前缀的或(or,1s,512MB)

【算法分析】

DP 状态非常显然：f[i][j]表示A序列的前i个元素有多少种取值方案，要求满足（1）对于所有的$h \in [1,i]$都有$A_h \in [1,2^j)$（2）对于任意的$1 < h \le i$都有$B_h > B_{h-1}$（3）$B_i = 2^j - 1$这三个条件。

$$f[0][0] = 1$$

$$f[i][j] = \sum_{h=0}^{j-1} f[i-1][h] \times \binom{j}{h} \times 2^h$$

转移即枚举B_{i-1}有多少个1记作h，那么A_i中有h位可为0可为1，剩下的$j-h$位必须为1才能使得$B_i = 2^j - 1$。

答案为$\sum_{i\ge 0} f[n][i] \times \binom{k}{i}$。

这个转移可以移项后变成

$$\frac{f[i][j]}{j!} = \sum_{h=0}^{j-1} \frac{f[i-1][h]}{h!} \times \frac{1}{(j-h)!} \times 2^h$$

设生成函数$F_i(x) = \sum_{j\ge 0} \frac{f[i][j]}{j!} x^j$，$G(x) = \sum_{i>0} \frac{1}{i!}$。

可以得出$F_i(x) = F_{i-1}(2x) \times G(x)$。

于是$F_n(x) = \prod_{i=0}^{n-1} G(2^i x)$。

考虑求得了$F_t(x)$之后如何得到$F_{2t}(x)$。

不难发现：

$$F_{2t}(x) = \left(\prod_{i=0}^{t-1} G(2^i x)\right)\left(\prod_{i=0}^{t-1} G(2^i \times 2^t x)\right) = F_t(x) \times F_t(2^t x)$$

使用 FFT 即可通过$F_t(x)$得到$F_{2t}(x)$以及$F_{2t+1}(x)$。

倍增即可得到最后的$F_n(x)$。复杂度$O(k \log n \log k)$。

【参考程序】

```
// 陈栉旷
#include <bits/stdc++.h>

template <class T>
inline void Swap(T &a, T &b) {T t = a; a = b; b = t;}

const int N = 1e5 + 5, ZZQ = 998244353;

int n, tl, k, fac[N], inv[N], ff = 1, tot, rev[N], yg[N], res[N], a[N],
tmp[N], ans;

int qpow(int a, int b)
{
    int res = 1;
    while (b)
```

```
    {
        if (b & 1) res = 1ll * res * a % ZZQ;
        a = 1ll * a * a % ZZQ;
        b >>= 1;
    }
    return res;
}

void FFT(int n, int *a, int op)
{
    for (int i = 0; i< n; i++) if (i< rev[i]) Swap(a[i], a[rev[i]]);
    yg[n] = qpow(3, (ZZQ - 1) / n * ((n + op) % n));
    for (int i = n >> 1; i; i>>= 1) yg[i] = 1ll * yg[i<< 1] * yg[i<< 1] %
ZZQ;
    for (int k = 1; k < n; k <<= 1)
    {
        int x = yg[k << 1];
        for (int i = 0; i< n; i += k << 1)
        {
            int w = 1;
            for (int j = 0; j < k; j++)
            {
                int u = a[i + j], v = 1ll * w * a[i + j + k] % ZZQ;
                a[i + j] = (u + v) % ZZQ;
                a[i + j + k] = (u - v + ZZQ) % ZZQ;
                w = 1ll * w * x % ZZQ;
            }
        }
    }
}

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

    std::cin>> n >> k;
    fac[0] = inv[0] = inv[1] = 1;
    for (int i = 1; i<= k; i++) fac[i] = 1ll * fac[i - 1] * i % ZZQ;
    for (int i = 2; i<= k; i++)
        inv[i] = 1ll * (ZZQ - ZZQ / i) * inv[ZZQ % i] % ZZQ;
```

```
    for (int i = 2; i<= k; i++) inv[i] = 1ll * inv[i] * inv[i - 1] % ZZQ;
    while (ff <= (k << 1)) ff <<= 1, tot++;
    for (int i = 0; i< ff; i++)
        rev[i] = (rev[i>> 1] >> 1) | ((i& 1) << tot - 1);
    res[0] = 1;
    int gg = qpow(ff, ZZQ - 2);
    for (int T = 14; T >= 0; T--)
    {
        int p2 = 1, delta = qpow(2, tl);
        for (int i = 0; i<= k; i++)
        {
            a[i] = 1ll * res[i] * p2 % ZZQ;
            p2 = 1ll * p2 * delta % ZZQ;
        }
        for (int i = k + 1; i< ff; i++) res[i] = a[i] = 0;
        FFT(ff, res, 1); FFT(ff, a, 1);
        for (int i = 0; i< ff; i++) res[i] = 1ll * res[i] * a[i] % ZZQ;
        FFT(ff, res, -1);
        for (int i = 0; i< ff; i++) res[i] = 1ll * res[i] * gg % ZZQ;
        tl<<= 1;
        if ((n >> T) & 1)
        {
            p2 = 1;
            for (int i = 0; i<= k; i++)
            {
                res[i] = 1ll * res[i] * p2 % ZZQ;
                p2 = (p2 << 1) % ZZQ;
                a[i] = i ? inv[i] : 0;
            }
            for (int i = k + 1; i< ff; i++) res[i] = a[i] = 0;
            FFT(ff, res, 1); FFT(ff, a, 1);
            for (int i = 0; i< ff; i++) res[i] = 1ll * res[i] * a[i] % ZZQ;
            FFT(ff, res, -1);
            for (int i = 0; i< ff; i++) res[i] = 1ll * res[i] * gg % ZZQ;
            tl++;
        }
    }
    for (int i = 0; i<= k; i++)
        ans = (1ll * inv[k - i] * res[i] % ZZQ * fac[k] + ans) % ZZQ;
    return std::cout<<ans<< std::endl, 0;
}
```

