藏尾诗:

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    int n = 4;
    while(n--)
    {
        cin >> s;
        cout << &(s[s.size()-2]);//利用指针的原理
    }
    return 0;
}
```

正整数A+B:

```
#include<iostream>
#include<cstring>
using namespace std;

int toInt(string s);

int main()
{
    string x, y, s;
    cin >> x;//第一个串读到第一个空格为止
    getchar();//吸收空格
    getline(cin, y); //第二个串读到换行符为止
    int a = toInt(x);
    int b = toInt(y);
    if(a==-1) cout << "? + ";
    else cout << a << " + ";
    if(b==-1) cout << "? = ";
    else cout << b << " = ";
    if(a==-1||b==-1) cout << "?";
    else cout << a+b;
    return 0;
}

int toInt(string s)
{//把s转换为整数，若s不满足格式要求，或者转换后不在[1,1000]，转为-1。
    if(s.size()>4) return -1;
    int res = 0;
    for(int i=0; i<s.size(); i++)
    {
        if(!isdigit(s[i])) return -1;
        res = res*10 + s[i]-48;
    }
    if(res==0 || res>1000) return -1;
    return res;
```

```
}
```

符号配对：

碰到左边入栈，碰到右边就判断，不满足就输出结果。右边全部满足 && 最后栈空 才是真正的配对

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    stack<char> st;
    map<char, int> mp;
    mp['('] = 0;
    mp['['] = 1;
    mp['{'] = 2;
    mp[')'] = 0;
    mp[']'] = -1;
    mp['}'] = -2;
    int flag = 1;
    char res;
    while(1)
    {
        getline(cin, s);
        if(s=="." || flag!=1) break;
        for(int i=0; s[i]; i++)
        {
            if(s[i]=='('||s[i]=='['||s[i]=='{')
            {
                st.push(s[i]);
                continue;
            }
            else if(s[i]=='/' && s[i+1] && s[i+1]=='*')
            {
                st.push('/');
                i++;
                continue;
            }
            if(s[i]==')'||s[i]==']'||s[i]=='}')
            {
                if(st.empty())
                {
                    flag = 0;//缺左边
                    cout << "NO\n" << "?-" << s[i];
                    break;
                }
                else if(mp[st.top()]+mp[s[i]]!=0)
                {
                    flag = 0;//缺右边
                    cout << "NO\n" << st.top() << "-?";
                    break;
                }
                st.pop();
            }
            else if(s[i]=='*' && s[i+1] && s[i+1]=='/')
            {
```

```
                if(st.empty())
                {
                    flag = 0;//缺左边
                    cout << "NO\n" << "?-" << "*/";
                    break;
                }
                else if(st.top()!='/')
                {
                    flag = 0;//缺右边
                    cout << "NO\n" << st.top() << "-?";
                    break;
                }
                st.pop();
                i++;
            }
        }
    }
    if(flag==1)
        if(!st.empty())//检查有没有未匹配的左边
        {
            cout << "NO\n";
            if(st.top()=='/') cout << "/*-?";
            else cout << st.top() << "-?";
        }
        else cout << "YES";
    return 0;
}
```

乘法口诀数列：

三个关键变量：
i：当前要输出的下标，j：当前要写入数组的下标，cur：当前要参与乘法运算的下标

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int a[3000];
    int n;
    cin >> a[0] >> a[1] >> n;
    int cur = 0, j = 2;
    for(int i=0; i<n; i++)
    {
        if(i>0) cout << " ";
        cout << a[i];
        a[j] = a[cur]*a[cur+1];
        if(a[j]>9) a[j+1]=a[j]%10, a[j]/=10, j++;
        j++;
        cur++;
    }
}
```

字符串A+B：

一边读入一边判断一边输出，使用has数组标记该字符是否出现过。又是一道以数据作为下标的题目。

```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    string s;
    int has[200] = {0};
    string res = "";
    for(int i=0; i<2; i++)
    {
        getline(cin, s);
        for(int j=0; j<s.size(); j++)
            if(has[s[j]]==0)
            {
                res += s[j];
                has[s[j]] = 1;
            }
    }
    cout << res;
    return 0;
}
```

分而治之：

检查每条边的两个顶点是否都没有被打击 如果都没有被打击，则说明这两个城市保持连通，打击失败

```
#include<iostream>
using namespace std;

int main()
{
    int n, m;
    int i, j, k, t, x;
    int a[10001][2];
    bool flag;
    cin >> n >> m; //顶点数、边数
    for(i=1; i<=m; ++i) //m条边
        cin >> a[i][0] >> a[i][1]; //每条边的两个顶点

    cin >> t; //打击方案数
    for(i=1; i<=t; ++i){
        bool b[10001] = {false};
        cin >> k; //第m个方案要打击k个城市
        for(j=1; j<=k; ++j)
        { //读入k个具体的城市编号到b数组
            cin >> x;
            b[x] = true;
        }
        flag = true;
        for(j=1; j<=m; ++j)
        {
```

