无向图的连通性问题

在无向图中，判断两点是否连通可以用搜索或并查集完成。在很多无向图问题中，需要找出一些关键的点或边，删除它可能导致图的不连通。对于有向图来说有一些点可以相互到达，这些相互到达的点在解决一些有序问题中可能不好处理，需要将它们合并成为一个点才可以进行有序操作。这些问题都将在本章中找到答案。

一、　无向图连通性问题常见概念

连通图：无向图中，任意取两个顶点都是互相连通的，那么则称此无向图是连通的。

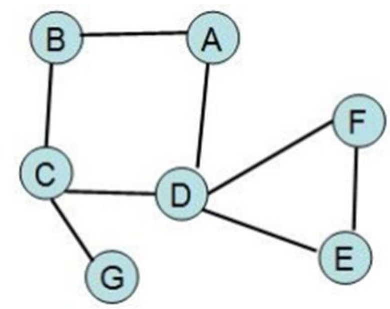
割点：无向连通图中，去掉一个顶点及和它相邻的所有边，图中的连通分量数增加，则该顶点称为割点。

割边：无向连通图中，去掉一条边，图中的连通分量数增加，则这条边，称为桥或者割边。

连通分量：无向图中相互可以到达的点集称为一个连通分量。

边双连通分量：一张无向连通图不存在割边，则称为“边双连通图”，其中极大边双连通子图被称为边双连通分量（e-DCC）。

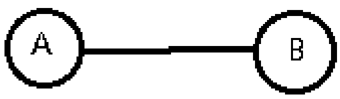
一张图中边双连通分量之间不相交，同时边双连通分量中任意两点之间，存在至少两条不相交的路径。如下图：



割边为 C-G，其中边双连通分量为 2 个：{G},{A,B,C,D,E,F}

点双连通分量：一张无向连通图不存在割点，则称为“点双连通图”，其中极大点双连通子图被称为点双连通分量（v-DCC）。点双连通分量的意义是，一个连通点双连通分量中，任意两点之间存在至少两条点不相交的路径。上图中，割点为 C,D，点双连通分量为：{A,B,C,D},{D,E,F},{C,G}.

特殊的，对于两个点相连的图，A,B 属于一个点双连通，属于两个边双连通。如下图：



有如下定理：

一个无向连通图是点双连通图，当且仅当满足下面 2 个条件之一：

1、图的顶点数量不超过 2.

2、图中任意两点都同时包含至少一个简单环中。

一个无向连通图是边双连通图，当且仅当任意一条边都包含在至少一个简单环中。

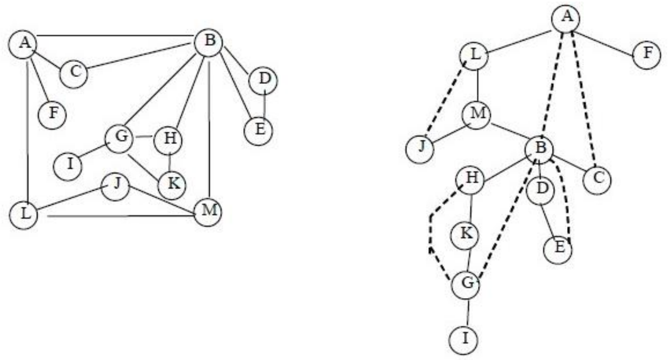
二、　Tarjan 算法

计算割点割桥，以及求双连通分量都可以通过 tarjan 算法来实现。Tarjan 算法是有科学家 Robert Tarjan 发明在线性时间内计算出割点割桥以及双连通分量的算法。同时在处理有向图的连通性问题上也可以求出强连通分量，必经点、边。