子集计数(acp,1s,512MB)

先考虑一个询问怎么做，加入新的元素后，记编号为 i 的物品出现次数为 cnt[i]。

对于编号为 i 的物品，我们用一个多项式 $\sum_{j=0}^{cnt[i]} x^j$ 来表示，则不同的 k 元集个数就是所有多项式相乘后 k 次项的系数，可以用分治 NTT 实现，时间复杂度 $O(n\log^2 n)$ 。

考虑多个询问怎么做，我们先用分治 NTT 算出原来集合的多项式，设为 P，记原来编号为 i 的物品的出现次数为 cnt[i]。则加入一个编号为 i 的物品后得到的多项式为：

$$P\cdot\frac{\sum_{j=0}^{cnt[i]+1} x^j}{\sum_{j=0}^{cnt[i]} x^j}$$

由等比数列求和公式得 $\sum_{i=0}^{m} x^i = \frac{1-x^{m+1}}{1-x}$ ，上式可化为：

$$P\cdot\frac{1-x^{cnt[i]+2}}{1-x^{cnt[i]+1}}$$

由于所乘的多项式只有两项，显然 $P\cdot(1-x^{cnt[i]+2})$ 可以在 $O(n)$ 的时间内模拟得到，记得到的多项式为 P’。考虑 $\frac{P'}{1-x^{cnt[i]+1}}$ 实际上就是 $P'\cdot(1-x^{cnt[i]+1})$ 的逆运算，我们把模拟的过程反过来执行一遍即可。

