306-03-BIT-逆序对应用

P1966 火柴排队

题目描述

涵涵有两盒火柴，每盒装有n根火柴，每根火柴都有一个高度。 现在将每盒中的火柴各自排成一列， 同一列火柴的高度互不相同， 两列火柴之间的距离定义为：$\sum(a_i-b_i)^2$

其中a_i表示第一列火柴中第i个火柴的高度，b_i表示第二列火柴中第i个火柴的高度。

每列火柴中相邻两根火柴的位置都可以交换，请你通过交换使得两列火柴之间的距离最小。请问得到这个最小的距离，最少需要交换多少次？如果这个数字太大，请输出这个最小交换次数对 $99,999,997$取模的结果。

输入样例#1： 复制

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

输出样例#1： 复制

```
1
```

输入样例#2： 复制

```
4
1 3 4 2
1 7 2 4
```

输出样例#2： 复制

```
2
```

题解

可以证明当两个数列都是经过排序之后的数列可以使得$\sum(a_i-b_i)^2$取到最小，

根据上面的结论，就是说如果对于数列$\{a_n\},\{b_n\}$：

$$\begin{matrix} a_1,\ a_2,\ a_3,\ \cdots,\ a_{n-1},\ a_n \\ b_1,\ b_2,\ b_3,\ \cdots,\ b_{n-1},\ b_n \end{matrix} \tag{1}$$

对它们的id排序之后，原数组排序的过程中相当于消除逆序对，但是本来的id是正序的，对于这个过程是对id增加逆序对的数量。可以说就因该是当a数组对于b数组想要移动到一模一样所需要花的时间。但是如何求出这个步骤是一个值得讨论的问题：

首先定于$q[a[i]] = b[i]$，最终的目标是$a[i] = b[i] = t$，即$q[t] = t$。可以发现终极目标就是将q数组进行排序所需要的次数是多少，即求q的逆序对的个数。

代码：

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

using namespace std;

const int maxn = 100005;
const int P = 99999997;
int n;

struct node {
    int id, val;
} a[maxn], b[maxn];

int T[maxn], m[maxn];

int lb(int i) { return i & (-i); }

bool cmp(node x, node y) {
    return x.val < y.val;
}

void add(int i, int delta) {
    while (i <= n) {
        T[i] += delta;
        i += lb(i);
    }
}

int sum(int i) {
    int rtn = 0;
    while (i > 0) {
        rtn += T[i];
        i -= lb(i);
    }
    return rtn;
}

int main() {

    cin >> n;
```

```
    for (int i = 1; i <= n; i ++)
        cin >> a[i].val;
    for (int i = 1; i <= n; i ++) {
        cin >> b[i].val;
        a[i].id = b[i].id = i;
    }
    sort(a + 1, a + 1 + n, cmp);
    sort(b + 1, b + 1 + n, cmp);
    for (int i = 1; i <= n; i ++) {
        m[a[i].id] = b[i].id;
    }
    int ans = 0;
    for (int i = 1; i <= n; i ++) {
        add(m[i], 1);
        ans = (ans + i - sum(m[i])) % P;
    }
    cout << ans << endl;
    return 0;
}
```

P2344 奶牛抗议

题目描述

约翰家的N 头奶牛正在排队游行抗议。一些奶牛情绪激动，约翰测算下来，排在第i 位的奶牛的理智度为Ai，数字可正可负。

约翰希望奶牛在抗议时保持理性，为此，他打算将这条队伍分割成几个小组，每个抗议小组的理智度之和必须大于或等于零。奶牛的队伍已经固定了前后顺序，所以不能交换它们的位置，所以分在一个小组里的奶牛必须是连续位置的。除此之外，分组多少组，每组分多少奶牛，都没有限制。

约翰想知道有多少种分组的方案，由于答案可能很大，只要输出答案除以1000000009的余数即可。

题解：

方程：

$$f(i) = \sum_{0 \le j < i,\ \sum_{t=j+1}^{i} a[t] \ge 0} f(j),\ f(0) = 1 \tag{2}$$

时间复杂度：$O(n^3)$，使用前缀和优化：

$$f(i) = \sum_{0 \le j < i,\ s[i] - s[j] \ge 0} f(j),\ f(0) = 1 \tag{3}$$

时间复杂度：$O(n^2)$，使用树状数组存f，其中下标为s。

对于s数组需要使用离散化，并且0要加入离散化的过程，因为$f(0)=1$。

代码：

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

using namespace std;
const int maxn = 100005;
const int P = 1000000009;
int n, a[maxn],s[maxn];

int T[maxn];

int lb(int i) { return i & (-i); }

void add_sum(int i, int d) {
    // 由于数组的大小加了1，所以要n + 1
    while (i <= n + 1) {
        T[i] = (T[i] + d) % P;
        i += lb(i);
    }
}

int query_sum(int i) {
    int ans = 0;
    while (i > 0) {
        ans = (ans + T[i]) % P;
        i -= lb(i);
    }
    return ans;
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; i ++) {
        cin >> a[i];
        s[i] = s[i-1] + a[i];
        a[i] = s[i];
    }
    // 对s进行离散化
    sort(a, a + n + 1);
    for(int i = 0; i <= n; i ++)
        s[i]=lower_bound(a, a + n + 1, s[i]) - a + 1;
    // 设置f[0] = 1，此处的s[0]表示在原数组中第0号元素在离散化过的数组中的位置
    add_sum(s[0], 1);
    int ans = 0;
    for (int i = 1; i <= n; i ++) {
```

```
        ans = query_sum(s[i]);
        add_sum(s[i], ans);
    }
    cout << ans << endl;
    return 0;
}
```

