number

可以注意到，每次操作要么让数位和减少 9 ，要么让总位数减少 1。

而终止的条件是总位数为 1 且数位和 ≤ 9。所以，如果假设一开始的数位和为 X ，总位数为 n ，则答案一定就是 $\lfloor \frac{X-1}{9} \rfloor + (n-1)$ ，所以只需要一开始求出数位和即可，复杂度 $O(n)$ 。

考察知识点

思维

gossip

明显关系可以采用并查集进行维护。每次八卦的传递可以直接当作整个集合所有元素全部加上一个数值。

但由于后面交到的朋友不会得知前面传递的八卦，这一点的处理要怎么做呢？

我们把每次并查集的合并稍微转换一下，假设这一次 x 和 y 成为了朋友，那么先获得两个集合的根结点，记作 x' 和 y' 。那么每次我们只需要创建一个新的根结点，并将 x' 和 y' 当作这个结点的两个儿子结点即可。这样整个并查集实际上就相当于好多棵二叉树，每次合并都会把两棵树合并起来。（当然不创建新的结点也可做，就是下一步记得把值差分掉。）

每次把八卦信息传递给整个集合的操作，就记在此时根结点的位置上即可，相当于每个结点上的值也会对以它作为根结点时的整棵子树内的每个点产生影响。之后只需要按照树上前缀和的方式，将每个结点上的值下传即可。

时间复杂度 $O(m \log\ n)$ 。

考察知识点

并查集 前缀和

burst

解法1

考虑一个最短的简单环一定是与 1 直接相连的两个点间的最短路径长度加上这两个点到 1 的距离，那么我们考虑最短路，每次随机将 1 的出点分组，每次将距离最短的两个点分到不同组内的概率为 50%，每次从一组向另一组跑最短路，跑 k 次后正确率便达到了 $\frac{2^k-1}{2^k}$，多跑几次即可。

也可以按照二进制分组。

解法2

以 1 为起点，跑一边最短路，建出最短路树。
然后枚举非树边，如果这条非树边和树边形成的环包含 1 号节点，则用这个环更新答案。

考察知识点

思维 最短路

interval

15%

根据题意，区间的合法性主要有两个条件。因此我们对应维护两个数组：

> - f_i：表示 i 右边第一个大于等于 A_i 的 A 的位置。
> - g_i：表示 i 左边连续小于 B_i 的最远位置。
>
> 其中：
>
> f 数组可以单调栈处理；
>
> g 数组可以 st 表加二分处理。

对于每一个询问，暴力枚举区间判断，通过 f 数组和 g 数组直接判断合法性。

时间复杂度：$O(Q*N^2)$

35%

预处理所有区间的合法性，然后将询问区间对应到二维平面上。

一个区间如果合法，那么它对于二维平面的某个点有贡献。

一个询问，相当于对于平面的一个矩阵查询。

于是做一个二维前缀和即可。

60%

发现对于每一个 r 合法的 l 最多只有 1 个。所以所有的合法区间最多只有 n 个。于是我们将所有合法区间 $[l,r]$ 处理出来。

然后对于一个询问 $[L,R]$ 就是询问有多少个合法区间在这个区间内。经典的二维偏序问题。

二维偏序问题模板：

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
//给n个求合法区间[l,r]
//给m个询问[L,R]，询问有多少个合法区间被包含。
int n,m,mx;
int vl[N],vr[N];
int ql[N],qr[N];
vector<int >V[N];
vector<int >Q[N];
int ans[N];
void add(int x){//此处省略树状数组实现
}
int ask(int l,int r){
}
```

```
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;++i){
        cin>>vl[i]>>vr[i];  mx=max(mx,vr[i]);
        V[vr[i]].push_back(i);
    }
    for(int i=1;i<=m;++i){
        cin>>ql[i]>>qr[i];  mx=max(mx,qr[i]);
        Q[qr[i]].push_back(i);
    }
    for(int i=1;i<=n;++i){
        for(auto v:V[i])    add(vl[v],1);
        for(auto v:Q[i])    ans[v]=ask(ql[v],mx);
    }
    for(int i=1;i<=n;++i)   cout<<ans[i]<<endl;
    return 0;
}
```

100%

由上一个点的启发，我们对于每个右端点 r 考虑以它为右端点的合法区间的左端点 l 有哪些。

> 对于第一个条件，我们维护到目前的一个单调栈。合法的 l 必须在单调栈中。
>
> 对于第二个条件，合法的 l 是存在一个区间内的，即 $l \in [g_r, r-1]$ 。
>
> 所以对于当前的右端点，合法的左端点是这两个条件的交集。

所以我们使用线段树维护两个数组：

> a : a_i 表示 i 是否在单调栈中。
>
> s ： s_i 表示 i 到目前为止是多少个合法区间的左端点。

我们需要实现的操作

> - 由于某个点在单调栈中的状态切换，我们会对于 a 单点取反。
> - 然后对于一个右端点 r ，我们把以它为右端点的合法区间的左端点在 s 数组中加一。相当于将区间 $[g_r, r-1]$ 中 a 的值往 s 中加一次。
> - 然后询问的时候我们会进行一个 s 的区间和查询。

```
#include<bits/stdc++.h>
using namespace std;
#define ls u<<1
#define rs u<<1|1
#define ll long long
#define ii pair<int,int>
#define vv vector<vector<int > >
```