显然如果加入的物品原来的 cnt 相同，最后得到的多项式也是相同的。我们预处理出所有不同 cnt 加入一个物品后得到的多项式。考虑所要预处理的多项式个数，我们构造一个极端情况，使得需要预处理的多项式个数尽量多，记预处理的多项式个数为 S，则

$$\sum_{i=1}^{S} i = \frac{S(S+1)}{2} \le n$$

所以 S 只有 $O(\sqrt{n})$ 级别，我们可以预处理出所有多项式后直接回答询问。

时间复杂度 $O(n\log^2 n + n\sqrt{n} + q)$ 。

【参考程序】

```
//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;
}

template <class T>
inline void put(T x)
{
    if (x > 9) put(x / 10);
    putchar(x % 10 + 48);
}

using std::vector;
const int N = 1e5 + 5;
const int M = 2e5 + 5;
const int mod = 1051721729;
const int g = 6;
const int inv_g = 175286955;
int cnt[N], b[N], p[N], pos[N], ans[N], rev[M << 1];
int m, n, pm, Q, T = 1;

inline int quick_pow(int x, int k)
{
    int res = 1;
    while (k)
    {
        (k & 1) ? res = 1ll * res * x % mod : 0;
        x = 1ll * x * x % mod; k >>= 1;
    }
    return res;
}

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

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

```
struct point
{
    int k, t;

    point() {}
    point(int K, int T):
        k(K), t(T) {}
};
vector<point> v[N];

#define sL s << 1
#define sR s << 1 | 1

struct poly
{
    vector<int> f;
    int fn;

    inline void Init(int nn)
    {
        fn = nn;
        f.clear();
        for (int i = 0; i <= fn; ++i)
            f.push_back(1);
    }

    inline void NTT(char opt)
    {
        int now_g = opt == 1 ? g : inv_g;
        for (int i = 0; i < fn; ++i)
            if (i < rev[i])
                std::swap(f[i], f[rev[i]]);
        for (int k = 1, w, res, u, v; k < fn; k <<= 1)
        {
            w = quick_pow(now_g, (mod - 1) / (k << 1));
            for (int i = 0; i < fn; i += k << 1)
            {
                res = 1;
                for (int j = 0; j < k; ++j)
                {
                    u = f[i + j];
                    v = 1ll * res * f[i + j + k] % mod;
                    f[i + j] = f[i + j + k] = u;
                    add_up(f[i + j], v);
```

```
                add_down(f[i + j + k], v);
                res = 1ll * res * w % mod;
            }
        }
    }
}

inline void operator *= (poly a)
{
    int tot = fn + a.fn, k = 0, gn;
    for (gn = 1; gn <= tot; gn <<= 1)
        ++k; --k;
    for (int i = gn - fn; i >= 1; --i)
        f.push_back(0);
    for (int i = gn - a.fn; i >= 1; --i)
        a.f.push_back(0);
    fn = a.fn = gn;

    for (int i = 1; i < gn; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << k);
    NTT(1); a.NTT(1);
    for (int i = 0; i < gn; ++i)
        f[i] = 1ll * f[i] * a.f[i] % mod;
    NTT(-1);

    for (int i = 0, inv = quick_pow(gn, mod - 2); i <= tot; ++i)
        f[i] = 1ll * f[i] * inv % mod;
    fn = tot;
    for (int i = gn - tot; i >= 1; --i)
        f.pop_back();
}
}tr[M], F, tF, G;

inline void solve(int s, int l, int r)
{ //分治 NTT
    if (l == r)
        return tr[s].Init(p[l]);
    int mid = l + r >> 1;
    solve(s, l, mid);
    int rc = ++T;
    solve(rc, mid + 1, r);
    tr[s] *= tr[rc];
}
```

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

    read(n);
    for (int i = 1, x; i <= n; ++i)
        read(x), ++cnt[x];
    for (int i = 1; i <= n; ++i)
    {
        b[++m] = cnt[i];
        if (cnt[i]) p[++pm] = cnt[i];
    }
    solve(1, 1, pm);
    F = tr[1];

    std::sort(b + 1, b + m + 1);
    m = std::unique(b + 1, b + m + 1) - b - 1;
    for (int i = 1; i <= n; ++i)
        pos[i] = std::lower_bound(b + 1, b + m + 1, cnt[i]) - b;

    read(Q);
    for (int i = 1, x, K; i <= Q; ++i)
    {
        read(x); read(K);
        v[pos[x]].push_back(point(K, i));
    }

    ++n;
    F.f.push_back(0);
    for (int i = 0; i <= n; ++i)
        tF.f.push_back(0);
    for (int i = 1; i <= m; ++i) //模拟乘除多项式
        if (!v[i].empty())
        {
            G = F;
            for (int j = 0; j <= n; ++j)
                if (j + b[i] + 2 <= n)
                    add_up(G.f[j + b[i] + 2], mod - F.f[j]);
            for (int j = 0; j <= n; ++j)
            {
                tF.f[j] = G.f[j];
                    if (j - b[i] - 1 >= 0)
                add_up(tF.f[j], tF.f[j - b[i] - 1]);
```

