T1

签到题。

简单观察，发现最后的数为 $\sum_{i=1}^{n} a_i \times \binom{n-1}{i-1} \pmod m$

题目需要求出无关量，本质上是求系数为 0 的项，即求 $\binom{n-i}{i-1} \equiv 0 \pmod m$ 的 i。

因为 m 不为质数，所以没法直接处理阶乘，但很容易给出一个递推的 $O(n^2)$ 做法。

注意到我们只关心 $\binom{n-i}{i-1}$ 是否为 m 的倍数，m 的质因子也比较少 ≤ 10，所以我们可以直接记录每一个质因子的质数即可，这样就可以直接阶乘。

T2

简单题。

有个很重要的限制，一条边只能经过两次，说白了就是进入一棵子树之后，必须先确定叶节点有没有他的女朋友之后才能出去。

所以就能设计出树形 DP，$dp[u][0/1]$，表示 u 的子树是否有渡边女朋友，走完子树 u 的期望。

最终答案为 $dp[u][1]$。

考虑转移，首先，有小弟把守的节点，$dp[u][0]=0$，因为渡边家兴只需要询问小弟就可以走完这棵子树。

计算 $dp[u][0]$，就是 $\sum\limits_{v \in E(u)} dp[v][0]+2$，就是搜索完所有的儿子节点加上进入每个儿子节点的消耗。

计算 $dp[u][1]$，本质上我们是要确定一个搜索子树的顺序，让期望搜索时间最小，考虑对于一个顺序 p，期望时间是多少，记 v 子树中叶子个数为 sz_v，显然是

$$\sum_{i=1}^{k} (dp[p_i][1]+1) \times \frac{sz_{p_i}}{sz_u} + (dp[p_i][0]+2) \times \frac{\sum\limits_{j=i+1}^{k} sz_{p_j}}{sz_u}$$

即枚举所在子树，加上额外的搜索空子树的时间。

我们可以证明，按照 $\frac{dp[v][0]+2}{sz_v}$ 升序排序后的顺序是最优的，具体证明参考国王游戏。

T3

有些毒瘤的题。

简单观察发现每个字符都和其上下的位置高度相关，将 `NOI` 划分为 9 个部分，三个字母均分为中间和两边三个部分，再加上两个空白部分共 `11` 部分，考虑 DP，设 $dp[i][s][l][r]$ 表示第 i 列，状态为 s，上下为 l,r 的最大值。

其它的转移是简单的，我们着重强调 `N` 中间部分的转移。它转枚举了上一个上下端点 l',r'，向 l,r 转移，条件为 $l \le r'+1$，枚举当前的上端点 l，我们可以动态维护一个数组 $dp\_max[i]$ 表示考虑所有 $r=i, l' \le l$。转移使用**当前的 l 对应的 dp_max 数组**的 $[1,r]$ 的 $\max$ 即可。

这道题的思维难度甚至没有 T2 大。

T4

思维题。

找规律和暴力各有 10pts。

Solution1

先把 BFS 序转化为 $1, 2, \ldots n$，然后再来考虑。

因为要求平均高度，所以我们需要求出总高度和总个数，转化后一棵树的高度等于 n 号节点的深度。

现在考虑划分 $1-n$ 这个 BFS 序列。可以发现一个划分**最多只能对应一棵树。**

一个显然的必要条件是**相同高度的元素在 DFS 序列中递增**，另一个显然的必要条件是 DFS 序列中 $dep_{a_i}+1 \geq dep_{a_{i+1}}$。

事实上，这两个条件也是充分的。

设 $dp[i][0/1]$ 表示考虑到 i，总数和总高度。转移检查哪些左端点可以转移即可，事实上，可以转移的左端点一定是一个区间，可以处理 dp 数组的前缀和完成快速转移。

判断转移区间和结论证明，留作思考，后文有 Details。

如果可能，尽量不要看 Details，自行思考。

Solution2

考虑一个划分合法的必要条件，Solution1 中已经提到。

考虑哪些位置可以划分，把一个间隔看成一个 01 变量，选择划分为 1，发现可以转化为一个这样的问题：

你需要确定长度为 $n-1$ 的二进制串，有若干个限制。每个限制形如

- $[l, r]$ 至多有一个 1。即：$dep_{a_i}+1 \geq dep_{a_{i+1}}$ 转化而来的 $[a_i, a_{i+1})$
- x 位置必须为 1 。即：如果 $pos_i > pos_{i+1}$，那么由于同高度在 a 中的位置递增，则 i 位置必须为 1。

很容易设计出一个 $O(n^2)$ 的动态规划做法。

观察第一个条件，如果 $a_{i+1} > a_i$ 才会有第一个限制，如果 $a_i + 1 = a_{i+1}$，那么相当于这个限制不存在。

否则，因为保证一定存在一棵合法的树，所以 $[a_i, a_{i+1}]$ 区间的 pos 数组必须能够被划分为两个连续上升子序列。a_i 又和 a_{i+1} 紧紧挨在一起，那么得知必定会被划分，因为中间的某个数一定会冲突。我们很容易模拟得到被划分的位置，之后这个区间的所有数都不能再被划分。

