【算法分析】

根据期望的线性性，拿到假钞的朋友个数的期望值等于对于所有的$1 \le i \le m$，第i种假钞被拿走的张数的期望之和。

把$p_{i,j}$视为概率，我们先求出$g_{i,j}$表示第i种假钞有j张，被拿走的期望张数。

先定义$h_{i,j,k}$表示前j个人中有k个人喜欢第i种假钞的概率。

显然$h_{i,0,0}=1$，$h_{i,j,k}=h_{i,j-1,k-1}\times p_{j,i}+h_{i,j-1,k}\times(1-p_{j,i})$。

这样就有$g_{i,j}=\sum_{k=0}^{n}\min(k,j)\times h_{i,n,k}$。

再定义$f_{i,j}$表示前i种假钞选了j张被拿走的最大期望值。

转移为$f_{i,j}=\sum_{k=0}^{j}\{f_{i-1,j-k}+g_{i,k}\}$。复杂度$O(n^2m)$。

为了优化，我们考虑g数组的性质。

考虑g数组第二维的差分：$g_{i,j}-g_{i,j-1}=\sum_{k=j}^{n}h_{i,n,k}$。显然它非负且随j单调不增。

而这个差分的意义可以看成是当第i种假钞已经拥有了$j-1$张时，再多选一张有$g_{i,j}-g_{i,j-1}$的收益。

于是我们可以把f的 DP 改成贪心：分n次，每次选出一种假钞i使得$g_{i,c_i+1}-g_{i,c_i}$最大（c_i为第i种假钞已经选出的张数），并新加入一个第i种假钞。由于g数组的单调性易得这个贪心的正确性。

这里还有另一个问题：计算h的复杂度是$O(n^2m)$的。

但我们有$\sum_{k=0}^{n}h_{i,n,k}=1$且$(g_{i,j+1}-g_{i,j})-(g_{i,j+2}-g_{i,j+1})=h_{i,n,j+1}$，这相当于我们每次$c_i$待加一时只需重新计算$h_{i,n,c_i+1}$的值就可以得到$g_{i,c_i+2}-g_{i,c_i+1}$。而如果$h_{i,*,c_i}$的值已经知道了，$h_{i,*,c_i+1}$的值可以$O(n)$DP 出来。

最终的复杂度为$O(n^2+nm)$。

【参考程序】

```
// 陈栉旷
#include <bits/stdc++.h>

template <class T>
inline void read(T &res)
{
    res = 0; bool bo = 0; char c;
    while (((c = getchar()) < '0' || c > '9') && c != '-');
    if (c == '-') bo = 1; else res = c - 48;
    while ((c = getchar()) >= '0' && c <= '9')
        res = (res << 3) + (res << 1) + (c - 48);
    if (bo) res = ~res + 1;
}

template <class T>
inline T Max(const T &a, const T &b) {return a > b ? a : b;}

const int N = 3005, M = 305;

int n, m, pt[M];
double p[N][M], ft[M], now[M], f[M][N], tmp[N], fsum[M], fsum2[M], ans;
```

```
void calc(int x) // 计算新的 h 值
{
    ft[x] += now[x]; pt[x]++;
    if (pt[x] > n) return (void) (now[x] = -1);
    for (int i = 0; i <= n; i++) tmp[i] = f[x][i];
    f[x][pt[x] - 1] = 0;
    for (int i = pt[x]; i <= n; i++)
        f[x][i] = f[x][i - 1] * (1.0 - p[i][x]) + tmp[i - 1] * p[i][x];
    fsum[x] += f[x][n]; fsum2[x] += f[x][n] * pt[x];
    now[x] = fsum2[x] + (1.0 - fsum[x]) * pt[x] - ft[x];
}

int main()
{
    freopen("coin.in", "r", stdin);
    freopen("coin.out", "w", stdout);

    int x;
    read(n); read(m);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            read(x), p[i][j] = 1.0 * x / 1000;
    for (int i = 1; i <= m; i++)
    {
        f[i][0] = 1;
        for (int j = 1; j <= n; j++)
            f[i][j] = f[i][j - 1] * (1.0 - p[j][i]);
        fsum[i] = f[i][n];
        calc(i);
    }
    for (int T = 1; T <= n; T++)
    {
        int it = 1;
        for (int i = 1; i <= m; i++)
            if (now[i] > now[it]) it = i; // 选出增量最大的一种假钞，为 it
        ans += now[it]; calc(it);
    }
    return printf("%.10lf\n", ans), 0;
}
```

【算法分析】

根据期望的线性性，我们考虑每一个点对答案的贡献。

注意到无论我们在什么时候选择了一个点，如果选择了这个点之后当前局面还不满足终止条件，那么下一步的决策一定是再随机选择一个点，也就是说，一个点被选择后期望的移动距离就是这个点到所有点的距离之和除以 n，这很容易用一个树形 DP 求出。