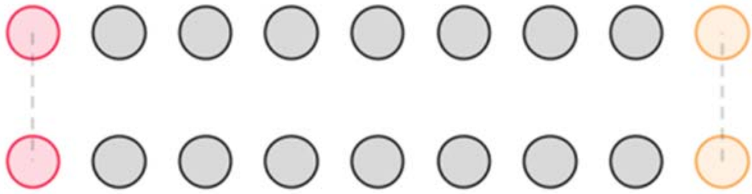
```
            }
            for (int j = 0, jm = v[i].size(); j < jm; ++j)
                ans[v[i][j].t] = tF.f[v[i][j].k];
        }
    for (int i = 1; i <= Q; ++i)
        put(ans[i]), putchar('\n');

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

花钟计数

首先解决整个圆比较困难，我们考虑一些更简单的情况。考虑一个长度为 i 的圆弧（包含 i 朵花的线段）以及这些花对面的花，假设它们被两组相对的颜色相同的花包在中间。

我们计算这些花对答案的贡献，$f_0(i)$——换句话说就是，当我们只考虑给线段染色时总的美丽度。



定义状态 $g(i)$，表示只用距离为 1 和 2 的相同颜色花的方案数。

$$g(0) = 1, g(1) = 0, g(i) = g(i-2) + g(i-4)$$

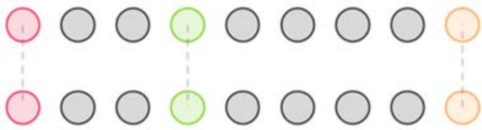
然后我们要算 $f_0(i)$。

第一种情况：不存在相对的颜色相同的花。方案数为 $g(i)$ ，总的美丽度为

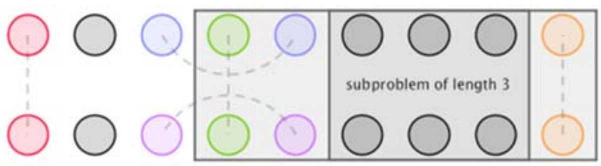
$$g(i) \times i^2$$

第二种情况：至少有一对相对的颜色相同的花。假设第一对这种花是第 j 朵花（花从0到$i-1$标号）。这又会分成两种情况：

(a) 没有一对距离为 2 的花跨过 j。在这种情况下，产生了一个长度为 $i-j-1$ 的子问题，总的美丽度为 $g(j) \times j^2 \times f_0(i-j-1)$。



(b) 有一对距离为 2 的花跨过 j。这种情况下，一个新的子问题产生了——一个长度为 $i-j-2$ 的圆弧以及它们对面的花，被一边是一对相同颜色的花，另一边是一对相同颜色的花以及一对已经匹配好的花包起来。设这个子问题的美丽度为 f_1，这种情况下的美丽度之和为 $g(j-1) \times j^2 \times f_1(i-j-2)$。



求和并简化之和，我们得到 f_0：

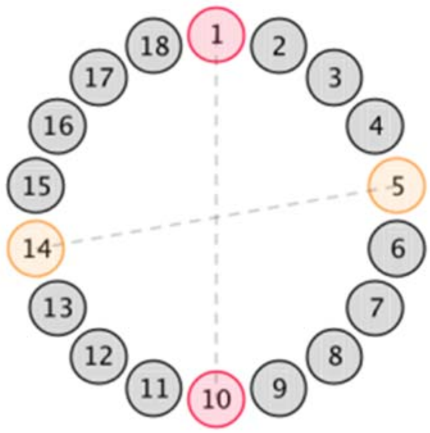
$$f_0(i) = g(i) \times i^2 + \sum_{j=0}^{i-1} g(j) \times j^2 \times f_0(i-j-1) + \sum_{j=0}^{i-3} g(j) \times (j+1)^2 \times f_1(i-j-3)$$

类似于求 f_0 的过程，我们可以得到 f_1：

$$f_1(i) = g(i) \times (i+1)^2 + \sum_{j=0}^{i-1} g(j) \times (j+1)^2 \times f_0(i-j-1) + \sum_{j=0}^{i-3} g(j) \times (j+2)^2 \times f_1(i-j-3)$$

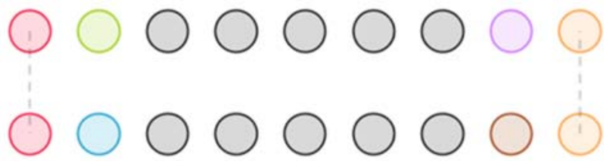
现在我们回到环上，对于一个环，我们假设一对相对的花标号为 1 和 n，这可以通过旋转来得到。

但是我们不知道在没有重复的情况下可以旋转几次，我们假设顺时针第二对相对的花的标号为 i，在 $[2, i-1]$ 中不存在相对的花



有可能有距离为 2 的花跨过 1 和 i，我们考虑所有 4 种情况。

只有一种之前没有解决，那就是一段圆弧的两边都带上了一对匹配好的点，设这种情况的美丽度为 f_2。



跟之前类似的，我们可以得到：

$$f_2(i) = g(i) \times (i+2)^2 + \sum_{j=0}^{i-1} g(j) \times (j+1)^2 \times f_1(i-j-1) + \sum_{j=0}^{i-3} g(j) \times (j+2)^2 \times f_2(i-j-3)$$

计算答案的时候枚举顺时针第二对相对的花的标号 i，然后我们就能在 $O(n^2)$ 的时间复杂度解决这个问题了。

然后注意到所有这些 DP 都是卷积的形式，那么我们可以用分治 FFT 在$O(n\log^2 n)$时间求出所有的值。

【参考程序】

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