对于没有限制的位置，我们划分与不划分的方案数是相同的，因此对期望的贡献是 0.5。

将所有位置的期望加起来就是答案。

Solution1 Details

Transform

看看样例，发现从 $1-n$ 排列比较好想，所以考虑先把点做个变换，弄成 $1-n$，所以 b 变成了 $1, 2, 3, \ldots, n$。

然后记 u 的深度为 dep_u，发现 $\forall i \in [1, n), dep_i \leq dep_{i+1} \leq dep_i + 1$。

然后又发现，对于 BFS 序列为 $1,2,3,...n$ 的树，一个点遍历子节点的顺序一定是**按编号从小到大遍历。**

所以，确定了每个点的深度和 DFS 序列。可以唯一确定一颗树。

感性证明的话，确定深度之后把点画出来，画在一个二维平面上，深度相同的点在一层按编号从小到大排列，然后在 DFS 序列上走，如果下一个点深度更大，那肯定是往下连，如果深度不变或者更小，那一定是回去了一部分再往下走了一个，这样的逻辑可以唯一确定一棵树。

自上到下，自左到右遍历二维平面所有点的过程，就是 BFS 的过程

请认真理解二维平面的含义。

内含比较严格的证明

本质上，确定深度的过程就是划分 b 序列的过程

Native DP Algorithm

所以考虑对 BFS 序列的划分过程 DP，设 $dp[i][k]$ 表示考虑到第 i 个点，深度为 k 的合法划分方案总数。

然后我们需要枚举一个左端点 j，考虑如何判断这个划分是否合法。

首先有个必要条件：**同一深度的点，在 DFS 序列上的位置必须递增**，因为我们遍历一个点的儿子的顺序是从小到大。**称该条件为条件一**

其次，我们模拟一下在 DFS 序列上走的过程，发现**一个点 u 的下一个点 x 一定有** $dep_u + 1 \geq dep_x$ ，原因是显然的。**称该条件为条件二**

满足了以上条件，我们可以说明一定可以构造出一棵唯一对应的树。具体的，对于一个点 u，下一个点是 v，找到它或者它的祖先 u'， 满足 $dep_v = dep_{u'} + 1$，那么 v 的父亲就是 u'。由于第一个条件，限制了处理的 v 一定是该层**第一个还没有安排父亲的点**。所以每个点只会**恰好被安排一次父亲**，得到一个合法的树。

这样的话，我们再记录一维划分起点，变成 $dp[i][j][k]$。

转移枚举当前起点和上一个划分的起点，判断两个条件可以简单的 $O(n)$ 做。

复杂度为 $O(n^4)$，因为对于 n 个深度 k ，一共只需要判断一次。

Observaion1&&Optimization1

- 发现其实不用记录具体每个深度有几个元素，只需要记录**深度之和与树的个数**就可以计算答案。
- 假设划分的区间为 $[l,r]$ ，深度为 d。条件二等价于，a 中 $[1,l)$ 的后一个元素 x，一定满足 $x \leq r$。因为每次加入一段区间的点之后，上一次的深度为 $d-1$ 的点的右端点，如果还没有确定深度，就会被确定为 d。所以加入后，存在右端点还没确定深度的点，其本身深度只能为 d 了。所以可以直接**判断交界处的深度关系**，判断方式是 $\forall i \in [1,n), a_i > j \vee a_{i+1} \geq i$。

40Pts 的代码运用了第二个观察，请阅读。

运用第一个观察，DP 状态简化为 $dp[i][j][0/1]$

再次运用第二个观察，其实已经无需记录 j，DP 状态进一步简化为 $dp[i][0/1]$。

上述做法的复杂度是 $O(n^3)$ 的，考虑优化。

参看 Codes 部分的 40Pts 做法。sum 数组的含义是，$sum[i][j]$ 表示以 i 结尾，深度为 j 的方案总数。

40Pts 的部分没有对层数做简化

Observaion2&&Optimization2

- 其实 dp 数组没有用，只需要记录一个 sum 数组就可以了。
- 条件一，显然可行的 j 是一个右端点为 i 的区间，对于每个 i，可以处理出满足 p 数组(参考题解开头的定义)区间递增的最小左端点，作为 j 左端点的限制。
- 条件二，显然可行的 j 是一个前缀，并且右端点随 i 增大，这限制了 j 的右端点。