Tarjan 算法是基于深度优先搜索，在介绍算法前首先了解以下几个概念：

时间戳：在图的 DFS 遍历中，每个节点第一次访问的时间顺序按 1~N 编号，该编号被称为该点的时间戳，记为 dfn[x]。

DFS 搜索树：在无向连通图中任选一个点出发 DFS 遍历，发生递归的边(u,v)构成一棵树，称为搜索树，如：



(a) 一个连通图无向图 G7　　(b) G7 的深度优先生成树

图（3）

树边： 称为父子边，DFS 搜索树上的边，可理解为在 DFS 过程中访问未访问节点时所经过的边。在搜索树中的实线所示。

回边：又称返祖边。在搜索树中的虚线所示，可理解为在 DFS 过程中遇到已访问节点时所经过的边。

如上图中，图(b)是图(a)的 dfs 搜索树，其中实线为**树边**，虚线为**返祖边**

追溯值：Tranjan 算法除时间戳外，再引入了追溯值的概念，记为 low[x]。low[x]表示节点 x 不通过搜索树上的父亲能追溯的最小时间。如上图中点 K,G 能访问到的最早时间为节点 B 的时间戳。

计算 low 值：

根据定义,为了计算 low[x],先令 low[x]=dfn[x],然后考虑从 x 出发的每一条边(x,y)：

1、若边(x,y)为树边（即 x 是 y 的父亲）,则令 low[x]=min{low[x],low[y]}

2、若边(x,y)为返祖边，则令 low[x]=min{low[x],dfn[y]}

$$low[u]=\begin{cases}\min\{low[u],\ low[v]\} & (u,v)\text{为树边}\\ \min\{low[u],\ dfn[v]\} & (u,v)\text{为回边且}v\text{不为}u\text{的父亲节点}\end{cases}$$

下图中标注了图（3）的 dfn，low 值。

i	0	1	2	3	4	5	6	7	8	9	10	11	12
vertex	A	B	C	D	E	F	G	H	I	J	K	L	M
dfn[i]	1	5	12	10	11	13	8	6	9	4	7	2	3
low[i]	1	1	1	5	5	1	5	5	8	2	5	1	1

三、　割点判断法则

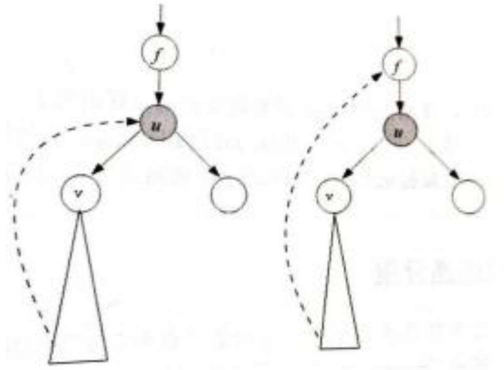
若 x 不是搜索树的树根，则 x 是割点当且仅当在搜索树中存在 x 的一个子节点 y ，满足：

dfn[x]≤low[y]

特别的，对于 x 是搜索树的根节点，则 x 是割点当且仅当搜索树上存在至少两个子节点 y1,y2 满足上述条件。

证明：

如图所示，考虑 u 的任意子结点 v。如果 v 及其后代不能连回 f，则删除 u 之后 f 和 v 不再连通；反过来，如果 v 或它的某一个后代存在一条返祖边连回 f，则删除 u 后，以 v 为根的整棵子树中的所有结点都可以利用这条返祖边与 f 连通。
如果所有子树中的结点都和 f 连通，根据“连通”关系的传递性，整个图就是连通的



注意：对于每个割点来说，上述条件可能不止成立一次，所以一般不要边判断边输出，而用数组来记录每个结点是不是割点，最后一次性输出。

四、　　割边判断法则

对于边(x,y)是割边，当且仅当搜索树上存在 x 的一个子节点 y，满足：

　　　　　　dfn[x]<low[y]

证明方法和割点的证明类似，不在赘述。

五、　　实例解析

例一、 P10100 网络

给你一个无向图，有 n 个节点，编号从 1 到 n，若干条边，现在要你求出求其中的割点个数。 1<=n<100

【解析】模板题。

【参考代码】

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
#define N 10001
vector<int>G[N];
int dfn[N],low[N];
int tcnt,n;
int flag[N]; //标记 i 是否是 割点
int root;
void dfs(int u,int fa){        //fa 是 u 的 father
    int son=0;                 //儿子个数
    dfn[u] = low[u]=++tcnt;
    for(int i=0;i<G[u].size();i++){
        int v = G[u][i];
        if(dfn[v] == 0){ //(u,v)是树边
            son ++ ;
            dfs(v , u);
```

