```
// 设置输出宽度和填充字符
cout << setw(N) << setfill(s) << a;
int gcd(int t1, int t2)   //求最大公约数
{
	return t2 == 0 ? t1 : gcd(t2, t1 % t2);
}

int lcm(int a, int b)     // 求最小公倍数
{
	return a*b/gcd(a,b);
}

// 根据后序和中序遍历输出层序遍历
#include<vector>
vector<int>post, in, level(100000, -1);   //post 存储后序遍历，  in 存储中序遍历
int N;
void ergodic(int root, int start, int end, int index) {
	int i = start;
	if (start > end)
		return;
	while (i < end&&in[i] != post[root])
		i++;
	level[index] = post[root];
	//cout << root << start << end << i << endl;
	ergodic(root - 1 - end + i, start, i - 1, 2 * index + 1);
	ergodic(root - 1, i + 1, end, 2 * index + 2);
}
int main() {
	cin >> N;
	post.resize(N);
	in.resize(N);
	for (int i = 0; i < N; i++)
		cin >> post[i];
	for (int i = 0; i < N; i++)
		cin >> in[i];
	ergodic(N - 1, 0, N - 1, 0);
	for (int i = 0, cnt = 0; i < level.size(); i++) {
		if (level[i] != -1) {
			cout << level[i];
			cnt != N - 1 ? cout << ' ' : cout << endl;
			cnt++;}}
	return 0;}

//由前序和中序遍历得到树，  输出反转后的树
```

```
int mid[1000], pre[1000];

struct node{
	int data;
	node *left;
	node * right;
};

node * create(int q, int z, int n) //  创建树
{
	node * T;
	int i;
	if(n <= 0)
	{
		T = NULL;
	}
	else
	{
		T = new node;
		T -> data = pre[q];
		for(i=0; mid[i+z] != pre[q]; i++);
		T -> left = create(q+1, z, i);
		T -> right = create(q+i+1, z+i+1, n-i-1);}
	return T;
}

node * change(node * T) //转换树
{
	node * t;
	if(T)
	{
		if(T->left != NULL || T->right != NULL)
		{
			t = T -> left;
			T -> left = T -> right;
			T -> right = t;
		}
		change(T -> left);
		change(T -> right);
	}
	return T;}

void print(node *T, int n) //层次输出树
{
```

```
    node * q[100];
    node * p;
    int f = 0, r = 0, cnt = 0;
    if(T)
    {
        q[r++] = T;
        while(f != r)
        {
            p = q[f++];
            cout << p -> data;
            cnt++;
            if(cnt != n)
                cout << " ";
            else
                cout << endl;
            if(p -> left != NULL)
            {
                q[r++] = p -> left;
            }
            if(p -> right != NULL)
            {
                q[r++] = p -> right;
            }}}}
int main()
{
    node * T;
    int i, n;
    cin >> n;
    for(int i=0; i<n; i++)
    {
        cin >> mid[i];
    }
    for(int i=0; i<n; i++)
    {
        cin >> pre[i];
    }
    T = create(0,0,n);
    T = change(T);
    print(T, n);
    system("pause");
    return 0;
}
//小顶堆 二叉树，从小到大排序
const int INF = 1000000;
```

```
int a[1010], n, m;
int insert(int i)
{
	int temp;
	while(a[i/2]>a[i] && i!= 1)
	{
		temp = a[i];
		a[i] = a[i/2];
		a[i/2] = temp;
		i = i/2;
	}}

int find(int x)   // 找爸爸
{
	int p, i;
	for(i=1; i<=n; i++)
	{
		if(a[i] == x)
			p = i;
	}
	return a[p/2];
}

int panduan()
{
	string str,str1,str2;
	int x,y;
	char ch;
	cin >> x >> str;
	if(str == "and")
	{
		cin >> y >> str1 >> str2;
		if(find(x) == find(y))
			cout << "T" << endl;
		else
			cout << "F" << endl;
		return 0;
	}
	cin >> str;
	if(str == "a")
	{
		cin >> str1 >> str2 >> y;
		if(find(x) == y)
			cout << "T" << endl;
```

```
            else
                cout << "F" << endl;
            return 0;
        }
        cin >> str;
        if(str == "root")
        {
            if(a[1] == x)
                cout << "T" << endl;
            else
                cout << "F" << endl;
            return 0;
        }
        cin >> str1 >> y;
        if(find(y) == x)
            cout << "T" << endl;
        else
            cout << "F" << endl;
        return 0;
}
int main()
{
        cin >> n >> m;
        cin >> a[1];
        for(int i=2; i<=n;i++)
        {
            cin >> a[i];
            insert(i);
        }
        while(m--)
            panduan();
        system("pause");
        return 0;
}

//并查集
int f[100];
void init()
{
        for(int i=1;i<=N;i++)
        {
            f[i] = i;
        }
}
```

```
void join(int x, int y)
{
	int fx = find(x); //find 会返回 x 的根节点
	int fy = find(y);
	if(fx != fy)
	{
		f[fx] = fy;
	}
}

int find(int x)
{
	if(f[x] != x)
	{
		return f[x] = find(f[x]);
	}
	return x;
}

//五服之内不得通婚
struct infor
{
	char sex;
	int father;
	int mother;
};

infor r[100100];

int visit[200001];
int flag;