const int mod = 998244353;
```

```
const int inv3 = (mod + 1) / 3;
const int N = 5e4 + 5;
const int M = 2e5 + 5;
int ex[N], f0[N], g0[N], f1[N], g1[N];
int tw[M], rev[M], x[M], y[M], z[M], a0[M], b0[M], a1[M], b1[M];
int n, ans;

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;
}

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 NTT(int *f, int fm, int opt)
{
    int g = opt == 1 ? 3 : inv3;
    for (int i = 0; i < fm; ++i)
        if (i < rev[i])
            std::swap(f[i], f[rev[i]]);
    for (int k = 1; k < fm; k <<= 1)
    {
        int w = quick_pow(g, (mod - 1) / (k << 1));
        tw[0] = 1;
        for (int j = 1; j < k; ++j)
            tw[j] = 1ll * tw[j - 1] * w % mod;
```

```
        for (int i = 0; i < fm; i += k << 1)
            for (int j = 0, *f1 = f + i, *f2 = f + i + k; j < k; ++j,
++f1, ++f2)
            {
                int u = *f1,
                    v = 1ll * tw[j] * (*f2) % mod;
                *f1 = *f2 = u;
                add(*f1, v);
                dec(*f2, v);
            }
    }
    if (opt == -1)
    {
        for (int i = 0, inv = quick_pow(fm, mod - 2); i < fm; ++i)
            f[i] = 1ll * f[i] * inv % mod;
    }
}

inline void solve(int l, int r)
{
    if (l == r)
        return ;
    int mid = l + r >> 1;
    solve(l, mid);
    for (int i = l; i <= r; ++i)
    {
        int t = i - l + 1, tmp = 1ll * (t - 1) * (t - 1) % mod;
        x[t] = 1ll * tmp * ex[t - 1] % mod;
        y[t] = t > 1 ? 1ll * tmp * ex[t - 2] % mod : 0;
        z[t] = t > 2 ? 1ll * tmp * ex[t - 3] % mod : 0;
    }
    for (int i = l; i <= mid; ++i)
    {
        a0[i - l] = f0[i];
        b0[i - l] = g0[i];
        a1[i - l] = f1[i];
        b1[i - l] = g1[i];
    }

    int tot = r - l + 1 + mid - l, fm = 1, k = -1;
    for (fm = 1; fm <= tot; fm <<= 1, ++k);
    for (int i = r - l + 2; i < fm; ++i)
        x[i] = y[i] = z[i] = 0;
    for (int i = mid - l + 1; i < fm; ++i)
```

```
        a0[i] = b0[i] = a1[i] = b1[i] = 0;