```
            //检查每条边的两个顶点是否出现在b数组
            //若都不出现，则说明这两个城市保持连通，打击失败
            if(!b[a[j][0]]&&!b[a[j][1]])
            {
                flag = false;
                break;
            }
        }
        if(flag) cout << "YES" << endl;
        else cout << "NO" << endl;
    }
}
```

排座位：

使用并查集
注意：A-朋友-朋友-...-朋友-B，但是A和B是敌人。这种情况下A和B会在同一个集合，正确输出是OK but...。

```
#include<bits/stdc++.h>
using namespace std;

int pre[101];


int root(int x)
{
    while(pre[x]!=x) x = pre[x];
    return x;
}

void un(int x, int y)
{
    int rx = root(x);
    int ry = root(y);
    if(rx!=ry) pre[rx] = ry;
}

int main()
{
    for(int i=0; i<101; i++) pre[i] = i;
    int n, m, k;
    cin >> n >> m >> k;
    int a[101][101] = {0};
    int x, y, z;
    for(int i=0; i<m; i++)
    {
        cin >> x >> y >> z;
        if(z==1) un(x, y);//朋友
        else a[x][y] = a[y][x] = 1;//敌人
    }
    for(int i=0; i<k; i++)
    {
        cin >> x >> y;
        if(a[x][y]==1)//敌人
```

```
        {
            if(root(x)!=root(y)) cout << "No way\n";//有共同朋友
            else cout << "OK but...\n";
        }
        else if(root(x)==root(y)) cout << "No problem\n";//是朋友，且没有敌对关系
        else cout <<"OK\n";//不是朋友也没有敌对
    }
    return 0;
}
```

功夫传人：

dfs或bfs都行， 不难，套模板就行

```
#include<bits/stdc++.h>
using namespace std;

vector<int> a[100000];
int n;
double z, r;
double sum = 0;

void dfs(int t, double w)
{
    if(a[t][0]==0) sum += w*a[t][1];
    else
        for(int i=1; i<=a[t][0]; i++)
            dfs(a[t][i], w*r);
}

int main()
{
    cin >> n >> z >> r;
    r = 1-r*0.01;
    for(int i=0; i<n; i++)
    {
        int k, x;
        cin >> k;
        a[i].push_back(k);//a[i][0]:徒弟个数
        cin >> x;
        a[i].push_back(x);//a[i][1]:第一个徒弟编号 或者 放大倍数
        for(int j=0; j<k-1; j++)
        {
            cin >> x;
            a[i].push_back(x);//剩下的徒弟编号
        }
    }
    dfs(0, z);
    cout << int(sum);
    return 0;
}
```

树的遍历：

fun()函数作用：遍历片段a和片段b对应的树。a[al:ar]:后序遍历的片段，b[bl:br]：中序遍历的片段，t:当前子树的根结点对应的层次遍历的顺序号。
遍历过程中，把当前子树的根写到res中，格式是：res[根结点对应的层次遍历的顺序号]=层次遍历的顺序号，然后递归遍历树的左子树 和 右子树。

```
#include<bits/stdc++.h>
using namespace std;

map<int, int> res;

void fun(int a[], int al, int ar, int b[], int bl, int br, int t)
{
    if(ar<al) return;
    res[t] = a[ar];
    int i;
    for(i=bl; b[i]!=a[ar]; i++);
    int cnt1 = i-bl, cnt2 = ar-al-cnt1;//左子树结点个数，右子树结点个数
    fun(a, al, al+cnt1-1, b, bl, i-1, t*2+1);
    fun(a, al+cnt1, ar-1, b, i+1, br, t*2+2);
}

int main()
{
    int n;
    cin >> n;
    int a[30], b[30];
    for(int i=0; i<n; i++) cin >> a[i];
    for(int i=0; i<n; i++) cin >> b[i];
    fun(a, 0, n-1, b, 0, n-1, 1);
    for(auto it=res.begin(); it!=res.end(); it++)
    {
        if(it!=res.begin()) cout << " ";
        cout << it->second;
    }
    return 0;
}
```

愿天下有情人都是失散多年的兄妹：

这个搞清楚题目意思其实不难
结构体数组记录每一个人的性别，父亲编号，母亲编号。数组下标对应该人的编号。
两次深搜把x的五服和y的五服分别保存到bool数组，0/1标记其下标对应的编号是否五服。
比较两个bool数组是否有相同的1即可。