由于条件一，二的限为区间转移限制，所以记录一下 sum 的前缀和即可做到转移 $O(n)$

利用尺取法的思想可以 $O(n^2)$ 的计算条件二的右端点。

但进一步观察，发现对条件二进行了一些无用的 check，每次移动端点时，有用的 check 只有一个，就是值为右端点本身的位置。

所以 check 变成了 $O(1)$，总复杂度 $O(n)$

参考 100Pts 代码，注意，其中的 dp 代表 40pts 写法中的 sum，sum 代表其前缀和。

Notes

- **DP 过程中最大值达到了 2^n，需要手写科学计数法，可以忽略指数差距过大的加减运算。**

Codes

40pts $O(n^3)$

```
#include<bits/stdc++.h>
using namespace std;
template<typename _type>
inline void read(_type &x){
    x=0;int f=1;char ch=getchar();
    while(ch!=45&&(ch>'9'||ch<'0'))ch=getchar();
    if(ch==45){f=-1,ch=getchar();}
    while(ch<='9'&&ch>='0'){x=x*10+ch-48;ch=getchar();}x*=f;
}const int N=205;
int i,j,k,n,s,t,m,tp1,tp2;
int a[N],p[N],b[N];
double dp[N][N][N],sum[N][N];
signed main()
{
    read(n);
    for(i=1;i<=n;i++)read(a[i]);
    for(i=1;i<=n;i++)read(b[i]),p[b[i]]=i;
    for(i=1;i<=n;i++)a[i]=p[a[i]];
    for(i=1;i<=n;i++)p[a[i]]=i;
    dp[1][1][1]=1;sum[1][1]=1;
    for(i=2;i<=n;i++){
        for(j=2;j<=i;j++){
            //条件1
            for(k=j+1;k<=i;k++)
            if(p[k-1]>p[k])break;

            if(k!=i+1)continue;

            //条件2
            for(k=1;k<n;k++)
            if(a[k]<j&&a[k+1]>i)break;
            //如果交界处，一个深度小于 d,另一个大于 d，那么寄。
            if(k!=n)continue;
```

```
            for(k=1;k<=n;k++)
            dp[i][j][k]+=sum[j-1][k-1];
        }
        for(j=1;j<=i;j++)
        for(k=1;k<=n;k++)
        sum[i][k]+=dp[i][j][k];

    }
    double ans=0,cnt=0;
    for(i=1;i<=n;i++){
        cnt+=sum[n][i];
        ans+=sum[n][i]*i;
    }
    printf("%0.3lf",ans/cnt);
    return 0;
}
```

100 pts $O(n)$

```
#include<bits/stdc++.h>
using namespace std;
template<typename _type>
inline void read(_type &x){
    x=0;int f=1;char ch=getchar();
    while(ch!=45&&(ch>'9'||ch<'0'))ch=getchar();
    if(ch==45){f=-1,ch=getchar();}
    while(ch<='9'&&ch>='0'){x=x*10+ch-48;ch=getchar();}x*=f;
}
const int N=202005;
int i,j,k,n,s,t,m,tp1,tp2;
int a[N],p[N],b[N],lst[N],far[N];
struct Double{
    double val;
    int p;
    Double cap(Double x){
        while(abs(x.val)>1e18){
            x.val/=2;
            x.p++;
        }
        while(abs(x.val)<1e-18){
            x.val*=2;
            x.p--;
        }
        return x;
    }
    void operator =(int x){p=0,val=x;}
    Double operator +(const Double &x){
        if(x.p-p>50)return x;
        if(p-x.p>50)return *this;
        return cap({val+x.val*pow(2,x.p-p),p});
    }
    void operator +=(const Double &x){*this=*this+x;cap(x);}
    Double operator -(){return Double{-val,p};}
    Double operator -(Double &x){return cap((*this)+(-x));}
    Double operator /(const Double &x){return cap({val/x.val,p-x.p});}
```

```
    double get(){return val*pow(2.0,p);}
};
Double dp[N][2],sum[N][2];
signed main()
{
    read(n);lst[1]=1;
    for(i=1;i<=n;i++)read(a[i]);
    for(i=1;i<=n;i++)read(b[i]),p[b[i]]=i;
    for(i=1;i<=n;i++)a[i]=p[a[i]];
    for(i=1;i<=n;i++)p[a[i]]=i;
    for(i=2;i<=n;i++){
        if(p[i]>p[i-1])lst[i]=lst[i-1];
        else lst[i]=i;
    }
    dp[1][0]=1,dp[1][1]=1;
    sum[1][0]=1,sum[1][1]=1;
    far[1]=1;
    for(i=2;i<=n;i++){
        for(j=far[i-1]+1;j<=i;j++){
            if(a[p[j-1]+1]<=i||p[j-1]==n)continue;
            else break;
        }
        far[i]=--j;
        if(far[i]>=lst[i]){
            dp[i][0]=sum[j-1][0]-sum[lst[i]-2][0];
            dp[i][1]=(sum[j-1][1]-sum[lst[i]-2][1])+(sum[j-1][0]-sum[lst[i]-2]
[0]);
        }
        sum[i][0]=dp[i][0]+sum[i-1][0];
        sum[i][1]=dp[i][1]+sum[i-1][1];
    }
    printf("%0.3lf",(dp[n][1]/dp[n][0]).get());
    return 0;
}
```