现在我们只需要计算出一个点期望被选择的次数（要求选择后当前局面不满足终止条件），注意到这已经和树的结构无关了，并且在固定了 n 个点中权值为 1 的点的个数之后，所有权值为 0 的点和所有权值为 1 的点期望被选择的次数相同。

于是我们设 $f_{i,0/1}$ 表示有 i 个权值为 1 的点时某一个权值为 0/1 的点期望被选择的次数，容易得到转移方程：

$$(1)\ f_{i,0}=\frac{i}{n}f_{i-1,0}+\frac{n-i-1}{n}f_{i+1,0}+\frac{1}{n}f_{i+1,1}+\frac{1}{n}$$

$$(2)\ f_{i,1=}\frac{i-1}{n}f_{i-1,1}+\frac{n-i}{n}f_{i+1,1}+\frac{1}{n}f_{i-1,0}+\frac{1}{n}$$

即考虑下一步选择的点以及是否计入次数。

特别地，$f_{0,0/1}=f_{n,0/1}=0$。

注意 $f_{1,1}$ 和 $f_{n-1,0}$ 的转移方程里没有 $\frac{1}{n}$ 这一项，因为此时若选择了这个点就直接结束了，不能计入次数。

因为转移是乱序的，相当于列出了关于若干个变量的若干个方程，现在我们需要解这个方程组，暴力高斯消元的时间复杂度 $O(n^3)$，不能通过所有数据。

考虑优化消元的过程。由方程(2)可得：已知 $f_{i-1,0/1}$ 和 $f_{i,0/1}$ 可以推出 $f_{i+1,1}$。我们再把 $f_{i+1,1}$ 代入方程(1)即可得到 $f_{i+1,0}$。因此我们可以把所有 $f_{i,0/1}$ 都表示成 $af_{1,0}+bf_{1,1}+c$ 的形式，再根据 $f_{n-1,0/1}$ 的转移方程列出一个二元一次方程组，由实际意义可知该方程组一定有唯一解，我们可以暴力解这个二元一次方程组得到 $f_{1,0/1}$，再暴力代回就得到了所有 $f_{i,0/1}$。

