主席树学习笔记

Tip：建议完成 Luogu P3919 后阅读。

目录

模板题：静态区间 k 小值

思路引导

首先我们想一想，如何用线段树求**数列** k 小值。

我们可以建一棵权值线段树，由于权值线段树的结点表示的是该结点指向的区间的数的数量，我们就可以直接使用一种类似二分的方法查询 k 小值。

假如你要查询第 5 小的数，如果你发现左儿子有 4 个数，那么就直接放弃左儿子去右儿子查找；反之，如果左儿子有 7 个数，那么就直接放弃右儿子在左儿子查找。

我们把这个方法推广到区间，显然，如果我们拥有一个区间的权值线段树，就可以查询该区间的 k 小值。

如何获得区间的权值线段树呢？

也许，我们可以把这个数列 $a_{1\sim n}$ 的权值线段树看作 $a_1, a_2, ..., a_n$ 各自的权值线段树的**前缀和树**（把前缀和的操作类比到树上），我们用 sum_i 表示 $a_{1\sim i}$ 的权值线段树的前缀和树，用 $tree_{l\sim r}$ 表示区间 $l \sim r$ 的权值线段树。

显然，我们可以利用前缀和的性质来求一个区间的权值线段树，即 $tree_{l\sim r}=sum_r-sum_{l-1}$。

接下来我们引入可持久化线段树，我们可以一个个地插入新的元素，从而获得 $n+1$ 个版本的权值线段树（第 0 个版本是空树）。

这 $n+1$ 棵权值线段树实质上就是前缀和树，接下来我们就可以直接获得任意区间的权值线段树来查询区间 k 小值啦。

值域一般都很大，所以要记得要离散化哦。

Code

时间复杂度 $O(n\log n)$

```
// 44-2 Baijian
#include <algorithm>
#include <iostream>
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

const int N = 1005000;

int n,m,q;
// m 表示值域

int a[N],b[N];

struct Segment{
        int l,r,sum;
}t[N<<2];

int root[N];
```

```cpp
#define lid (t[id].l)
#define rid (t[id].r)

class Segment_Tree{
        int tot = 0;
        private:
                int Clone(int id) {
                        tot ++;
                        t[tot] = t[id];
                        t[tot].sum += 1;
                        return tot;
                }
        public:
                int Build(int id,int l,int r) {
                        id = ++tot;
                        if(l == r) {
                                t[id].sum = a[l];
                                return id;
                        }
                        int mid = (l + r) >> 1;
                        lid = Build(lid,l,mid);
                        rid = Build(rid,mid + 1,r);

                        return id;
                }

                int Update(int id,int l,int r,int val) {
                        id = Clone(id);
                        if(l == r)
                                return id;

                        int mid = (l + r) >> 1;
                        if(val <= mid)
                                lid = Update(lid,l,mid,val);
                        else
                                rid = Update(rid,mid + 1,r,val);
```

```
                                return id;
                        }

                        int Query(int l,int r,int L,int R,int val) {
                                int x = t[t[r].l].sum - t[t[l].l].sum;
                                if(L == R)
                                        return L;
                                int mid = (L + R) >> 1;
                                if(x >= val)
                                        return
Query(t[l].l,t[r].l,L,mid,val);
                                else
                                        return Query(t[l].r,t[r].r,mid +
1,R,val - x);
                        }
}tree;

int main() {
        ios::sync_with_stdio(false);
        cin >> n >> q;
        for(int i = 1;i <= n; i++) {
                cin >> a[i];
                b[i] = a[i];
        }

        sort(b + 1,b + n + 1);
        m = unique(b + 1,b + n + 1) - (b + 1);
        // 离散化

        tree.Build(root[0],1,m);
        for(int i = 1,x;i <= n; i++) {
                x = lower_bound(b + 1,b + m + 1,a[i]) - b;
                root[i] = tree.Update(root[i - 1],1,m,x);
        }

        for(int i = 1,l,r,k,ans;i <= q; i++) {
                cin >> l >> r >> k;
```

```
                ans = tree.Query(root[l - 1],root[r],1,m,k);
                cout << b[ans] << "\n";
        }
        return 0;
}
```