BZOJ-1782 : 奶牛散步/P2982 [USACO10FEB]

题目描述

约翰有n个牧场，编号为1到n。它们之间有n-1条道路，每条道路连接两个牧场，通过这些道路，所有牧场都是连通的。

1号牧场里有个大牛棚，里面有n头奶牛。约翰会把它们放出来散步。奶牛按编号顺序出发，首先出发的是第一头奶牛，等它到达了目的地后，第二头奶牛才会出发，之后也以此类推。每头奶牛的目的地都不同，其中第iii头奶牛的目的地是t_i号牧场。假如编号较大的奶牛，在经过一座牧场的时候，遇到了一头编号较小的奶牛停在那里散步，就要和它打个招呼。请你统计一下，每头奶牛要和多少编号比它小的奶牛打招呼。

题解

首先这道题可以这样理解：

所有的奶牛从1号到n号依次离开出发到各自的牧场t_i，在经过的道路上如果遇到编号比自己小的牛打招呼，统计总共打多少次招呼。由于编号从小到大，所以只要路径上有牛就肯定会打招呼。

至此，题目的要求的就是对于每头牛，统计在去的路径上有多少头牛。

定义一个农场编号与牛编号的映射关系：$cow[t[i]] = i$。

在dfs整张图的过程中，我们会发现

- 从根节点开始，对树进行深度优先遍历。
- 当进行到节点 i 时，有：
- i 的祖先们 Father[i] 已经被访问过了，但还没有退出。
- 其他节点或者已经被访问过并退出了，或者还没有被访问。
- 那么需要一个数据结构，维护那些**已经被访问过了，但还没有退出的点权**，支持查询**小于特定值的元素的数量** 。
- 可以使用**树状数组**。（使用奶牛编号的下标标记）

所以有以下的代码：

```
#include <iostream>

using namespace std;

```

```cpp
const int maxn = 100005;

int n, p[maxn], head[maxn], cow[maxn], T[maxn], ans[maxn];

int lb(int i) { return i & (-i); }

void modify(int i, int delta) {
    while (i <= n) {
        T[i] += delta;
        i += lb(i);
    }
}

int query(int i) {
    int ret = 0;
    while (i > 0) {
        ret += T[i];
        i -= lb(i);
    }
    return ret;
}

struct edge {
    int to, next;
} g[maxn * 2];

int ecnt = 2;

void add_edge(int u, int v) {
    g[ecnt] = (edge) {v, head[u]};
    head[u] = ecnt ++;
}

void dfs(int u, int fa) {
    int current = cow[u];
    // 这个头牛的答案就是查询：树状数组当中编号比它小的，在路径上的个数和
    ans[current] = query(current);
    // 刚刚进入这个节点(及其子树)，所以把这个节点加入树状数组
    modify(current, 1);
    for (int e = head[u]; e != 0; e = g[e].next)
        if (g[e].to != fa)
            dfs(g[e].to, u);
    // 即将退出该节点，再也不会访问到，所以将其从树状数组中删除
    modify(current, -1);
}

int main() {
    cin >> n;
    for (int i = 1; i < n; i ++) {
```

```
        int a, b; cin >> a >> b;
        add_edge(a, b); add_edge(b, a);
    }
    for (int i = 1; i <= n; i ++) {
        cin >> p[i];
        cow[p[i]] = i;
    }
    dfs(1, 0);
    for (int i = 1; i <= n; i ++) {
        cout << ans[i] << endl;
    }
    return 0;
}
```