void find(int a, int sum)
{
	if(sum>5 || a==-1 || a==0)
	{
		return ;//如果超过五代或者没有父亲或母亲返回
	}
	visit[a]++;
	if(visit[a]>=2)//说明五代以内有重叠的亲人
		flag = 0;
	find(r[a].father,sum+1);
	find(r[a].mother,sum+1);
```

```
        return;
}

int main(){
        int n;
        int myid, fid, mid;
        char c;
        cin >> n;
        for(int i=0; i<n; i++)
        {
                cin >> myid >> c >> fid >> mid;
                r[myid].sex = c;
                r[myid].father = fid;
                r[myid].mother = mid;
                r[fid].sex = 'M';
                r[mid].sex = 'F';
        }
        int m;
        int data1, data2;
        cin >> m;
        while(m--)
        {
                flag = 1;
                memset(visit,0,sizeof(visit));
                cin >> data1 >> data2;
                if(r[data1].sex == r[data2].sex)
                {
                        cout << "Never Mind" << endl;
                        continue;
                }
                find(data1,1);
                find(data2,1);
                if(flag)
                        cout << "Yes" << endl;
                else
                        cout << "No" << endl;

        }
        return 0;
}

//多项式除法
#include<iostream>
#include<cmath>
```

```
using namespace std;
int main(){
    int a,b;int ma,mb;int zhi,xi;double c[100000];
    cin>>a;
    double* data1 = new double[1000000]();
    for(int i=0;i<a;i++){//数组的索引为多项式的指数，数组的内容为数组的系数
        cin>>zhi;
        if(i == 0)ma = zhi;
        cin>>xi;
        data1[zhi] = xi;
    }
    cin>>b;
    double* data2 = new double[1000000]();
    for(int i=0;i<b;i++){
        cin>>zhi;
        if(i == 0)mb = zhi;
        cin>>xi;
        data2[zhi] = xi;
    }
    for(int i=ma;i>=mb;i--){
        c[i-mb] = data1[i]/data2[mb];//i-mb 为商的指数
        for(int j=mb;j>=0;j--){
            data1[i+j-mb] -= data2[j]*c[i-mb];//除数每一项都乘以商
        }
    }
    int num1 = 0,num2 = 0;
    for(int i=ma-mb;i>=0;i--){
        if(abs(c[i]) >= 0.1)num1++;
    }
    for(int i=mb-1;i>=0;i--){
        if(abs(data1[i]) >= 0.1)num2++;
    }

    if(num1 == 0){
        printf("0 0 0.0\n");
    }else{
        printf("%d",num1);
        for(int i=ma-mb;i>=0;i--){
            if(abs(c[i]) >=0.1)printf(" %d %.1lf",i,c[i]);
        }
        printf("\n");
    }

    if(num2 == 0){
```

```
            printf("0 0 0.0\n");
        }else{
            printf("%d",num2);
            for(int i=mb-1;i>=0;i--){
                if(abs(data1[i])>0.1)printf(" %d %.1lf",i,data1[i]);
            }
        }

        delete data1;
        delete data2;
        return 0;
}

//单链表逆置
typedef struct _NODE_
{
        int data;
        struct _NODE_ *next;
} NODE;

void Reverse(NODE * head)
{
         if (head == NULL || head->next == NULL) {
                return;
            }
     NODE * ends = head -> next;
     NODE * change = head -> next -> next;
     NODE * temporary = ends;
     while(change != NULL)
     {
          ends -> next = change -> next;
          change -> next = temporary;
          temporary = change;
          head -> next = change;
          change = ends -> next;

     }

}

//单链表逆置 2
typedef struct Node *PtrToNode;
typedef int ElementType;
struct Node {
```

```
    ElementType Data; /* 存储结点数据 */
    PtrToNode     Next; /* 指向下一个结点的指针 */
};
typedef PtrToNode List; /* 定义单链表类型 */

List Reverse(List L)
{
    List p1 = NULL, p2 = NULL;
    while(L)
    {
        p2 = L -> Next;
        L -> Next = p1;
        p1 = L;
        L   = p2;
    }
    return p1;
}

//循环节点找到循环入口
class Node{
public:
    Node* next;
    Node(Node* p=NULL){
        next = p;
    }
};

int find_cycling_position ( Node* head )
{
    Node * faster = head;
    Node * slower = head;
    int count = 1;
    while(faster != NULL && faster->next != NULL) //第一次在环中相遇
    {
        faster = faster -> next -> next;
        slower = slower -> next;
        if(faster == slower)
        {
            faster = head;
            while(faster != slower) //第二次相遇就是入口结点
            {
                faster = faster -> next;
                slower = slower -> next;
                count ++;
```

```
                }
                return count;
            }
        }
    }
```