模板题：动态区间 k 小值

题目大意

与上一道模板相比，多了一个把 a_x 修改为 y 的操作。

思路引导

实际上，动态主席树是树套树，不过动态主席树用到了与静态主席树类似的思想，即维护前缀和。

我们先来看看单点修改。

假设要将 a_x 修改为 y，它将影响到根为 $root_{x\sim n}$ 的前缀和树。要是暴力的话，我们就直接把 x 到 n 的前缀和树修改一下，a_x 位置的值 -1，y 位置的值 $+1$。

这样的复杂度显然不满足要求。

我们要维护一个前缀和，我们过去好像用树状数组维护过单点修改的前缀和，利用 lowbit 可以以 $\log n$ 的复杂度进行区间修改，原先 $O(n)$ 的外层修改优化成了 $O(\log n)$。

那我们就可以写一个树状数组套主席树。

Code

时间复杂度：$O(n\log^2 n)$，比线段树套平衡树少个 $\log$。

实现的思路是将修改操作用一棵主席树存，在查询时与初始的主席树一并操作。

```cpp
// 44-9 Baijian
#include <algorithm>
#include <iostream>
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

const int N = 6605000;

int n,m;
int tot = 0;

int a[N],b[N],cnt;

int cnt1,cnt2;
int Left[N],Right[N];

struct Option{
        int op;
        int l,r,k,x,y;
}q[N];

int Find(int x) {
        // Discretizing
        return (lower_bound(b + 1,b + cnt + 1,x) - b);
}

// Segment Tree

struct Segment{
        int l,r,sum;
}t[N<<2];

int root[N],rt[N];
```

```
class Segment_Tree{
        public:
                void Update(int id1,int &id2,int l,int r,int
pos,int val) {
                        if(!id2)
                                id2 = ++ tot;
                        t[id2].sum = t[id1].sum + val;
                        if(l == r)
                                return ;

                        int mid = (l + r) >> 1;
                        if(pos <= mid) {
                                t[id2].r = t[id1].r;

Update(t[id1].l,t[id2].l,l,mid,pos,val);
                        }
                        else {
                                t[id2].l = t[id1].l;
                                Update(t[id1].r,t[id2].r,mid +
1,r,pos,val);
                        }

                        return ;
                }

                int Query(int l,int r,int L,int R,int val) {
                        if(L == R)
                                return L;

                        int sum1 = 0,sum2 = 0;
                        for(int i = 1;i <= cnt1; i++)
                                sum1 += t[t[Left[i]].l].sum;
                        for(int i = 1;i <= cnt2; i++)
                                sum2 += t[t[Right[i]].l].sum;
                        int x = t[t[r].l].sum - t[t[l].l].sum +
sum2 - sum1;
```

```
                        int mid = (L + R) >> 1;
                        if(x >= val) {
                                for(int i = 1;i <= cnt1; i++)
                                        Left[i] = t[Left[i]].l;
                                for(int i = 1;i <= cnt2; i++)
                                        Right[i] = t[Right[i]].l;

                                return
Query(t[l].l,t[r].l,L,mid,val);
                        }
                        else {
                                for(int i = 1;i <= cnt1; i++)
                                        Left[i] = t[Left[i]].r;
                                for(int i = 1;i <= cnt2; i++)
                                        Right[i] = t[Right[i]].r;

                                return Query(t[l].r,t[r].r,mid +
1,R,val - x);
                        }
                }
}tree;

// BIT

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

void Add(int l,int r,int pos1,int pos2) {
        for(int i = l;i <= r; i += lowbit(i))
                tree.Update(root[i],root[i],1,cnt,pos1,-1);
        for(int i = l;i <= r; i += lowbit(i))
                tree.Update(root[i],root[i],1,cnt,pos2,1);

        return ;
}
```

```
void Get(int l,int r) {
        cnt1 = cnt2 = 0;
        for(int i = l - 1;i; i -= lowbit(i))
                Left[++cnt1] = root[i];
        for(int i = r;i; i -= lowbit(i))
                Right[++cnt2] = root[i];

        return ;
}

void Clear() {
        for(int i = 1;i <= tot; i++)
                t[i].l = t[i].r = t[i].sum = 0;
        for(int i = 1;i <= n; i++)
                rt[i] = root[i] = 0;
        tot = 0;
        cnt = 0;
        return ;
}

// main

signed main() {
        ios::sync_with_stdio(false);
        int T;
        cin >> T;
        while(T--) {
                cin >> n >> m;
                for(int i = 1;i <= n; i++) {
                        cin >> a[i];
                        b[++cnt] = a[i];
                }

                char opt;
                for(int i = 1;i <= m; i++) {
                        cin >> opt;
```

```
                if(opt == 'Q') {
                        q[i].op = 1;
                        cin >> q[i].l >> q[i].r >> q[i].k;
                }
                else {
                        q[i].op = 2;
                        cin >> q[i].x >> q[i].y;
                        b[++cnt] = q[i].y;
                }
        }

        // Discretizing
        sort(b + 1,b + 1 + cnt);
        cnt = unique(b + 1,b + 1 + cnt) - (b + 1);

        for(int i = 1;i <= n; i++)
                tree.Update(rt[i -
1],rt[i],1,cnt,Find(a[i]),1);
        for(int i = 1;i <= m; i++) {
                if(q[i].op == 1) {
                        Get(q[i].l,q[i].r);
                        int ans;
                        ans = tree.Query(rt[q[i].l -
1],rt[q[i].r],1,cnt,q[i].k);
                        cout << b[ans] << "\n";
                }
                else {

Add(q[i].x,n,Find(a[q[i].x]),Find(q[i].y));
                        a[q[i].x] = q[i].y;
                }
        }
        Clear();
    }
    return 0;
}
```