需要注意我们的转移是没有考虑第一步（不进行翻转）的随机选点的，最后计算答案时我们要令所有的 $f_{i,0/1}$ 加上 $\frac{1}{n}$。

时间复杂度 $O(n)$。

【参考代码】

```
//陈贤
#include <bits/stdc++.h>

template <class T>
inline void read(T &res)
```

```
{
    char ch;
    while (ch = getchar(), !isdigit(ch));
    res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
}

using std::vector;
const int mod = 1e9 + 7;
const int N = 1e5 + 5;
int inv[N], f[N], sze[N], g[N];
int n, cnt, ans; char s[N];
vector<int> e[N];

inline int quick_pow(int x, int k)
{
    int res = 1;
    while (k)
    {
        if (k & 1)
            res = 1ll * res * x % mod;
        x = 1ll * x * x % mod;
        k >>= 1;
    }
    return res;
}

inline void add(int &x, int y)
{
    x += y;
    x >= mod ? x -= mod : 0;
}

inline void dec(int &x, int y)
{
    x -= y;
    x < 0 ? x += mod : 0;
}

struct point
{ //定义三元组 (a,b,c) 表示 af[i][0] + bf[i][1] + c，方便运算
    int a, b, c;
```

```
    point() {}
    point(int A, int B, int C):
        a(A), b(B), c(C) {}

    inline void operator += (const point &x)
    {
        add(a, x.a);
        add(b, x.b);
        add(c, x.c);
    }

    inline void operator -= (const point &x)
    {
        dec(a, x.a);
        dec(b, x.b);
        dec(c, x.c);
    }

    inline point operator * (const int &x) const
    {
        return point(1ll * a * x % mod,
                     1ll * b * x % mod,
                     1ll * c * x % mod);
    }
}val[N][2];

//通过树形 DP 求出每个点到所有点的距离之和
inline void Dfs1(int x)
{
    sze[x] = 1;
    for (int i = 0, im = e[x].size(); i < im; ++i)
    {
        int y = e[x][i];
        Dfs1(y);
        sze[x] += sze[y];
        add(f[x], f[y]);
        add(f[x], sze[y]);
    }
}

inline void Dfs2(int x)
{
    int res = 0;
    for (int i = 0, im = e[x].size(); i < im; ++i)
```

```
    {
        int y = e[x][i];
        g[y] = f[x];
        dec(g[y], sze[y]);
        dec(g[y], f[y]);
        add(g[y], g[x]);
        add(g[y], n - sze[y]);
        Dfs2(y);
    }
    add(f[x], g[x]);
}

int main()
{
    freopen("C.in", "r", stdin);
    freopen("C.out", "w", stdout);

    read(n);
    scanf("%s", s + 1);
    for (int i = 1; i <= n; ++i)
        cnt += s[i] == '1';
    for (int i = 2, fa; i <= n; ++i)
    {
        read(fa);
        e[fa].push_back(i);
    }
    inv[1] = 1;
    for (int i = 2; i <= n; ++i)
        inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
    Dfs1(1);
    Dfs2(1);

    val[0][0] = val[0][1] = point(0, 0, 0);
    val[1][0] = point(1, 0, 0);
    val[1][1] = point(0, 1, 0);
    for (int i = 1; i < n; ++i)
    {
        point &tmp1 = val[i + 1][1];
        tmp1 = val[i][1];
        tmp1 -= val[i - 1][1] * (1ll * (i - 1) * inv[n] % mod);
        tmp1 -= val[i - 1][0] * inv[n];
        if (i > 1)
            dec(tmp1.c, inv[n]);
        tmp1 = tmp1 * (1ll * n * inv[n - i] % mod);
```

```
        point &tmp2 = val[i + 1][0];
        tmp2 = val[i][0];
        tmp2 -= val[i - 1][0] * (1ll * i * inv[n] % mod);
        tmp2 -= val[i + 1][1] * inv[n];
        if (i < n - 1)
        {
            dec(tmp2.c, inv[n]);
            tmp2 = tmp2 * (1ll * n * inv[n - i - 1] % mod);
        }
    }
    point tmp1 = val[n][0],
          tmp2 = val[n][1];
    tmp1.c = 1ll * (mod - 1) * tmp1.c % mod;
    tmp2.c = 1ll * (mod - 1) * tmp2.c % mod;
    tmp1 = tmp1 * (1ll * tmp2.a * quick_pow(tmp1.a, mod - 2) % mod);

    int y = 1ll * (tmp2.c + mod - tmp1.c) * quick_pow(tmp2.b + mod -
tmp1.b, mod - 2) % mod;
    int x = 1ll * (tmp2.c + mod - 1ll * tmp2.b * y % mod) *
quick_pow(tmp2.a, mod - 2) % mod;

    tmp1 = val[cnt][0];
    tmp2 = val[cnt][1];
    int v0 = (1ll * tmp1.a * x + 1ll * tmp1.b * y + tmp1.c) % mod,
        v1 = (1ll * tmp2.a * x + 1ll * tmp2.b * y + tmp2.c) % mod;
    add(v0, inv[n]);
    add(v1, inv[n]);
    for (int i = 1; i <= n; ++i)
        ans = (1ll * f[i] * inv[n] % mod * (s[i] == '1' ? v1 : v0) + ans) %
mod;
    printf("%d\n", ans);

    fclose(stdin); fclose(stdout);
    return 0;
}
```

由于题目中合并是连锁的，所以我们需要考虑在一定的长度限制下，序列的第一个数为定值的概率/期望。这启发我们按照序列的第一个数进行考虑（注意到题目中隐含的限制是「任何时刻，序列的长度都不会超过结束时的长度」，所以我们不需要对过程中序列的长度进行约束。且由于每次添加的数$\leq m$，所以任何时刻序列中的数字均不大于$lim = \min\{n + m - 1, t\}$）。

记$p_{i,j}$表示结束时序列长度为i、第一个数为j的概率。注意到这里仅对序列的第一个数有限制，则$p_{i,j}$亦可看作是序列长度始终不超过i，结束时序列仅有一个数j的概率。转移通过讨论j的来由分为两种情况：

1. j由两个$j-1$合并，概率为$p_{i,j-1} \times p_{i-1,j-1}$；
2. j是直接添加而成的，概率为$\frac{1}{m}$。

记$q_{i,j}$表示初始时序列只有一个数j，结束时序列长度为i、第一个数为j的概率。此时一定有第二个数不为j，则$q_{i,j} = 1 - p_{i-1,j}$（注意特判$j = lim$的情况）。

记$g_{i,j}$表示初始时序列只有一个数j，结束时序列长度为i、第一个数为j的情况下的期望。

记ans_i表示结束时序列长度为i的期望，则：

$$ans_i = \sum_{j=1}^{lim} p_{i,j} \times q_{i,j} \times g_{i,j}$$

最终答案即为ans_n。

问题转化为如何求$g_{i,j}$。由于$g_{i,j}$需保证第一个数始终为j，所以第二个数「始」「终」不为j。于是记$f_{i,j}$表示初始时序列只有一个数j，结束时序列长度为i的情况下的期望。则：

$$g_{i,j} = j + \frac{ans_{i-1} - p_{i-1,j} \times f_{i-1,j}}{q_{i,j}}$$

（注意特判$j = lim$的情况）。

现考虑如何求出$f_{i,j}$。由于$f_{i,j}$限制了初始时第一个数为j，则我们可以对结束时的第一个数进行讨论：

1. 结束时第一个数仍为j，贡献为$q_{i,j} \times g_{i,j}$；
2. 结束时第一个数大于j，贡献为$(1 - q_{i,j}) \times f_{i,j+1}$。

