交互题

通俗来讲，交互题与平时题目的输入输出反过来，是让你设计程序去向用户提出询问，由用户给出答案，并且在这基础上由程序推断出正确答案的一种形式。在下面第一道例题中会认识到交互题的基本写法。

应用cf上的话说明怎样调整输入输出：

```
//This is an interactive problem. Remember to flush your output while communicating with the testing program.** You may use fflush(stdout) in C++, system.out.flush() in Java, stdout.flush() in Python or flush(output) in Pascal to flush the output.
```

注意事项

- 每一次输出后都要刷新缓冲区，否则会引起$idleness\ limit\ exceeded$错误，在codeforces中是$Wrong\ Answer$错误。另外，如果题目含多组数据并且程序可以在未读入所有数据时就知道答案，也仍然要读入所有的数据，否则同样会因为读入混乱引起ILE（可以一次提出多个询问，一次接收所有询问的答案）。同时尽量不要使用快读。
- 交互题一般会限制查询次数或者查询区间，所以要尽量选择最优的查询方法来得出结果，尽可能做到每次查询都能得到完全不一样的信息，以免浪费机会。

例题

CF1167B Lost Numbers

序列a是4,8,15,16,23,42组成的一个排列。给定4次询问，每次回答出两个位置的数之积，求序列a。

```
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
using namespace std;
const int N=2e5+7;
const int mod=1e9+7;

int a[]={0,4,8,15,16,23,42},d[5];
```

```
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}

signed main(){
    for(int i=1;i<=4;i++){
        cout<<"? "<<i<<" "<<i+1<<"\n";
        fflush(stdout);
        cin>>d[i];
    }
    do{
        if(a[1]*a[2]==d[1]&&a[2]*a[3]==d[2]&&a[3]*a[4]==d[3]&&a[4]*a[5]==d[4]){
            cout<<"! ";
            for(int i=1;i<=6;i++) cout<<a[i]<<" ";
            return 0;
        }
    }while(next_permutation(a+1,a+7));
    return 0;
}
```

CF843B Interactive LowerBound

给最多2000个询问和50000个递增链表上的结点，给出链头st和x，需要你找到第一个大于等于x的结点。

做法：先用1000次尝试随机地去找比x大且最近的数，然后再用1000次尝试向前找，就能极大概率求出正解。

```
#include<bits/stdc++.h>
using namespace std;
const int N=50100;
int n,st,x,gl;
int val[N],nex[N];
int main(){
    srand(time(0)^getpid());
    cin>>n>>st>>x;
    gl=st;
    for(int i=1;i<=1000;i++){
        int q=1ll*rand()*rand()%n+1;
        cout<<"? "<<q<<"\n";
        fflush(stdout);
        cin>>val[q]>>nex[q];
        if(val[gl]<val[q]&&val[q]<=x){
            gl=q;
        }
    }
    for(int i=gl;i!=-1;i=nex[i]){
        cout<<"? "<<i<<"\n";
        fflush(stdout);
        cin>>val[i]>>nex[i];
        if(val[i]>=x){
            cout<<"! "<<val[i]<<"\n";
            fflush(stdout);
            return 0;
```

```
        }
    }
    cout<<"! -1\n";
    fflush(stdout);
}
```