```
            low[u] = min(low[u] , low[v]);//更新当前点能回到最早的时间戳
            if(  low[v] >= dfn[u]){//割点标记
                flag[u] = 1;
            }

        }
        else if(v!=fa){//非树边
             low[u] = min(low[u] , dfn[v]);
        }
    }
    if(u==root && son==1)      //如果 u 是根，且有 2 个 s 儿子，就是割点
        flag[u] =0;
}

int main(){
    while(cin>>n&&n!=0){
        int a,b;
        while(cin>>a&& a!=0){
            while(getchar()!='\r'){
                cin>> b;
                G[a].push_back(b);
                G[b].push_back(a);
            }
        }
        tcnt = 0;
        int ans=0;//割点数
        root =1;
        dfs(root, -1) ;
        for(int i=1;i<=n;i++)
            if(flag[i]) ans++;
        cout<<ans<<endl;
        for(int i=1;i<=n;i++)G[i].clear();
    }
    return 0;
}
```

例二、 H7692 「ZJOI2004」嗅探器

在无向图中寻找出所有的满足下面条件的点：割掉这个点之后，能够使得一开始给定的两个点 a 和 b 不连通，割掉的点不能是 a 或者 b。

【解析】

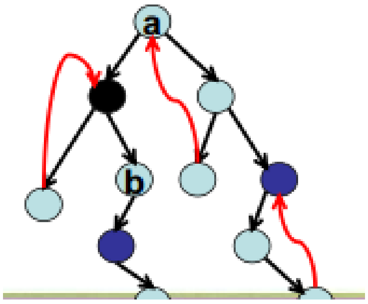
朴素算法：□ 枚举每个点，删除它，然后判断 a 和 b 是否连通，时间复杂度 O(NM)。如果数据范围扩大，该算法就失败了

题目要求的点一定是图中的割点，但是图中的割点不一定题目要求的点。如上图中的蓝色点，它虽然是图中的割点，但是割掉它之后却不能使 a 和 b 不连通。

由于 a 点肯定不是我们所求的点，所以可以以 a 为根开始 DFS 遍历整张图。

□ 对于生成的 DFS 树，如果点 u 是割点，并且以他为根的子树中存在点 b，那么该点就是问题所求的点。换句话说：割点一定在从 b 沿父子边走到 a 的路径上。

□ 时间复杂度是 O(M)的，如图，黑色的点表示问题的答案，蓝色的点虽然是图的割点，但却不是问题要求的答案。



【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define N 200
int
n,a,b,tot=0,low[N],dfn[N],ans=110,f[N],nex[N*200],fis[N],x,y,cnt=
0;
void read(int &x) {
    x=0;int f=1;
    char c=getchar();while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    if(c=='-') f=-1;else x=c-'0';
    while(c=getchar(),c>='0'&&c<='9')x=x*10+c-'0';
    x*=f;return;
}
void write(int x) {
    if(x<0) {
        putchar('-');write(-x);return;
    }
    if(x>9) write(x/10);
    putchar(x%10+'0');
    return;
}
struct node{
    int u,v;
}e[N*200];
void add(int u,int v){
    cnt++; e[cnt].u=u; e[cnt].v=v;
    nex[cnt]=fis[u]; fis[u]=cnt;
}
void dfs(int u,int fa){
    ++tot; low[u]=tot; dfn[u]=tot; f[u]=fa;
    for(int i=fis[u];i;i=nex[i]){
```

```
        int v=e[i].v;
        if(v==fa) continue;
        if(dfn[v]==0){
            dfs(v,u); low[u]=min(low[u],low[v]);
        }
        else low[u]=min(low[u],dfn[v]);
    }
}
void check(int u){
    if(u==a) return;//为根节点，返回
    if(low[u]>=dfn[ f[u] ]&&f[u]!=a&&f[u]<ans)
       ans=f[u];
    check(f[u]);
}
int main(){
    read(n);
    while(read(x),read(y),x!=0,y!=0){
        add(x,y); add(y,x);
    }
    read(a); read(b);
    dfs(a,0); check(b);
    if(ans<110){
        write(ans); printf("\n");
    }
    else printf("No solution");
    return 0;
}
```

