polya计数

组合数学 polya计数

定理总结：

计数总数 = 在每一种置换下染色不变的染色数/所有置换数

简单应用

hdoj 3923
题目意思是给我们n个小球，求用m种颜色在一个圈上排列的方法数，我们只需要列举出每一种置换的循环节个数就好。
1.旋转：
旋转的方法有n种，每一种的循环节个数为gcd(n, i)(0<=i<=n), 循环节长度为n/gcd(n, i)。
2.翻转：
翻转要分情况：
(1)n为奇数：当n是奇数时只有一种翻折方式，沿着一个点和对边的中点翻折。循环节个数为(n-1)/2+1.
(2)n为偶数：当n为偶数的时候有两种翻折方式，一个是沿着一个点和对边翻折，循环节为(n-2)/2+2，一个是沿着一条边的中点和对边的中点翻折，循环节是n/2.

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const long long mod = 1e9 + 7;
int n, c;

long long p(long long base, long long r)
{
    long long ret = 1;
    while (r) {
        if (r & 1) {
            ret = (ret * base) % mod;
        }
        r >>= 1;
        base = (base * base) % mod;
    }
    return ret;
}

int gcd(int a, int b)
{
    int r;
    while (b) {
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}

int main()
{
    int t, tc = 1;
    scanf("%d", &t);
    while (t --) {
        scanf("%d%d", &c, &n);
        long long ans;
        if (n & 1) {
            ans = (1LL * p(c, (n + 1) / 2) * n % mod);
        } else {
            ans = (1LL * n / 2 * p(c, n / 2) % mod + 1LL * n / 2 * p(c, n / 2 + 1) % mod) % mod;
        }
        for (int i = 1; i <= n; i ++) {
            ans = (ans + p(c, gcd(n, i))) % mod;
```

```
        }
        ans = (ans * p(2 * n, mod - 2) % mod);
        printf("Case #%d: %d\n", tc ++, ans);
    }
    return 0;
}
```

欧拉函数优化

poj2154
题目要求：给出两个整数n和p，代表n个珠子，n种颜色，要求不同的项链数，并对结果mod（p）处理。
置换只有旋转一种方式，那么共有n个置换.

暴力枚举每一种置换会超时，可以考虑这样的优化：我们只枚举循环节的长度l（这样循环节的个数就是$fracnl$），显然l必然是n的因数，如果可以这样做，我们就可以在$\sqrt{n}$的时间里完成。
我们知道对于第i个置换，其循环节长度为$gcd(n,i)$,我们要找的就是使$gcd(n,i)=l$的i的个数，也就是
$S_n^l = s^1_{\frac{n}{l}} = \phi(\frac{n}{l})$

$$ans = \frac{\sum_{l|n}\phi(l)\cdot n^{\frac{n}{l}}}{n}$$
$$= \sum_{l|n}\phi(l)\cdot n^{\frac{n}{l}-1}$$

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;
const int MAX = 1e5 + 10;
int n, mod, vis[MAX], prime[MAX], cnt, euler[MAX];

void init()
{
    euler[1] = 1;
    for (int i = 2; i < MAX; i ++) {
        if (!vis[i]) {
            prime[cnt ++] = i;
            euler[i] = i - 1;
        }
        for (int j = 0; j < cnt; j ++) {
            if (1LL * i * prime[j] >= MAX) break;
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                euler[i * prime[j]] = euler[i] * prime[j];
                break;
            } else {
                euler[i * prime[j]] = euler[i] * (prime[j] - 1);
            }
        }
        //cout << euler[i] << " " ;
    }
}

int p(int base, int r)
{
    int ret = 1;
    while (r) {
        if (r & 1) {
            ret = (ret * base) % mod;
        }
        r >>= 1;
        base = (base * base) % mod;
    }
    return ret;
}

int getEuler(int x)
{
    if (x < MAX) return euler[x];
    int ret = x;
```

```
    for (int i = 0; i < cnt; i ++) {
        if (prime[i] * prime[i] > x) break;
        if (x % prime[i] == 0) {
            ret = ret / prime[i] * (prime[i] - 1);
            x /= prime[i];
            while (x % prime[i] == 0) {
                x /= prime[i];
            }
        }
    }
    if (x != 1) {
        ret = ret / x * (x - 1);
    }
    return ret;
}

int main()
{
    init();
    int t;
    scanf("%d", &t);
    while (t --) {
        scanf("%d%d", &n, &mod);
        int ans = 0;
        for (int l = 1; l * l <= n; l ++) {
            if (n % l == 0) {
                // 乘方幂次减一的原因是最后要除以总置换数n
                if (l * l < n) {
                    ans = (ans + getEuler(l) % mod * p(n % mod, (n / l - 1) )) % mod;
                    ans = (ans + getEuler(n / l) % mod * p(n % mod, (l - 1) )) % mod;
                } else {
                    ans = (ans + getEuler(l) % mod * p(n % mod, (l - 1) )) % mod;
                }
            }
        }
        printf("%d\n", ans);
    }
}
```