花神的嘲讽计划

简明题意

给定一个长度为 n 的数列 a，进行 m 次询问，每次询问一个区间中是否存在一个长度为 k 的子数列（每次询问的 k 值相等）。

思路

判断两个区间是否相等，显然我们可以用 hash。

我们把一个长度为 k 的区间的 hash 值看作一个元素，这道题实际上就转化为了求给定区间中是否存在某个数。

由于每个测试点中 k 的值是一定的，那么就可以直接预处理得到整个数列中任意连续 k 个数的 hash，然后把它塞进树。

我们可以用主席树获取任意区间的权值线段树，直接在查询区间的权值线段树中寻找查询数列的 hash 即可。

Code

```
// 44 - 1 BZOJ3207 Baijian
#include <algorithm>
#include <iostream>
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

const int N = 4000500;

int n,m,k;

// Hash
```

```
typedef unsigned long long ULL;
ULL base = 131;
ULL Hash[N],p[N];

ULL Get(int l,int r) {
        return Hash[r] - Hash[l - 1] * p[r - l + 1];
}

// Segment Tree

struct Segment{
        int l,r,sum;
}t[N<<2];

int root[N];

#define lid (t[id].l)
#define rid (t[id].r)

class Segment_Tree{
        int tot = 0;
        private:
                int Clone(int id) {
                        tot ++;
                        t[tot] = t[id];
                        t[tot].sum += 1;
                        return tot;
                }
        public:
                int Update(int id,ULL l,ULL r,ULL val) {
                        id = Clone(id);
                        if(l >= r)
                                return id;
                        ULL mid = l + ((r - l) >> 1);
                        if(val <= mid)
                                lid = Update(lid,l,mid,val);
```

```cpp
                else
                        rid = Update(rid,mid + 1,r,val);

                return id;
        }

        bool Query(int l,int r,ULL L,ULL R,ULL val) {
                if(L >= R) {
                        if(t[r].sum != t[l].sum)
                                return 1;
                        return 0;
                }
                int x = t[r].sum - t[l].sum;
                if(!x)
                        return 0;
                ULL mid = L + ((R - L) >> 1);
                if(val <= mid)
                        Query(t[l].l,t[r].l,L,mid,val);
                else
                        Query(t[l].r,t[r].r,mid + 1,R,val);
        }
}tree;

int main() {
        ios::sync_with_stdio(false);
        cin >> n >> m >> k;
        for(int i = 1,x;i <= n; i++) {
                cin >> x;
                Hash[i] = base * Hash[i - 1] + x;
        }
        p[0] = 1;
        for(int i = 1;i <= n; i++)
                p[i] = p[i - 1] * base;

        for(int i = k;i <= n; i++) {
                ULL x = Get(i - k + 1,i);
                root[i] = tree.Update(root[i - 1],0,ULLONG_MAX,x);
```

```
        }

        for(int i = 1,l,r;i <= m; i++) {
                cin >> l >> r;

                ULL tmp = 0;
                for(int j = 1,x;j <= k; j++) {
                        cin >> x;
                        tmp = tmp * base + x;
                }
                if(tree.Query(root[l + k -
2],root[r],0,ULLONG_MAX,tmp))
                        cout << "No\n";
                else
                        cout << "Yes\n";
        }
        return 0;
}
```