例三、 P3051. 关键网线

无向连通图中，某些点具有 A 属性，某些点具有 B 属性。请问哪些边割掉之后能够使得某个连通区域内没有 A 属性的点或者没有 B 属性的点。

【解析】

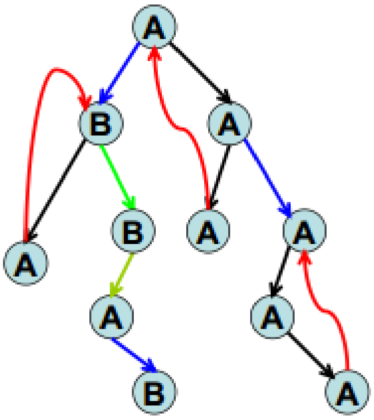
朴素算法：

 枚举每条边，删除它，然后判断是否有独立出来的连通区域内没有 A 属性或者没有 B 属性。复杂度 O(M^2)

正解：正如嗅探器一样，题目要求的边一定是原图中的割边，但是原图中的割边却不一定是题目中要求的边。

设 A 种属性总共有 SUMA 个，B 种属性总共有 SUMB 个。和嗅探器类似的，如果边 e=u→v 是割边，且以 v 为根的子树中，A 种属性的数目为 0 或者为 SUMA，或者 B 种属性的数目为 0 或者为 SUMB，那么 e 就是题目要求的边。

下图中，蓝色的边表示题目要求的边，绿色的边表示虽然是图中的割边，但不是题目要求的边




```
#include<bits/stdc++.h>
using namespace std;
struct bian{int v,nxt;
}a[2100001];
int ans,cnt,n,m,k,l,c,b,num;
int
nd[1100010],low[1100010],dft[1100010],cona[1100010],conb[1100010]
,fa[1100100];
void add(int x,int y){
    a[++cnt].v=y;a[cnt].nxt=nd[x];nd[x]=cnt;
}
inline int read(){
    int data=0;int w=1; char ch=0;
    ch=getchar();
    while(ch!='-' && (ch<'0' || ch>'9')) ch=getchar();
    if(ch=='-') w=-1,ch=getchar();
    while(ch>='0' && ch<='9')
      data=(data<<3)+(data<<1)+ch-'0',ch=getchar();
    return data*w;
}
void dfs(int x){
    dft[x]=low[x]=++num;
    int v;
    for(int i=nd[x];i;i=a[i].nxt){
        v=a[i].v;
        if(fa[x]!=v){
            if(dft[v]){
                low[x]=min(low[x],dft[v]);
            }
            else{
                fa[v]=x;
                dfs(v);
                low[x]=min(low[x],low[v]);
```

```
                cona[x]+=cona[v];conb[x]+=conb[v];

    if(low[v]>dft[x]&&(!cona[v]||!conb[v]||cona[v]==k||conb[v]==l)
)
                    ans++;
            }
        }
    }
}
int main(){
    n=read();
    m=read();
    k=read();
    l=read();
    for(int i=1;i<=k;i++){
        int h;h=read();cona[h]++;
    }
    for(int i=1;i<=l;i++){
        int h;h=read();conb[h]++;
    }
    for(int i=1;i<=m;i++){
        int h,o;
        h=read();o=read();
        add(h,o);add(o,h);
    }
    dfs(1);
    printf("%d",ans);
    return 0;
}
```

六、　　双连通分量计算

1、点双连通分量 v-DCC

为求点双连通分量，在 Tarjan 算法过程中利用一个栈来维护，按以下方法维护栈中元素：

1. 节点第一次访问，则节点入栈。
2. 当满足割点判断条件 dfn[x]≤low[y]时，无论 x 是否为根，则：

   从栈顶不停弹出节点，直到 y 被弹出；

   刚才弹出的所有节点与节点 x 一起构成一个 v-DCC

注意，不能将 x 弹出栈，x 可能是其他 v-DCC 的点。所以也可以将边压人栈中，这样可以避免将 x 不小心弹出栈。

2、边双连通分量 v-DCC

边双连通分量计算相对比较容易，只需要求出所有的割边，删除割边后，无向图自然分成若干连通块，每一个连通块就是一个 v-DCC.