    for (int i = 1; i < fm; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << k);

    NTT(x, fm, 1), NTT(y, fm, 1), NTT(z, fm, 1);
    NTT(a0, fm, 1), NTT(b0, fm, 1), NTT(a1, fm, 1), NTT(b1, fm, 1);
    for (int i = 0; i < fm; ++i)
    {
        int ta0 = a0[i], ta1 = a1[i],
            tb0 = b0[i], tb1 = b1[i];
        a0[i] = (1ll * ta0 * x[i] + 1ll * tb0 * y[i]) % mod;
        b0[i] = (1ll * ta0 * y[i] + 1ll * tb0 * z[i]) % mod;
        a1[i] = (1ll * ta1 * x[i] + 1ll * tb1 * y[i]) % mod;
        b1[i] = (1ll * ta1 * y[i] + 1ll * tb1 * z[i]) % mod;
    }
    NTT(a0, fm, -1), NTT(b0, fm, -1), NTT(a1, fm, -1), NTT(b1, fm, -1);

    for (int i = mid + 1; i <= r; ++i)
    {
        add(f0[i], a0[i - l]);
        add(g0[i], b0[i - l]);
        add(f1[i], a1[i - l]);
        add(g1[i], b1[i - l]);
    }
    solve(mid + 1, r);
}

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

    scanf("%d", &n);
    if (n <= 2)
        return puts("0"), 0;
    ex[0] = ex[2] = 1;
    for (int i = 4; i <= n; ++i)
    {
        ex[i] = ex[i - 2];
        add(ex[i], ex[i - 4]);
    }
    for (int i = 1; i < n; ++i)
    {
        f0[i + 1] = 1ll * i * i % mod * ex[i] % mod;
```

```
        g0[i + 1] = 1ll * i * i % mod * ex[i - 1] % mod;
        f1[i + 1] = 1ll * i * i % mod * ex[i - 1] % mod;
        if (i > 1)
            g1[i + 1] = 1ll * i * i % mod * ex[i - 2] % mod;
    }
    if (n > 4)
        solve(2, n - 2);
    for (int j = 1; j < n - 2; ++j)
    {
        int tmp = 1ll * j * j % mod * (j + 1) % mod;
        ans = (1ll * f0[n - j - 1] * tmp % mod * ex[j]
             + 1ll * g0[n - j - 1] * tmp % mod * ex[j - 1]
             + 1ll * f1[n - j - 1] * tmp % mod * ex[j - 1] + ans) % mod;
        if (j > 1)
            ans = (1ll * g1[n - j - 1] * tmp % mod * ex[j - 2] + ans) %
mod;
    }
    ans = (1ll * (n - 1) * (n - 1) % mod * (ex[n - 1] + ex[n - 3]) % mod
* n % mod + ans) % mod;

    printf("%d\n", ans);
    fclose(stdin); fclose(stdout);
    return 0;
}
```