字符串

KMP

```
int len1, len2, nxt[MAXN], ans[MAXN];
char s1[MAXN], s2[MAXN]; //s1 文本串, s2 模式串

int main()
{
    scanf("%s%s", s1+1, s2+1);
    len1=strlen(s1+1), len2=strlen(s2+1);
    for(int i=2, p=0; i<=len2; i++)
    {
        while(p && s2[i]!=s2[p+1]) p=nxt[p];
        if(s2[i]==s2[p+1]) p++;
        nxt[i]=p;
    }
    for(int i=1, p=0; i<=len1; i++)
    {
        while(p && s1[i]!=s2[p+1]) p=nxt[p];
        if(s1[i]==s2[p+1]) p++;
        // p==len2 时, 存在一个匹配
    }
}
```

AC自动机

```
int n, tot, fail[MAXN], t[MAXN][26], num[MAXN];
char s2[MAXN], s1[MAXN];  //s1 文本串, s2 模式串

queue<int> q;
vector<int> vec[MAXN];

void insert(char *s, int id)
{
    int len=strlen(s), p=0;
    for(int i=0; i<len; i++) {
        int ch=s[i]-'a';
        if(!t[p][ch]) t[p][ch]=++tot;
        p=t[p][ch];
    }
    vec[p].push_back(id);
}

void getfail()
{
    for(int i=0; i<26; i++)
```

```cpp
        if(t[0][i]) fail[t[0][i]]=0, q.push(t[0][i]);
    while(!q.empty()) {
        int x=q.front(); q.pop();
        for(int i=0; i<26; i++) {
            if(t[x][i]) {
                fail[t[x][i]]=t[fail[x]][i];
                q.push(t[x][i]);
            } else {
                t[x][i]=t[fail[x]][i];
            }
        }
    }
}

void solve(char *s)
{
    int len=strlen(s), p=0;
    for(int i=0; i<len; i++) {
        int ch=s[i]-'a';
        p=t[p][ch];
        for(int j=p; j; j=fail[j])
            for(int e: vec[j]) num[e]++;
    }
}

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; i++) {
        scanf("%s", s2);
        insert(s2, i);
    }
    getfail();
    scanf("%s", s1);
    solve(s1);
    //num[i] 为模式串 i 在文本串中出现次数
    return 0;
}
```

后缀自动机

```cpp
void insert(int ch)
{
    /* 广义后缀自动机
    if(a[last].nxt[ch]) {
        int p=last, x=a[p].nxt[ch];
        if(a[p].len+1==a[x].len) last=x;
        else {
            int y=++tot;
            memcpy(a[y].nxt, a[x].nxt, sizeof(a[y].nxt));
            a[y].len=a[p].len+1; a[y].fa=a[x].fa;
            for(; p && a[p].nxt[ch]==x; p=a[p].fa) a[p].nxt[ch]=y;
            a[x].fa=y; last=y;
        }
        return;
    }
    */
    int cur=++tot, p=last;
    a[cur].len=a[last].len+1;
    a[cur].val=1;
    for(; p && !a[p].nxt[ch]; p=a[p].fa) a[p].nxt[ch]=cur;
    if(!p) a[cur].fa=1;
    else {
        int x=a[p].nxt[ch];
        if(a[p].len+1==a[x].len) a[cur].fa=x;
        else {
            int y=++tot;
            memcpy(a[y].nxt, a[x].nxt, sizeof(a[y].nxt));
            a[y].len=a[p].len+1; a[y].fa=a[x].fa;
            for(; p && a[p].nxt[ch]==x; p=a[p].fa) a[p].nxt[ch]=y;
            a[x].fa=a[cur].fa=y;
        }
    }
    last=cur;
}
```

线性代数

高斯消元

```
int n;
double a[MAXN][MAXN];

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; ++i)
        for(int j=1; j<=n+1; ++j) scanf("%lf", &a[i][j]);
    int c=1, r=1;
    for(; c<=n; ++c) {
        int pos=0;
        for(int i=r; i<=n; ++i)
            if(a[i][c]!=0) pos=i;
        if(pos==0) continue;
        for(int i=c; i<=n+1; ++i) swap(a[r][i], a[pos][i]);
        for(int i=n+1; i>=c; --i) a[r][i]/=a[r][c];
        for(int i=1; i<=n; ++i) {
            if(r==i) continue;
            for(int j=n+1; j>=c; --j) a[i][j]-=a[i][c]*a[r][j];
        }
        r++;
    }
    if(r<=n) {
        printf("No Solution\n");
        return 0;
    }
    for(int i=1; i<=n; ++i) printf("%.2lf\n", a[i][n+1]);
    return 0;
}
```