具体实现时，还是借助一个栈，每一个点第一次访问时压入栈中，当一条边为割边时，

则将栈内的元素弹出至当前节点 x。

七、 双连通实例

例一、 P3072. 修路

【题意】

有一个公园有 n 个景点，公园的管理员准备修建 m 条道路，并且安排一些形成回路的参观路线。

如果一条道路被多条道路公用，那么这条路是冲突的；如果一条道路没在任何一个回路内，那么这条路是不冲突的。

问分别有多少条有冲突的路和没有冲突的路 ？

【问题原型】

有一个无向图，n 个节点 M 条边。如果至少有 2 个环共用一些边，那么这些边是“冲突边”

如果一条边不在任何一个回路中，这些边为“多余边”。

找出冲突边，多余边条数并输出。图不一定连通。

【分析】

1．“多余边”不在任何一个环中，那么多余边一定是桥，所以统计这个无向图中有多少桥即可

2． “冲突边”有多少，比较麻烦。

　　如果一个环比较特殊，n 个点刚好 n 条边，例如（1,2）（2,3）（1，3）这种环。

　　　　这个环内，一条“冲突边”都没有，

但是如果一个环内的边数大于点数，　　那么这个环内所有边都是“冲突边”

【做法】

求出这个无向图有多少个点双连通分量，对于每个点双连通分量，如果内部的边数>点数，那么这些边全部都是冲突边

【参考代码】

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

const int M = 200010;
const int N = 10010;

struct Edge {
   int from, to;
   int next;
} edge[M];
int head[N];
int cnt_edge;
void add_edge(int u, int v)
{
   edge[cnt_edge].from = u;
   edge[cnt_edge].to = v;
   edge[cnt_edge].next = head[u];
   head[u] = cnt_edge++;
}
```

```
int dfn[N]; int idx;
int low[N];
stack<Edge> stk;
set<int> bcc;
int cut;    // 桥的数量
int ans;    // 冲突边数量
int m, n;

void dfs(int u, int pre)
{
    dfn[u] = low[u] = ++idx;
    for (int i = head[u]; i != -1; i = edge[i].next)
    {
        int v = edge[i].to;
        if (v == pre) continue;
        if (!dfn[v])
        {
            stk.push(edge[i]);
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u])      // 割点
            {
                Edge tmp;
                int cnt = 0;
                bcc.clear();
                do {
                    cnt++;
                    tmp = stk.top();
                    stk.pop();
                    bcc.insert(tmp.from);
                    bcc.insert(tmp.to);
                } while (tmp.from != u || tmp.to != v);
                if (cnt > bcc.size()) ans += cnt;
            }
            if (low[v] > dfn[u]) ++cut;
        }
        else if (dfn[v] < dfn[u])
        {
            stk.push(edge[i]);
            low[u] = min(low[u], dfn[v]);
        }
    }
}
```

```
void init()
{
    memset(head, -1, sizeof head);
    memset(dfn, 0, sizeof dfn);
    ans = cut = cnt_edge = idx = 0;
}

