生成树求和(sum,1s,512MB)

【算法分析】

显然3进制下每一位的贡献独立，可以分开算。

假设现在枚举到第T位，设每条边的新边权为原边权3进制意义下第T位的值，那么如果我们能够求出有多少棵生成树的边权之和分别为$0,1,\dots 2(n-1)$，那么我们就能计算出原图有多少棵生成树上所有边权通过三进制不进位加法得到的第T位值为0,1,2，这样就能比较方便地计算答案了。

考虑使用 Matrix-Tree 矩阵树定理。由于矩阵树定理的本质是求所有生成树的边权积之和，故考虑把每条边的权值设成一个多项式。具体地，如果某一条边原边权的第T位值为k，就在 Matrix-Tree 中把这条边的权值设为x^k。易得这样进行 Matrix-Tree 矩阵树定理求得的行列式是一个最高$2(n-1)$次的多项式，其中i次项表示选出边的k之和为i的方案数。

直接这么做的复杂度是$O(n^5 \log c)$，再加上多项式的项数有个2的常数，难以通过此题。不过我们可以注意到，由于求得行列式的多项式最高次数不会超过$2(n-1)$，可以考虑插值。具体地，对矩阵每个位置上的多项式代入$x=1,2,\dots 2n-1$得到$2n-1$个点值，然后对于每个$x=1,2,\dots 2n-1$都进行一遍高斯消元求行列式，就能得到待求的行列式的点值表达，最后插值回去即可得到系数。这样的复杂度是$O(n^4 \log c)$，可以通过此题。

【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;
}

const int N = 44, E = 10, R = 84, M = 1005, rqy = 1e9 + 7;

int n, m, U[M], V[M], W[M], l, p3[E], a[N][N], px[R][3], tmp[R], ans;

int qpow(int a, int b)
{
    int res = 1;
    while (b)
    {
        if (b & 1) res = 1ll * res * a % rqy;
        a = 1ll * a * a % rqy;
        b >>= 1;
    }
    return res;
}

int jiejuediao(int T)
{
    int res = 0;
```

```
    for (int u = 1; u <= l; u++) //枚举 x 的取值从 1 到 2n-1
    {
        memset(a, 0, sizeof(a));
        for (int i = 1; i<= m; i++) //求矩阵每个位置的点值
        {
            int w = px[u][W[i] / p3[T] % 3];
            a[U[i]][V[i]] = a[V[i]][U[i]] = (rqy - w) % rqy;
            a[U[i]][U[i]] = (a[U[i]][U[i]] + w) % rqy;
            a[V[i]][V[i]] = (a[V[i]][V[i]] + w) % rqy;
        }
        int delta = 1;
        for (int i = 1; i< n; i++) //高斯消元求行列式
        {
            int tmp = qpow(a[i][i], rqy - 2);
            delta = 1ll * delta * a[i][i] % rqy;
            for (int j = i + 1; j < n; j++)
            {
                int pmt = 1ll * tmp * a[j][i] % rqy;
                for (int k = i; k < n; k++)
                  a[j][k] = (a[j][k] - 1ll*a[i][k] * pmt % rqy + rqy) % rqy;
            }
        }
        memset(tmp, 0, sizeof(tmp)); tmp[0] = 1;
        for (int j = 1; j <= l; j++) if (j != u) // 插值
            for (int k = j - (j > u); k >= 0; k--)
              tmp[k] = ((k ?tmp[k-1] : 0) - 1ll*tmp[k]*j % rqy + rqy) % rqy;
        int pr = 1;
        for (int j = 1; j <= l; j++)
            if (j != u) pr = 1ll * pr * (u - j + rqy) % rqy;
        delta = 1ll * delta * qpow(pr, rqy - 2) % rqy;
        for (int j = 0; j < l; j++)
            res = (1ll * j % 3 * delta * tmp[j] + res) % rqy;
    }
    return res;
}

int main()
{
    #ifdef nealchentxdy
    #else
        freopen("sum.in", "r", stdin);
        freopen("sum.out", "w", stdout);
    #endif