初始时$g_{1,j} = f_{i,j} = j$。

时间复杂度$O(nm \log(10^9 + 7))$。

【参考程序】

```
//潘恩宁
#include <bits/stdc++.h>
#define For(i, a, b) for (int i = a, bb = b; i <= bb; ++i)
#define Rof(i, a, b) for (int i = a, bb = b; i >= bb; --i)

typedef long long s64;

template <class T>
inline void get(T &res)
{
    char ch;
    bool bo = false;
    while ((ch = getchar()) < '0' || ch > '9')
```

```
        if (ch == '-') bo = true;

    res = ch - '0';
    while ((ch = getchar()) >= '0' && ch <= '9')
        res = (res << 1) + (res << 3) + ch - '0';

    if (bo) res = ~ res + 1;
    return;
}

template <class T>
inline void _put(T x)
{
    if (x > 9) _put(x / 10);
    putchar(x % 10 + '0');
    return;
}

template <class T>
inline void put(T x, char ch)
{
    if (x < 0)
        putchar('-'), x = ~ x + 1;
    _put(x);
    putchar(ch);
    return;
}

const int MaxN = 1005, mod = 1e9 + 7;
int p[MaxN][MaxN << 1], q[MaxN][MaxN << 1], g[MaxN][MaxN << 1],
f[MaxN][MaxN << 1];
//p[i][j]表示结束时序列长度为 i、第一个数为 j 的概率（亦可看作是序列长度始终不超过 i，结束时序列仅有一个数 j 的概率）
//q[i][j]表示初始时序列只有一个数 j，结束时序列长度为 i、第一个数为 j 的概率
//g[i][j]表示初始时序列只有一个数 j，结束时序列长度为 i、第一个数为 j 的情况下的期望
//f[i][j]表示初始时序列只有一个数 j，结束时序列长度为 i 的情况下的期望
int ans[MaxN], n, m, lim, inv;

inline void ckmin(int &x, const int &y)
{
    if (x > y) x = y;
    return;
}
```

```
inline int ksm(int x, int y)
{
    int res = 1;
    while (y)
    {
        if (y & 1)
            res = 1LL * res * x % mod;
        x = 1LL * x * x % mod;
        y >>= 1;
    }
    return res;
}

inline void Plus(int &x, const int &y)
{
    x += y;
    if (x >= mod) x -= mod;
    return;
}

int main()
{
    freopen("sequence.in", "r", stdin), freopen("sequence.out", "w",
stdout);

    get(n), get(m), get(lim);
    ckmin(lim, n + m - 1);
    inv = ksm(m, mod - 2);

    For(i, 1, n)
    For(j, 1, lim)
    {
        p[i][j] = (1LL * p[i][j - 1] * p[i - 1][j - 1] + (j <= m ? inv :
0)) % mod; //数 j 要么由两个 j-1 合成，要么直接添加而成
        q[i][j] = (1 - (j < lim ? p[i - 1][j] : 0) + mod) % mod; //结束时
第一个数为 j 则一定满足第二个数不为 j
    }

    For(j, 1, lim) //处理序列长度始终不超过 1 的情况
    {
        g[1][j] = f[1][j] = j;
        ans[1] = (1LL * p[1][j] * q[1][j] % mod * j + ans[1]) % mod;
    }
```

```
    For(i, 2, n)
    {
        For(j, 1, lim)
        {
            int tmp = ans[i - 1] - (j < lim ? 1LL * p[i - 1][j] * f[i - 1][j] % mod : 0) + mod; //tmp 表示结束时序列长度为 i-1、第一个数不为 j 的期望
            g[i][j] = (1LL * tmp * ksm(q[i][j], mod - 2) + j) % mod;
            ans[i] = (1LL * p[i][j] * q[i][j] % mod * g[i][j] + ans[i]) % mod; //累加结束时长度为 i、第一个数为 j 的期望
        }
        Rof(j, lim, 1) //初始时序列第一个数为 j，则结束时序列第一个数一定≥j
            f[i][j] = (1LL * q[i][j] * g[i][j] + 1LL * (1 - q[i][j] + mod) * f[i][j + 1]) % mod;
    }
    put(ans[n], '\n');

    fclose(stdin), fclose(stdout);
    return 0;
}
```