疯狂的颜色序列

题目大意

HH 的项链强制在线版。

给定一个颜色序列，询问 $[l, r]$ 之间出现了多少种颜色，强制在线。

思路

和 HH 的项链思路一致。

依次遍历这个数列，若当前数在前面出现过，就把前面那个数的位置 -1，把当前位置 $+1$；若当前数在前面没有出现过，就直接把当前位置 $+1$。

Code

```cpp
// 44-3 Baijian
#include <algorithm>
#include <iostream>
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

const int N = 5005000;

int n,m;

int last[N],ans;

struct Segment{
	int l,r,sum;
}t[N<<2];

int root[N];

#define lid (t[id].l)
#define rid (t[id].r)

class Segment_Tree{
	int tot = 0;
	private:
		int Clone(int id,int val) {
			tot ++;
			t[tot] = t[id];
			t[tot].sum += val;
			return tot;
		}
	public:
		int Update(int id,int l,int r,int pos,int val) {
			id = Clone(id,val);
```

```
            if(l >= r)
                return id;

            int mid = (l + r) >> 1;
            if(pos <= mid)
                lid = Update(lid,l,mid,pos,val);
            else
                rid = Update(rid,mid +
1,r,pos,val);

            return id;
        }

        int Query(int id,int l,int r,int pos) {
            if(l == r)
                return t[id].sum;
            int mid = (l + r) >> 1;
            if(pos <= mid)
                return Query(lid,l,mid,pos) +
t[rid].sum;
            else
                return Query(rid,mid + 1,r,pos);
            /*
            * 需要向左儿子递归时
            * 我们发现右儿子肯定在我们查询区间里
            * （因为你是从 root[r] 找的啊）
            * 所以要加上 t[rid].sum
            */
        }
}tree;

int main() {
    ios::sync_with_stdio(false);
    cin >> n >> m;
    for(int i = 1,col;i <= n; i++) {
        cin >> col;
        if(!last[col]) {
```

```
                        root[i] = tree.Update(root[i - 1],1,n,i,1);
                        last[col] = i;
                }
                else {
                        int tmp = tree.Update(root[i -
1],1,n,last[col],-1);
                        root[i] = tree.Update(tmp,1,n,i,1);
                        last[col] = i;
                }
        }

        for(int i = 1,l,r;i <= m; i++) {
                cin >> l >> r;
                l = (l + ans) % n + 1;
                r = (r + ans) % n + 1;
                if(l > r)
                        swap(l,r);
                ans = tree.Query(root[r],1,n,l);
                cout << ans << "\n";
        }
        return 0;
}
```

SP10628：Count on a tree

题目大意

给定一棵 n 个节点的树，每个点有一个权值。有 m 个询问，每次给你 u, v, k，你需要回答 $u \operatorname{xor} \text{last}$ 和 v 这两个节点间第 k 小的点权（其中 last 是上一个询问的答案）。