    read(n); read(m); l = (n << 1) - 1;
    for (int i = 1; i<= m; i++) read(U[i]), read(V[i]), read(W[i]);
    p3[0] = 1;
    for (int i = 1; i<= 8; i++) p3[i] = 3 * p3[i - 1];
    for (int i = 1; i<= l; i++) px[i][0] = 1, px[i][1] = i, px[i][2] = i*i;
    for (int i = 0; i<= 8; i++)
        ans = (1ll * p3[i] * jiejuediao(i) + ans) % rqy;
    return std::cout<<ans<< std::endl, 0;
```

}

距离之和(distance,1s,256MB)

【算法分析】

记a_i表示到原点距离为i的点数，那么答案为：$\sum_{i=1}^{N} a_i \times i$。

a_i不方便直接计算，那么记b_i表示到原点距离$\geq i$的点数（不包括原点），答案为：$\sum_{i=1}^{N} b_i$。

考虑用补集转化求b_i，即总点数减去到原点距离$< i$的点数。到原点距离$< i$的点要满足每一维都$< i$，那么有：$b_i = (2N+1)^K - (2i-1)^K$。那么答案为$\sum_{i=1}^{N}(2N+1)^K - (2i-1)^K$，求出$\sum_{i=1}^{N}(2i-1)^K$即可。

发现$(2i-1)^K$是K次多项式，而答案是K次多项式的前缀和，也就是说，$\sum_{i=1}^{N}(2i-1)^K$是关于N的$K+1$次多项式。

那么考虑lagrange插值求解。记$f(L) = \sum_{i=1}^{L}(2i-1)^K$，要先算出$f(1), f(2), \ldots, f(K+1), f(K+2)$。直接求是$O(K\log K)$的，考虑优化。我们可以对$2K+3$以内的每个数$x$，预处理出$g(x) = x^K$。显然$g(x)$是积性函数，可以线性筛。线性筛中只要对每个质数$x$，直接用快速幂求$x^K$，剩下的$g_x$都可以$O(1)$递推得到。设$\pi(x)$表示不大于$x$的质数个数，有$\pi(x) \approx \frac{x}{\ln x}$，那么预处理的时间复杂度约为$O(K)$。

直接lagrange插值是$O(K^2)$的。观察插值的式子：$f(N) = \sum_{i=1}^{K+2} f(i) \prod_{j=1, j\neq i}^{K+2} \frac{N-j}{i-j}$。我们可以预处理出$pre[i] = \prod_{j=1}^{i}(N-j), suf[i] = \prod_{j=i}^{K+2}(N-j)$。那么$f(N) = \sum_{i=1}^{K+2} f(i) \frac{pre[i-1] \times suf[i+1]}{(i-1)! \times (K+2-i)! \times (-1)^{K+2-i}}$。

时间复杂度$O(K)$。

【参考程序】

```
#include <bits/stdc++.h>

usingnamespace std;

#define ll longlong

constint e = 2e6 + 5, mod = 1e9 + 7;

int L, T, n, k, ans, a[e], b[e], f[e], inv[e], p[e], pri[e], cnt, c[e];
int pre[e], suf[e];
bool bo[e];

inlineint plu(int x, int y) {
    (x += y) >= mod && (x -= mod);
    return x;
}

inlineint sub(int x, int y) {
    (x -= y) <0&& (x += mod);
    return x;
}

inlineint ksm(int x, int y) {
int res = 1;
while (y) {
```

```
if (y &1)
            res = (ll)res * x % mod;
        y >>= 1;
        x = (ll)x * x % mod;
    }
    return res;
}

inlinevoid init() {
int i, j;
    L = 2 * k + 4;
    p[1] = 1;
    cnt = 0;
    for (i = 2; i <= L; i++) {
    if (!bo[i]) {
            pri[++cnt] = i;
            p[i] = ksm(i, k);
        }
    for (j = 1; j <= cnt && i * pri[j] <= L; j++) {
            bo[i * pri[j]] = 1;
            p[i * pri[j]] = (ll)p[i] * p[pri[j]] % mod;
            if (i % pri[j] == 0)
            break;
        }
    }
}

inlinevoid solve() {
int i, j;
    scanf("%d%d", &k, &n);
    init();
    ans = (ll)n * ksm((2 * n + 1) % mod, k) % mod;
    a[0] = b[0] = 1;
    inv[1] = 1;
    for (i = 2; i <= k + 2; i++)
       inv[i] = (ll)(mod - mod / i) * inv[mod % i] % mod;
    for (i = 1; i <= k + 2; i++) {
        f[i] = plu(f[i - 1], p[2 * i - 1]);
        a[i] = (ll)a[i - 1] * inv[i] % mod;
        b[i] = (ll)b[i - 1] * (mod - inv[i]) % mod;
    }
    if (n <= k + 2) {
        printf("%d\n", sub(ans, f[n]));
        return;
    }
    pre[0] = suf[k + 3] = 1;
    for (i = 1; i <= k + 2; i++) pre[i] = (ll)pre[i - 1] * (n - i) % mod;
    for (i = k + 2; i >= 1; i--) suf[i] = (ll)suf[i + 1] * (n - i) % mod;
    for (i = 1; i <= k + 2; i++) {
    int res=(ll)f[i]%mod*pre[i-1]%mod*suf[i+1]%mod*a[i-1]%mod*b[k+2-i]%mod;
        ans = sub(ans, res);
    }
```