矩阵优化

poj 2888
题意：做一串项链，长度为n，有m种珠子，旋转后项链和旋转前项链视为同一种。有k种限制，表示两种颜色的珠子不能挨在一起，求染色数。

除了用到了上面的euler函数优化之外，还用到了一个计数技巧：
在图上求从a走到b恰好用i步的走法=(邻接矩阵^i)[a][b]

试想：可以把循环节看作是一个回路，假设循环节的长度为l，那么第1个珠子和第$l+1$个珠子的颜色是一样的，枚举第1个珠子的颜色，上色的种数就是从第一个颜色出发，走l步回到第1种颜色的路径数。
因此，在循环节长度为l的置换下，染色不变的染色数就是邻接矩阵的l次幂的迹。

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int MAX = 1e9 + 10;
const int MAXM = 12;
const int MAXP = 1e5 + 10;
const int mod = 9973;
int prime[MAXP], cnt, vis[MAXP], n, k, m;

struct Matrix {
    int r, c;
    int mat[MAXM][MAXM];
    Matrix () : r(m), c(m) {
        memset(mat, 0, sizeof mat);
    }
    Matrix (int i) : r(m), c(m) {
        memset(mat, 0, sizeof mat);
        for (int j = 1; j <= r; j ++) {
            mat[j][j] = i;
```

```
        }
    }
    Matrix (int rr, int cc) : r(rr), c(cc) {
        memset(mat, 0, sizeof mat);
    }
    void fill(void) {
        for (int i = 1; i <= r; i ++) {
            for (int j = 1; j <= c; j ++) {
                mat[i][j] = 1;
            }
        }
    }
    void mul(const Matrix& rhs) {
        Matrix ret;
        for (int i = 1; i <= r; i ++) {
            for (int j = 1; j <= rhs.c; j ++) {
                for (int k = 1; k <= c; k ++) {
                    ret.mat[i][j] = ((ret.mat[i][j] + mat[i][k] * rhs.mat[k][j]) ) % mod;
                }
            }
        }
        memcpy(mat, ret.mat, sizeof ret.mat);
    }
    static Matrix p(Matrix base, long long r) {
        Matrix ret;
        ret.c = ret.r = m;
        for (int i = 1; i <= m; i ++) {
            ret.mat[i][i] = 1;
        }
        while (r) {
            if (r & 1) {
                ret.mul(base);
            }
            r >>= 1;
            base.mul(base);
        }
        return ret;
    }
} g;

int quickp(int base, int r)
{
    int ret = 1;
    while (r) {
        if (r & 1) {
            ret = (ret * base) % mod;
        }
        r >>= 1;
        base = (base * base) % mod;
    }
    return ret;
}


void getprime()
{
    for (int i = 2; i < MAXP; i ++) {
        if (!vis[i]) {
            prime[cnt ++] = i;
        }
        for (int j = 0; j < cnt; j ++) {
            if (i * prime[j] >= MAXP) break;
            vis[i * prime[j]] = 0;
            if (i % prime[j] == 0) {
                break;
            }
        }
    }
}

int euler(int x)
{
    int i = 0, ret = x;
    while (prime[i] * prime[i] <= x) {
        if (x % prime[i] == 0) {
            ret = ret / prime[i] * (prime[i] - 1);
            do {
                x /= prime[i];
```

```
            } while (x % prime[i] == 0);
        }
        ++ i;
    }
    if (x != 1) {
        ret = ret / x * (x - 1);
    }
    return ret % mod;
}

int gettr(const Matrix& mm, long long r)
{
    int ret = 0;
    Matrix mmm = Matrix::p(mm, r);
    for (int i = 1; i <= mmm.r; i ++) {
        ret = (ret + mmm.mat[i][i]) % mod;
    }
    return ret;
}

int polya(void)
{
    int ans = 0;
    for (int i = 1; i * i <= n; i ++) {
        if (n % i == 0) {
            if (i * i != n) {
                ans = (ans + euler(n / i) * gettr(g, i)) % mod;
                ans = (ans + euler(i) * gettr(g, n / i)) % mod;
            } else {
                ans = (ans + euler(i) * gettr(g, i)) % mod;
            }
        }
    }
    ans = (ans * quickp(n % mod, mod - 2)) % mod;
    return ans;
}

int main()
{
    getprime();
    int t;
    scanf("%d", &t);
    while (t --) {
        scanf("%d%d%d", &n, &m, &k);
        g.c = g.r = m;
        g.fill();
        int u, v;
        for (int i = 1; i <= k; i ++) {
            scanf("%d%d", &u, &v);
            g.mat[u][v] = g.mat[v][u] = 0;
        }
        printf("%d\n", polya());
    }
}
```