思路

这不是树上前缀和？由于主席树拥有前缀和的性质，所以我们可以直接树上差分获取 u 到 v 这条路径的权值线段树，然后在这个权值线段树中查找区间

k 小值。

前置芝士：LCA，树上差分

Code

```
// P2633
#include <algorithm>
#include <iostream>
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

const int N = 9005000;

int n,m,S;
int a[N],b[N];
int last = 0;

// Discretizing
int Find(int x) {
        return (lower_bound(b + 1,b + S + 1,x) - b);
}

// Graph

struct Edge{
        int next,to;
}e[N << 1];

int cnt,h[N];

void add(int u,int v) {
        cnt++;
```

```
        e[cnt].next = h[u];
        h[u] = cnt;
        e[cnt].to = v;
}

// Segment_Tree
struct Segment{
        int l,r,sum;
}t[N<<1];

int tot = 0;

class Segment_Tree{
        public:
                void Build(Segment &id,int l,int r) {
                        id.sum = 0;
                        if(l == r)
                                return ;

                        int mid = (l + r)>>1;
                        id.l = ++tot;
                        Build(t[id.l],l,mid);
                        id.r = ++tot;
                        Build(t[id.r],mid + 1,r);

                        return ;
                }

                void Update(Segment pre,Segment &now,int l,int
r,int pos) {
                        now.sum = pre.sum + 1;
                        if(l == r)
                                return ;
                        int mid = (l + r) >> 1;
                        if(pos <= mid) {
                                now.l = ++tot;
```

```
Update(t[pre.l],t[now.l],l,mid,pos);
                                now.r = pre.r;
                        }
                        else {
                                now.r = ++tot;
                                Update(t[pre.r],t[now.r],mid +
1,r,pos);
                                now.l = pre.l;
                        }
                        return ;
                }

                int Query(Segment a,Segment b,Segment c,Segment
d,int l,int r,int k) {
                        // u v lca(u,v) fa[lca(u,v)]
                        if(l == r)
                                return l;
                        int x = t[a.l].sum + t[b.l].sum -
t[c.l].sum - t[d.l].sum;
                        int mid = (l + r) >> 1;
                        if(x >= k)
                                return
Query(t[a.l],t[b.l],t[c.l],t[d.l],l,mid,k);
                        else
                                return
Query(t[a.r],t[b.r],t[c.r],t[d.r],mid + 1,r,k - x);
                }
}tree;

// lca

int root[N];

int dep[N];
int f[N][40];

void dfs(int u,int pre) {
```

```cpp
        root[u] = ++tot;
        tree.Update(t[root[pre]],t[root[u]],1,S,Find(a[u]));

        dep[u] = dep[pre] + 1;
        f[u][0] = pre;
        for(int i = 1;i <= 26; i++)
                f[u][i] = f[f[u][i - 1]][i - 1];

        for(int i = h[u];i; i = e[i].next) {
                int v = e[i].to;
                if(v != pre)
                        dfs(v,u);
        }
}

int lca(int x,int y) {
        if(dep[x] < dep[y])
                swap(x,y);
        for(int i = 26;i >= 0; i--)
                if(dep[f[x][i]] >= dep[y])
                        x = f[x][i];
        if(x == y)
                return x;

        for(int i = 26;i >= 0; i--)
                if(f[x][i] != f[y][i]) {
                        x = f[x][i];
                        y = f[y][i];
                }

        return f[x][0];
}


int main() {
        ios::sync_with_stdio(false);
        cin >> n >> m;
```

```
        for(int i = 1;i <= n; i++) {
                cin >> a[i];
                b[i] = a[i];
        }
        for(int i = 1,x,y;i < n; i++) {
                cin >> x >> y;
                add(x,y);
                add(y,x);
        }

        // Discretizing
        sort(b + 1,b + n + 1);
        S = unique(b + 1,b + n + 1) - (b + 1);

        root[0] = ++tot;
        tree.Build(t[root[0]],1,S);
        dfs(1,0);

        for(int i = 1,u,v,k;i <= m; i++) {
                cin >> u >> v >> k;
                u = u ^ last;
                int _lca = lca(u,v);
                int fa_lca = f[_lca][0];
                int ans =
b[tree.Query(t[root[u]],t[root[v]],t[root[_lca]],t[root[fa_lca]],1,
S,k)];
                // 路径的权值线段树中查找
                cout << ans << "\n";
                last = ans;
        }
        return 0;
}
```

[SDOI2013] 森林

题目大意

多了一个在点 x 和点 y 之间连边的操作，保证操作后仍然是一片森林。

对于查询操作，保证 x 和 y 连通，且二者之间路径上至少有 k 个点。

思路

恼！

查询？ 主席树？

连边？ LCT？

~~嗯，可以搞~~

5 hours later . . .