```
    printf("%d\n", ans);
}

int main() {
    freopen("distance.in", "r", stdin);
    freopen("distance.out", "w", stdout);
    scanf("%d", &T);
    while (T--) solve();
    fclose(stdin); fclose(stdout);
    return0;
}
```

染色（color，3s，512MB）

【算法分析】

首先b_i的限制比较不太容易处理，我们考虑设c_i表示**至多**用i种颜色进行染色的不同方案的分数和。那么不难得到下面两个式子

$$(1)c_i = \sum_{j=0}^{i} \binom{i}{j} b_j$$

$$(2)c_i = \sum_{j=0}^{n} a_j \cdot i^j$$

我们可以将式子(1)拆开，就可以用FFT求出c_i了：

$$c_i = \sum_{j=0}^{i} \frac{i!}{j!\,(i-j)!} b_j = i! \sum_{j=0}^{i} \frac{b_j}{j!} \cdot \frac{1}{(i-j)!}$$

观察式子(2)我们又发现，c_i即为a的生成函数$A(x) = \sum_{i=0}^{n} a_i x^i$在$x = i$处的点值。因此要求$A(x)$的系数表示，我们可以直接考虑将$c_i$作为点值来插值出多项式。

考虑将其代入拉格朗日插值的式子中

$$A(x) = \sum_{i=0}^{n} c_i \frac{\prod_{j \neq i}(x-j)}{\prod_{j \neq i}(i-j)}$$

设$d_i = \frac{c_i}{\prod_{j \neq i}(i-j)}$，那么$d_i$是可以比较容易求出的。

设$M = \prod_{i=0}^{n}(x-i)$，那么插值的式子就可以表示为

$$A(x) = \sum_{i=0}^{n} \frac{d_i M}{x-i}$$

那么这个式子就比较容易处理了，观察到每个d_i所乘的多项式是$\frac{M}{x-i}$，这启发我们用分治处理。设

$$M_{l,r} = \sum_{i=l}^{r} (x-i), \quad A_{l,r} = \sum_{i=l}^{r} \frac{d_i M_{l,r}}{x-i}$$

那么令$mid = \left\lfloor \frac{l+r}{2} \right\rfloor$，就有

$$A_{l,r} = \sum_{i=l}^{r} \frac{d_i M_{l,r}}{x-i}$$

$$= M_{mid+1,r} \sum_{i=l}^{mid} \frac{d_i M_{l,mid}}{x-i} + M_{l,mid} \sum_{i=mid+1}^{r} \frac{d_i M_{mid+1,r}}{x-i}$$

$$= A_{l,mid}M_{mid+1,r} + A_{mid+1,r}M_{l,mid}$$

那么就A, M一起分治来求即可，乘法用FFT实现，时间复杂度为$O(n\log^2 n)$。

【参考程序】

```
//陈煜翔
#include <bits/stdc++.h>

template <class T>
inline void read(T &x)
{
    static char ch;
    while (!isdigit(ch = getchar()));
    x = ch - '0';
    while (isdigit(ch = getchar()))
        x = x * 10 + ch - '0';
}

template <class T>
inline void putint(T x)
{
    static char buf[15], *tail = buf;
    if (!x)
        putchar('0');
    else
    {
        for (; x; x /= 10) *++tail = x % 10 + '0';
        for (; tail != buf; --tail) putchar(*tail);
    }
}

typedef std::vector<int> vi; //用 std::vector<int>表示多项式
#define mp(x, y) std::make_pair(x, y)

const int mod = 998244353;

inline int qpow(int x, int y) //快速幂求 x^y
{
    int res = 1;
    for (; y; y >>= 1, x = 1LL * x * x % mod)
        if (y & 1)
            res = 1LL * res * x % mod;
    return res;
}

inline void add(int &x, const int &y) //模意义下的 x+=y
{
    x += y;
    if (x >= mod)
        x -= mod;
}