int main()
{
    while (~scanf("%d%d", &n, &m))
    {
        if (n == 0 && m == 0) break;
        int u, v;
        init();
        for (int i = 0; i < m; ++i)
        {
            scanf("%d%d", &u, &v);
            add_edge(u, v);
            add_edge(v, u);
        }
        for (int i = 1; i <= n; ++i)
        if (!dfn[i]) dfs(i, -1);
        printf("%d %d\n", cut, ans);
    }
    return 0;
}
```

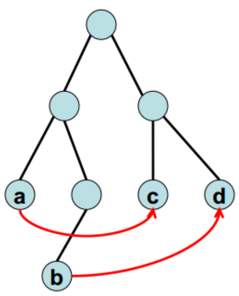
例二、 P3074

题意：

在一个连通的无向图中，问你最少需要添加多少条边可以使得图边双连通？

分析：

如图所示，点代表了原图中的一个块，它们之间的连边是割边。连接 a 与 c，b 与 d 之后，图中就没有割边了

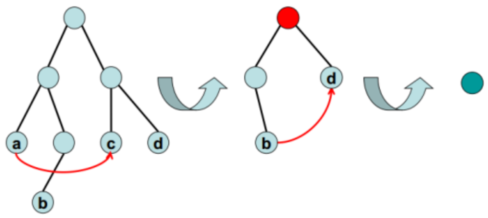


但并不是任意连接两个叶子结点就可以达到目标。

假如连接了 a 与 b，c 与 d，原图并没有变成一个块

进一步分析刚才的算法，每次连接两个叶子结点之后，把新生成的圈压缩成为一个点，以前和圈上的点关联的点，都和新生成的这个“压缩点”

相关联。于是原来的树在添加一条边之后，又变回了一棵树
在连接 a 与 c 之后，新生成的树只剩下 2 个叶子结点；连接 b 与 d 之后，树就被压缩成了一个点。



而如果先连接 a 与 b，那么新生成的树会剩下 3 个叶子结点，连接 c 与 d 之后，树中还剩 2 个叶子结点，所以这种连接方法还需要多连一条边。
现在的问题是，是否一定能找出这样子的两个叶子结点，使得压缩成的点不会成为新的叶子节点呢？
连接的两个点的那条树中的唯一路径上，如果除了它们的最近公共祖先到自己的父亲有连边以外，其他的结点没有别的分叉，那么连接这两个点之后缩圈得到的点将会是一个叶子结点
假设图中的任意两个叶子连接之后，都会多产生一个叶子结点。
当图中的叶子结点是 2 个或者 3 个的时候，怎么连都没有区别
当图中的叶子结点有 4 个的时候，a 和 b 到它们的最近公共祖先都没有别的分叉，且 c 和 d 到它们的最近公共祖先没有别的分叉，可以知道，a 和 c 到它们的最近公共祖先上一定有分叉。
这个与假设矛盾。所以我们总能找到两个叶子结点，使得它们连边之后缩成的树不会新产生叶子结点
算法流程：
1.读入数据建立无向图；
2.利用 Tarjan 算法求出所有连通块，并缩块；
3.统计每个结点的度，然后求出度为 1 的结点个数 Ans；
4.答案=(Ans+1)/2
【参考代码】

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 50010;
inline char nc(){
    static char buf[100000],*p1=buf,*p2=buf;
    return
p1==p2&&(p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?EOF:*p1++;
}
#define nc getchar
template<typename T>
inline void read(T &x){
    x = 0;T f = 1;
    char c = nc();
    while(c<'0'||c>'9'){if(c=='-')f = -1;c = nc();}
    while(c>='0'&&c<='9'){x = x*10+(c^48);c = nc();}
```

```
    x*=f;
}
struct edge{
    edge(int f,int t):f(t),t(t){}
    edge(){}
    int f,t,next;
}edges[MAXN<<1];
int head[MAXN],top;
void add(int f,int t){
    edges[++top].next = head[f];
    edges[top].t = t;
    head[f] = top;
}
int dfn[MAXN],low[MAXN],dfn_cnt;
int idx[MAXN],id;
stack<int> stk;
void tarjan(int x,int fa){
    dfn[x] = low[x] = ++dfn_cnt;
    stk.push(x);
    for(int i = head[x];i;i = edges[i].next){
        int t = edges[i].t;
        if(t==fa)continue;
        if(!dfn[t]){
            tarjan(t,x);
            low[x] = min(low[x],low[t]);
        }else if(idx[t]==-1)low[x] = min(dfn[t],low[x]);
    }
    if(dfn[x]==low[x]){
        id++;
        while(stk.top()!=x){
            idx[stk.top()] = id;
            stk.pop();
        }
        idx[x] = id;
        stk.pop();
    }
}
int n,m;
int deg[MAXN];
int main(){
    read(n),read(m);
    memset(idx,-1,sizeof(idx));
    for(int i = 1;i<=m;i++){
        int f,t;
```

```
        read(f),read(t);
        add(f,t);add(t,f);
    }
    for(int i = 1;i<=n;i++)if(idx[i]==-1)tarjan(i,i);
    for(int i = 1;i<=n;i++){
        for(int j = head[i];j;j = edges[j].next){
            int t = edges[j].t;
            if(idx[i]!=idx[t])deg[idx[i]]++,deg[idx[t]]++;
        }
    }
    int ans = 0;
    for(int i = 1;i<=id;i++){
        if(deg[i]==2)ans++;
    }
    cout<<(ans+1)/2;
    return 0;
}
```