只有连边操作，就是合并两棵树，我们来想想启发式合并。

我们维护一下每个点所处的集合的大小，在连边的时候，直接把小的集合合并到大的上面（感觉好暴力）。

Code

复杂度 $O(\text{能过})$，代码和上面那道题代码一脉相承~~（就是复制下来的）~~

~~但事实告诉我们，最好不要粘上一道题代码o(>﹏<)o~~

```
// 44-4
#include <algorithm>
#include <iostream>
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>

using namespace std;

const int N = 5005000;

int testcase;
```

```
int n,m,T;

int a[N],b[N];

int len,size[N];

// Graph
struct Edge{
        int next,to;
}e[N<<1];

int h[N],cnt;

void Add(int u,int v) {
        cnt ++;
        e[cnt].next = h[u];
        h[u] = cnt;
        e[cnt].to = v;
}

// Segment Tree
struct Segment{
        int l,r,sum;
}t[N<<1];

int tot = 0,root[N];

class Segment_Tree{
        public:
                void Update(Segment pre,Segment &now,int l,int
r,int pos) {
                        now.sum = pre.sum + 1;

                        if(l == r)
                                return ;
                        int mid = (l + r) >> 1;
```

```
                        if(pos <= mid) {
                                now.l = ++tot;

Update(t[pre.l],t[now.l],l,mid,pos);
                                now.r = pre.r;
                        }
                        else {
                                now.r = ++tot;
                                Update(t[pre.r],t[now.r],mid +
1,r,pos);
                                now.l = pre.l;
                        }
                        return ;
                }

                int Query(Segment a,Segment b,Segment c,Segment
d,int l,int r,int k) {
                        // u v lca(u,v) fa[lca(u,v)]
                        if(l == r)
                                return l;
                        int x = t[a.l].sum + t[b.l].sum -
t[c.l].sum - t[d.l].sum;
                        int mid = (l + r) >> 1;
                        if(x >= k)
                                return
Query(t[a.l],t[b.l],t[c.l],t[d.l],l,mid,k);
                        else
                                return
Query(t[a.r],t[b.r],t[c.r],t[d.r],mid + 1,r,k - x);
                }
}tree;

// Discretizing
int Find(int x) {
        return (lower_bound(b + 1,b + len + 1,x) - b);
}
```

```cpp
int dep[N];
int f[N][40];

int lca(int x,int y) {
        if(dep[x] < dep[y])
                swap(x,y);
        for(int i = 26;i >= 0; i--)
                if(dep[f[x][i]] >= dep[y])
                        x = f[x][i];
        if(x == y)
                return x;

        for(int i = 26;i >= 0; i--)
                if(f[x][i] != f[y][i]) {
                        x = f[x][i];
                        y = f[y][i];
                }

        return f[x][0];
}

bool vis[N];
int fat[N];

void dfs(int x,int fa,int Anc) {
        vis[x] = 1;
        fat[x] = Anc;
        // ancestor
        // number of set
        f[x][0] = fa;
        dep[x] = dep[fa] + 1;

        root[x] = ++ tot;
        tree.Update(t[root[fa]],t[root[x]],1,n,Find(a[x]));

        for(int i = 1;i <= 26; i++)
        f[x][i] = f[f[x][i - 1]][i - 1];
```

```
        for(int i = h[x];i; i = e[i].next) {
                int to = e[i].to;
                if(to == fa)
                        continue;
                dfs(to,x,Anc);
                size[x] += size[to];
        }
        return ;
}

int last = 0;

int main() {
        ios::sync_with_stdio(false);
        cin >> testcase;
        cin >> n >> m >> T;
        for(int i = 1;i <= n; i++) {
                cin >> a[i];
                b[i] = a[i];
                size[i] = 1;
        }
        for(int i = 1,x,y;i <= m; i++) {
                cin >> x >> y;
                Add(x,y);
                Add(y,x);
        }

        sort(b + 1,b + n + 1);
        len = unique(b + 1,b + n + 1) - (b + 1);

        for(int i = 1;i <= n; i++)
                if(!vis[i])
                        dfs(i,0,i);

        char op;
        for(int i = 1,x,y,k;i <= T; i++) {
```

```
                cin >> op;
                cin >> x >> y;
                x = x xor last;
                y = y xor last;

                if(op == 'Q') {
                        cin >> k;
                        k = k xor last;

                        int _lca = lca(x,y);
                        int fa_lca = f[_lca][0];

                        last =
b[tree.Query(t[root[x]],t[root[y]],t[root[_lca]],t[root[fa_lca]],1,
n,k)];
                        cout << last << "\n";
                }
                else {
                        Add(x,y);
                        Add(y,x);

                        if(size[fat[x]] < size[fat[y]]) {
                                size[fat[y]] += size[fat[x]];
                                dfs(x,y,fat[y]);
                        }
                        else {
                                size[fat[x]] += size[fat[y]];
                                dfs(y,x,fat[x]);
                        }
                        // Seemed like a Brute force
                }
        }
        return 0;
}
```

END