CF750F New Year and Finding Roots

给定深度为$h(h \leq 7)$的满二叉树，每次询问可以知道某个节点连向哪几个结点，要求在16次询问内找到根节点。

做法：分析满二叉树的性质：①度为1是叶子节点；②度为2是根节点，直接输出答案；③度为3是普通结点，需要向三个方向bfs访问。

其实只需要讨论第三种情况即可，随机度为3的点st出发，在深度为7的情况下bfs最多需要询问17次找到根节点（可以画图证明），但是我们bfs了10个普通节点和6个叶子结点后（知道其中一个儿子是叶子，那么另外一个就不用搜了），剩下的一个结点就是根节点啦。

```
#include<bits/stdc++.h>
#define LL long long
#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))
using namespace std;

const LL maxn=1<<8;
LL dep[maxn],vis[maxn],c1[maxn],c2[maxn],c3[maxn],c[maxn],s[maxn],up[maxn];
LL h;

void Try(LL x,LL from){ //询问
    if (vis[x]) return;
    cout<<"? "<<x<<endl;
    fflush(stdout);
    cin>>s[x];
    vis[x]=1;
    c[x]=from;
    cin>>c1[x];
    if (s[x]>1) cin>>c2[x];
    if (s[x]>2) cin>>c3[x];
}

LL next(LL x,LL from){
    Try(x,from);
    if (c1[x]!=c[x]) return c1[x];
    else if (c2[x]!=c[x]) return c2[x];
    else return c3[x];
}

LL next2(LL x,LL from){
    Try(x,from);
    if (c3[x]!=c[x]) return c3[x];
    else if (c2[x]!=c[x]) return c2[x];
    else return c1[x];
}

LL solve(LL x){ //bfs
    if (s[x]==2) return x; //碰到度为2的点直接输出答案
```

```
    if (!dep[x]){
        LL u1=x,u2=x,u3=x;
        LL n1=c1[x],n2=c2[x],n3=c3[x];
        dep[x]=h;
        while(s[u1]!=1&&s[u2]!=1&&s[u3]!=1){
            LL l1=u1,l2=u2,l3=u3;
            u1=n1,u2=n2,u3=n3;
            n1=next(u1,l1),n2=next(u2,l2),n3=next(u3,l3);
            if (s[u1]==2) return u1;
            if (s[u2]==2) return u2;
            if (s[u3]==2) return u3;
            dep[x]--;
        }
        if (s[x]==1) up[x]=c1[x],Try(up[x],x);
        else{
            if (s[u1]!=1) up[x]=c1[x];
            else if (s[u2]!=1) up[x]=c2[x];
            else up[x]=c3[x];
        }
        dep[up[x]]=dep[x]-1;
        return solve(up[x]);
    }
    if(dep[x]==2){
        LL u=next(x,c[x]);
        LL v=next2(x,c[x]);
        Try(u,x);
        if (s[u]==2) return u;
        else return v;
    }
    else if(dep[x]==3){
        LL u=next(x,c[x]); Try(u,x);
        LL v=next2(x,c[x]); Try(v,x);
        LL u1=next(u,c[u]),u2=next2(u,c[u]);
        LL v1=next(v,c[v]),v2=next2(v,c[v]);
        Try(u1,u),Try(u2,u),Try(v1,v);
        if (s[u1]==2) return u1;
        else if (s[u2]==2) return u2;
        else if (s[v1]==2) return v1;
        else return v2;
    }else{
        LL u=x,lu=c[x],nu;
        for (LL i=1;i<=h-dep[x];i++){
            nu=next(u,lu);
            if (s[u]==2) return u;
            lu=u,u=nu;
        }
        Try(u,lu);
        if (s[u]==2) return u;
        else {
            LL nx;
            if (s[u]==3) nx=next(x,c[x]);
            else nx=next2(x,c[x]);
            Try(nx,x);
            dep[nx]=dep[x]-1;
            return solve(nx);
        }
    }
}
```

```
int main(){
    LL T;
    cin>>T;
    while (T--){
        memset(dep,0,sizeof(dep));
        memset(vis,0,sizeof(vis));
        memset(c,0,sizeof(c));
        memset(c1,0,sizeof(c1));
        memset(c2,0,sizeof(c2));
        memset(c3,0,sizeof(c3));
        memset(s,0,sizeof(s));
        memset(up,0,sizeof(up));
        cin>>h;
        if (h<=4){  //深度小于5可以一个一个问
            for (LL i=1;i<=(1<<h)-1;i++{
                cout<<"? "<<i<<endl;
                fflush(stdout);
                LL n;
                cin>>n;
                LL x;
                for (LL j=1;j<=n;j++) cin>>x;
                if (n==2) {
                    cout<<"! "<<i<<endl;
                    break;
                }
            }
        }else{ //从1号点出发搜索
            LL from=1;
            Try(from,0);
            cout<<"!"<<" "<<solve(from)<<endl;
        }
    }
    return 0;
}
```

参考资料

https://oi-wiki.org/contest/interaction/