l, int r, int x){
    if (A <= l && r <= B) return sum[x];
    int mid = (l + r) >> 1;
    ll ret = 0;
    if (A <= mid) ret += query(A, B, l, mid, ls);
    if (mid < B)  ret += query(A, B, mid + 1, r, rs);
    return ret;
}
```

# 代码实现

```
int main(){
    int tc;
    scanf("%d", &tc);
    while(tc --){
        scanf("%d", &n);
        for(int i = 1;i <= n;i ++) scanf("%d", &a[i]), bin[++ cnt] = a[i];
        scanf("%d", &m);
        for(int i = 1;i <= m;i ++) scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;
        sort(bin + 1, bin + cnt + 1);
        cnt = unique(bin + 1, bin + cnt + 1) - bin - 1;
        for(int i = 1;i <= n;i ++) a[i] = lower_bound(bin + 1, bin + cnt + 1,
a[i]) - bin;
```

# 代码实现

```
for(int i = 1;i <= cnt;i ++) last[i] = 0;
for(int i = 1;i <= n;i ++){
    le[i] = last[a[i]];
    p[i] = (node){le[i], i, bin[a[i]]};
    last[a[i]] = i;
}
sort(p + 1, p + n + 1, cmp);
sort(q + 1, q + m + 1, cmp);
for(int i = 1;i <= n * 4;i ++) sum[i] = 0;
```

# 代码实现

```
        int j = 1;
        for(int i = 1;i <= m;i ++){
            while(j <= n && p[j].l < q[i].l){
                add(p[j].r, p[j].id, 1, n, 1);
                j ++;
            }
            ans[q[i].id] = query(q[i].l, q[i].r, 1, n, 1);
        }
        for(int i = 1;i <= m;i ++) printf("%lld\n", ans[i]);
    }
    return 0;
}
```

# 下节课再见