P3988 [SHOI2013]发牌

题目描述

在一些扑克游戏里，如德州扑克，发牌是有讲究的。一般称呼专业的发牌手为荷官。荷官在发牌前，先要销牌（burn card）。所谓销牌，就是把当前在牌库顶的那一张牌移动到牌库底，它用来防止玩家猜牌而影响游戏。

假设一开始，荷官拿出了一副新牌，这副牌有N 张不同的牌，编号依次为1到N。由于是新牌，所以牌是按照顺序排好的，从牌库顶开始，依次为1, 2,.....直到N，N 号牌在牌库底。为了发完所有的牌，荷官会进行N 次发牌操作，在第i 次发牌之前，他会连续进行Ri次销牌操作， Ri由输入给定。请问最后玩家拿到这副牌的顺序是什么样的?

举个例子，假设N = 4，则一开始的时候，牌库中牌的构成顺序为{1, 2, 3, 4}。

假设R1=2，则荷官应该连销两次牌，将1 和2 放入牌库底，再将3 发给玩家。目前牌库中的牌顺序为{4, 1, 2}。

假设R2=0，荷官不需要销牌，直接将4 发给玩家，目前牌库中的牌顺序为{1,2}。

假设R3=3，则荷官依次销去了1, 2, 1，再将2 发给了玩家。目前牌库仅剩下一张牌1。

假设R4=2，荷官在重复销去两次1 之后，还是将1 发给了玩家，这是因为1 是牌库中唯一的一张牌。

输入输出格式

输入格式:

第1 行，一个整数N，表示牌的数量。

第2 行到第N + 1 行，在第i + 1 行，有一个整数Ri，0<=Ri<N

输出格式:

第1 行到第N行：第i 行只有一个整数，表示玩家收到的第i 张牌的编号。

输入样例#1： 复制

```
4
2
0
3
2
```

输出样例#1： 复制

```
3
4
2
1
```

题解

维护一个数组，其中存的是每张牌是否还在牌库当中，若在，则值为1，反之，值为0。

每次摸牌，先销牌s张就是在剩下的m张牌中往后寻找s张牌就是了，如果还未找到s就已经为原状态的最后一张了，其实只需要进行对m的牌数进行取模，其实这个想法非常好理解，因为销牌的这个过程是滚动的。

所以我们定义r_0为原来的找牌的“指针”，r_t为找到牌的指针，会有下式：

$$r_t = (s + r_0) \mod m \tag{4}$$

现在，我们来思考一下r_t的意义，其实它就是说剩下的牌中（牌的先后位置关系始终未变，变的是找牌的指针）第r_t张，也就是说我们要找到要维护的数组当中前缀和为r_t的那个位置就是第i张牌的位置，即第i个答案。

从上述的表述可以理解：我们需要维护一个**树状数组**，并**二分答案**。

但是其实有一个比二分答案更为简单的做法，就是模拟lb通过二分的方法访问$T[]$来得到位置，时间复杂度仅为$O(\log n)$，而不是$O(\log^2 n)$。

代码：

```cpp
#include <iostream>
#include <cstdio>
#include <stdio.h>

using namespace std;

const int maxn = 700005;
const int maxx = 1 << 20;

int n, T[maxn];
```

```cpp

inline int lb(int i) { return i & (-i); }

int query(int x) {
    int pos = 0;
    // 第一次查询的区间最大，其后每次减半，相当于二分，但这里访问T数组的时间复杂度为O(1)，故时间复杂度为O(log n)而不是O(log^2 n)
    for (int i = maxx; i > 0; i >>= 1) {
        // 更新位置
        int j = i + pos;
        // 整个过程相当于在进行lb，所以x代表直到pos的前缀和与原来所求的差值，即距query目标还差的一部分
        if (j <= n && T[j] <= x) {
            pos = j;
            x -= T[j];
        }
    }
    return pos + 1;
}

void modify(int i, int d) {
    while (i <= n) {
        T[i] += d;
        i += lb(i);
    }
}

int main() {
    scanf("%d", &n);
    // O(n)时间建树状数组，原因是a[i]=1
    for (int i = 1; i <= n; i ++)
        T[i] = lb(i);
    int r = 0;
    for (int m = n; m > 0; m --) {
        int s; scanf("%d", &s);
        r = (r + s) % m;
        // 在树状数组当中查询值为r的位置，时间复杂度为O(n)
        int pos = query(r);
        // 由于这张牌被发掉了，所以应该将这张牌从树状数组当中移除
        modify(pos, -1);
        // 答案就为这个位置编号
        printf("%d\n", pos);
    }
    return 0;
}
```