inline void dec(int &x, const int &y)//模意义下的 x-=y
```

```
{
    x -= y;
    if (x < 0)
        x += mod;
}

const int MaxN = 1e6 + 5;

namespace polynomial
{
    int L, P;
    int rev[MaxN];

    inline void init(int n)
    {
        P = 0, L = 1;
        while (L < n)
            L <<= 1, ++P;

        for (int i = 1; i < L; ++i)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (P - 1));
    }

    inline void DFT(vi &a, int n, int opt) //给 a 做 DFT/IDFT 变换
    {
        for (int i = 0; i < n; ++i)
            if (i < rev[i])
                std::swap(a[i], a[rev[i]]);

        int g = opt == 1 ? 3 : (mod + 1) / 3;
        for (int k = 1; k < n; k <<= 1)
        {
            int omega = qpow(g, (mod - 1) / (k << 1));
            for (int i = 0; i < n; i += k << 1)
            {
                int x = 1;
                for (int j = 0; j < k; ++j)
                {
                    int u = a[i + j];
                    int v = 1LL * a[i + j + k] * x % mod;
                    add(a[i + j] = u, v);
                    dec(a[i + j + k] = u, v);
                    x = 1LL * x * omega % mod;
                }
            }
        }
        if (opt == -1)
        {
            int inv = qpow(n, mod - 2);
            for (int i = 0; i < n; ++i)
                a[i] = 1LL * a[i] * inv % mod;
        }
```

```
    }

    inline vi mul(vi a, vi b) //将 a,b 两个多项式相乘
    {
        int na = a.size(), nb = b.size();
        int n = na + nb - 1;

        init(n);
        a.resize(L), b.resize(L);

        DFT(a, L, 1), DFT(b, L, 1);
        for (int i = 0; i < L; ++i)
            a[i] = 1LL * a[i] * b[i] % mod;
        DFT(a, L, -1);

        return a.resize(n), a;
    }

    inline vi plus(vi a, vi b) //将 a,b 两个多项式相加
    {
        vi res = a;
        int n = std::max(a.size(), b.size());
        res.resize(n);

        for (int i = 0; i < n; ++i)
            add(res[i], b[i]);
        return res;
    }
}

inline vi convert(int *a, int n) //将数组转换成 vector 的形式
{
    vi res(0);
    for (int i = 0; i < n; ++i)
        res.push_back(a[i]);
    return res;
}

int n, b[MaxN];
int fac[MaxN], fac_inv[MaxN];

vi val, ans;
int c[MaxN];

inline void c_init()
{
    for (int i = 0; i < n; ++i)
    {
        c[i] = 1LL * val[i] * fac_inv[i] % mod * fac_inv[n - i - 1] % mod;
        if ((n - i - 1) & 1)
            c[i] = c[i] ? mod - c[i] : 0;
    }
```

```
}

inline void fac_init(int n)
{
    fac[0] = 1;
    for (int i = 1; i <= n; ++i)
        fac[i] = 1LL * fac[i - 1] * i % mod;
    fac_inv[n] = qpow(fac[n], mod - 2);
    for (int i = n - 1; i >= 0; --i)
        fac_inv[i] = 1LL * fac_inv[i + 1] * (i + 1) % mod;
}

inline std::pair<vi, vi> solve(int l, int r) //分治求 M 和 A
{
    if (l == r)
    {
        vi t(2);
        t[0] = l ? mod - l : 0;
        t[1] = 1;
        return mp(t, vi(1, c[l]));
    }
    int mid = (l + r) >> 1;
    std::pair<vi, vi> pl = solve(l, mid);
    std::pair<vi, vi> pr = solve(mid + 1, r);

    using namespace polynomial;
    return mp(mul(pl.first, pr.first), plus(mul(pl.first, pr.second),
mul(pl.second, pr.first)));
}

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

    read(n); ++n;
    for (int i = 0; i < n; ++i)
        read(b[i]);

    fac_init(n);
    for (int i = 0; i < n; ++i)
        b[i] = 1LL * b[i] * fac_inv[i] % mod;

    val = polynomial::mul(convert(b, n), convert(fac_inv, n));
    val.resize(n); //先用卷积求出 c

    for (int i = 0; i < n; ++i)
        val[i] = 1LL * val[i] * fac[i] % mod;
    c_init();

    ans = solve(0, n - 1).second;
    for (int i = 0; i < n; ++i)
```

```
    {
        putint(ans[i]);
        putchar(" \n"[i == n - 1]);
    }

    return 0;
}
```