```
#define fi first
#define se second
#define endl '\n'
#define debug(x) cout << #x << ": " << x << endl
#define pub push_back
#define pob pop_back
#define puf push_front
#define pof pop_front
#define lb lower_bound
#define ub upper_bound
#define i128 __int128
#define Time 1.0*clock()/CLOCKS_PER_SEC
#define pcnt(x) __builtin_popcount(x)
#define mem(a,goal) memset(a,(goal),sizeof(a))
#define rep(x,start,end) for(int x=(start)-((start)>(end));x!=(end)-((start)>(end));
((start)<(end)?x++:x--))
#define all(x) (x).begin(),(x).end()
#define sz(x) (int)(x).size()
ll rd()
{
    ll a=0;int f=0;char p=getchar();
    while(!isdigit(p)){f|=p=='-';p=getchar();}
    while(isdigit(p)){a=(a<<3)+(a<<1)+(p^48);p=getchar();}
    return f?-a:a;
}
const int INF=998244353;
const int P=998244353;
const int N=1e6+5;
int n,m;
int a[N],b[N];
int t[N],top;
struct node{
    int n,s;
    vector<vector<int > >st;
    void init(int n,int val[]){
        this->n=n;
        this->s=log2(n);
        st.resize(n+1,vector(s+1,0));
        for(int i=1;i<=n;++i)   st[i][0]=val[i];
        for(int k=1;k<=s;++k){
            for(int i=1;i+(1<<k)-1<=n;++i){
                st[i][k]=max(st[i][k-1],st[i+(1<<(k-1))][k-1]);
            }
        }
    }
    int query(int l,int r){
        int k=log2(r-l+1);
        return max(st[l][k],st[r-(1<<k)+1][k]);
    }
}ST;
ll ans[N];
int ql[N],qr[N];
```

```
vector<int >Q[N];

struct Msg{
    ll tot,sum;
    Msg():tot(0ll),sum(0ll) {}
    Msg(ll TOT,ll SUM):tot(TOT),sum(SUM) {}
    Msg operator + (const Msg &p)   const{//信息合并
        return Msg(tot+p.tot,sum+p.sum);
    }
};
struct Tag{
    ll lazy;
    Tag():lazy(0) {}
    Tag(ll LAZY):lazy(LAZY) {}
    operator bool () const {return lazy;}
    Tag operator * (const Tag & p)const{//懒标记合并
        return Tag(lazy+p.lazy);
    }
    Msg apply(const Msg &m)const{//懒标记下放
        return Msg(m.tot,m.sum+m.tot*lazy);
    }
};
Msg msg[N<<1];
Tag tag[N<<1];
void apply(int u,Tag t){
    tag[u]=tag[u]*t;
    msg[u]=t.apply(msg[u]);
}
void pushdown(int u){
    if(!tag[u]) return ;
    apply(ls,tag[u]);
    apply(rs,tag[u]);
    tag[u]=Tag();
}
void pushup(int u){
    msg[u]=msg[ls]+msg[rs];
}
void modify(int u,int l,int r,int L,int R,Tag t){
    if(L<=l&&r<=R){
        apply(u,t);
        return ;
    }
    int mid=(l+r)>>1;   pushdown(u);
    if(L<=mid)  modify(ls,l,mid,L,R,t);
    if(R>mid)   modify(rs,mid+1,r,L,R,t);
    pushup(u);
}
void modify(int u,int l,int r,int x,int v){
    if(l==r){
        msg[u].tot+=v;
        return ;
    }
```

```
    int mid=(l+r)>>1;   pushdown(u);
    if(x<=mid)  modify(ls,l,mid,x,v);
    else    modify(rs,mid+1,r,x,v);
    pushup(u);
}
Msg query(int u,int l,int r,int L,int R){
    if(L<=l&&r<=R)  return msg[u];
    int mid=(l+r)>>1;   pushdown(u);    Msg res=Msg();
    if(L<=mid)  res=res+query(ls,l,mid,L,R);
    if(R>mid)   res=res+query(rs,mid+1,r,L,R);
    pushup(u);
    return res;
}
int main()
{
    n=rd();
    for(int i=1;i<=n;++i)   a[i]=rd();
    for(int i=2;i<=n;++i)   b[i]=rd();

    ST.init(n,a);

    m=rd();
    for(int i=1;i<=m;++i){
        ql[i]=rd();
        qr[i]=rd();
        Q[qr[i]].pub(i);
    }

    t[++top]=1;
    modify(1,1,n,t[top],1);
    for(int i=2;i<=n;++i){
        int l=1;    int r=i-1;
        while(l<=r){
            int mid=(l+r)>>1;
            if(ST.query(mid,i-1)<b[i])  r=mid-1;
            else    l=mid+1;
        }
        modify(1,1,n,l,i,Tag(1));
        for(auto v:Q[i]){
            ans[v]=query(1,1,n,ql[v],qr[v]).sum;
        }
        while(top&&a[t[top]]<=a[i]){
            modify(1,1,n,t[top],-1);
            top--;
        }
        t[++top]=i;
        modify(1,1,n,t[top],1);
    }
    for(int i=1;i<=m;++i)   cout<<ans[i]<<"\n";
    return 0;
}
```

考察知识点

ST表 二分 单调栈 线段树