```
#include<bits/stdc++.h>
using namespace std;

struct node
{
    char sex;
    int pa = -1, ma = -1;
};
```

```
node a[100000];

void bfs(int x, bool vi[], int t)
{
    if(t==6 || x==-1) return;
    vi[x] = 1;
    bfs(a[x].pa, vi, t+1);
    bfs(a[x].ma, vi, t+1);
}

void ju(int x, int y)
{
    if(a[x].sex==a[y].sex)
    {
        cout << "Never Mind\n";
        return;
    }
    bool v1[100000] = {0};
    bool v2[100000] = {0};
    bfs(x, v1, 1);//把x的五服祖先都标记在V1里
    bfs(y, v2, 1);//把y的五服祖先都标记在V2里
    int i;
    for(i=0; i<100000; i++)//看看有没有共同祖先
        if(v1[i]==1 && v2[i]==1)
        {
            cout << "No\n";
            break;
        }
    if(i==100000) cout << "Yes\n";
}

int main()
{
    int n;
    cin >> n;
    for(int i=0; i<n; i++)
    {
        int x;
        cin >> x;
        cin >> a[x].sex >> a[x].pa >> a[x].ma;
        if(a[x].pa!=-1) a[a[x].pa].sex = 'M';
        if(a[x].ma!=-1) a[a[x].ma].sex = 'F';
    }
    cin >> n;
    for(int i=0; i<n; i++)
    {
        int x, y;
        cin >> x >> y;
        ju(x, y);
    }
    return 0;
}
```

清点代码库：

mp[输出]=该输出出现的次数， res[次数]=输出，res是multimap，允许key重复
边读入边写入mp。然后再将mp的内容转入到res中
使用multimap的好处是允许key重复，还自动按key排序，key相同则按生成顺序排。爽！
你也可以使用vector来保存结果，但要排序才行。

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    getchar();
    map<vector<int>, int> mp;
    while(n--)
    {
        int k = m;
        vector <int> v;
        while(k--)
        {
            int x;
            cin >> x;
            v.push_back(x);
        }
        mp[v]++;
    }
    cout << mp.size() << endl;
    multimap<int, vector<int>, greater<int>> res;
    for(auto x:mp) res.insert({x.second, x.first});
    for(auto x:res)
    {
        cout << x.first;
        for(auto y:x.second) cout << " " << y;
        cout << endl;
    }
    return 0;
}
```

二叉搜索树的最近公共祖先：

无需建立树，利用递归直接在先序序列里查找最近公共祖先
先检查当前子树的根，如果根的键值是两节点之一，那么根就是另一个的祖先
再检查两节点是否分别位于当前子树的左右子树，是则说明根是它们的共同祖先
再检查两者同在左子树还是右子树，递归继续查找
在查找左右子树的分界点时（注意利用二叉搜索树的原理），若自左向右顺序找，测试点4超时。改为自右向左就可以。但始终不是完美的办法。我分析这里应该使用二分。程序其他地方实在想不到还能怎样优化。

```
#include<bits/stdc++.h>
using namespace std;
```

```cpp
int a[10001];//存储n和键值
map<int, int> pos;//pos[i]=j, 表示键值i在a数组的下标是j

void ju(int l, int r, int x, int y)
{//判断x和y是不是分别位于以a[l]为根的左子树和右子树中
//是的话，则a[l]是x与y的最近公共祖先。否则递归往下找。
    if(a[l]==x)//x是y的祖先
    {
        printf("%d is an ancestor of %d.\n", x, y);
        return;
    }
    if(a[l]==y)//y是x的祖先
    {
        printf("%d is an ancestor of %d.\n", y, x);
        return;
    }
    if((x<a[l] && y>=a[l]) || (y<a[l] && x>=a[l]))//分别位于以a[l]为根的左右子树
    {
        printf("LCA of %d and %d is %d.\n", x, y, a[l]);
        return;
    }
    int i;
    for(i=r; i>l && a[i]>a[l]; i--);//找左右子树的分界点
    //如果上面是从左往右找，测试点4会超时
    //从右往左找只是应付了这种测试数据，真正高效应该使用二分搜索。
    if(x<a[l] && y<a[l])//都在左子树
        ju(l+1, i, x, y);
    else // 都在右子树
        ju(i+1, r, x, y);
}

int main()
{
    int m, n;
    cin >> m >> n;
    set<int> st;
    for(int i=1; i<=n; i++)
    {
        scanf("%d", &a[i]);
        pos[a[i]] = i;
    }
    while(m--)
    {
        int x, y, f1=0, f2=0;
        scanf("%d%d", &x, &y);
        if(pos.find(x)==pos.end()) f1=1;
        if(pos.find(y)==pos.end()) f2=1;
        if(f1+f2==2) printf("ERROR: %d and %d are not found.\n", x, y);
        else if(f1==1) printf("ERROR: %d is not found.\n", x);
        else if(f2==1) printf("ERROR: %d is not found.\n", y);
        else ju(1, n, x, y);
    }
    return 0;
}
```