线性基

```
int n;
LL base[66];

void insert(LL *base, LL x)
{
    for(int i=60; i>=0; --i)
        if((x>>i)&1) {
            if(!base[i]) {
                base[i]=x;
                return;
            }
            x^=base[i];
        }
}

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; ++i) {
        LL temp;
        scanf("%lld", &temp);
        insert(base, temp);
    }
    return 0;
}
```

矩阵树定理

度数矩阵 $D(G)$　邻接矩阵 $A(G)$

Kirchhoff 矩阵 $L(G)=D(G)-A(G)$

生成树个数 $t(G)=detL^{n-1}(G)$

有向图中 $t^{root}(G,k)=detL^{out}(G,k)$

$L^{n-1}(G)$ 为 $L(G)$ 的任意 $n-1$ 阶主子式

LGV 引理

$\omega(P)$ 表示 P 这条路径上所有边的边权之积。（路径计数时，可以将边权都设为 1）（事实上，边权可以为生成函数）

$e(u,v)$ 表示 u 到 v 的 **每一条** 路径 P 的 $\omega(P)$ 之和，即 $e(u,v)=\sum\limits_{P:u\to v}\omega(P)$。

起点集合 A，是有向无环图点集的一个子集，大小为 n。

终点集合 B，也是有向无环图点集的一个子集，大小也为 n。

一组 $A\to B$ 的不相交路径 S：S_i 是一条从 A_i 到 $B_{\sigma(S)_i}$ 的路径（$\sigma(S)$ 是一个排列），对于任何 $i\neq j$，S_i 和 S_j 没有公共顶点。

$N(\sigma)$ 表示排列 σ 的逆序对个数。

$$M = \begin{bmatrix} e(A_1,B_1) & e(A_1,B_2) & \cdots & e(A_1,B_n) \\ e(A_2,B_1) & e(A_2,B_2) & \cdots & e(A_2,B_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(A_n,B_1) & e(A_n,B_2) & \cdots & e(A_n,B_n) \end{bmatrix}$$

$$\det(M) = \sum_{S:A\to B} (-1)^{N(\sigma(S))} \prod_{i=1}^{n} \omega(S_i)$$

BEST 定理

$ec(G)$ 不同欧拉回路总数　欧拉图中 $t^{root}(G) = t^{leaf}(G)$

$ec(G) = t^{root}(G,k) \prod_{v \in V} (deg(v) - 1)!$

组合数学

拉格朗日插值

```cpp
// 给定 x 出点值，求多项式 k 处点值 (k>=0 && k<P)
// x 连续时使用 interp2
int T, n, k, ifac[MAXN];

int interp1(vi x, vi y, int k)
{
    int n=x.size(), ans=0;
    for(int i=0; i<n; ++i) {
        int up=y[i], down=1;
        for(int j=0; j<n; ++j) {
            if(i==j) continue;
            up=1LL*up*sub(k, x[j])%P;
            down=1LL*down*sub(x[i], x[j])%P;
        }
        ans=(ans+1LL*up*qpow(down))%P;
    }
    return ans;
}

int interp2(vi y, int k)
{
    int n=y.size(), ans=0;
    vi pre(n), suf(n);
    pre[0]=suf[n-1]=1;
    for(int i=0; i<n-1; ++i) pre[i+1]=1LL*pre[i]*sub(k, i)%P;
    for(int i=n-1; i>=1; --i) suf[i-1]=1LL*suf[i]*sub(k, i)%P;
    for(int i=0; i<n; ++i) {
        int up=1LL*y[i]*pre[i]%P*suf[i]%P;
        int down=1LL*ifac[i]*((n-i)&1?ifac[n-i-1]:P-ifac[n-i-1])%P;
        ans=(ans+1LL*up*down)%P;
    }
    return ans;
}

int main()
{
    init(2001);
    scanf("%d", &T);
    while(T--) {
        scanf("%d%d", &n, &k);
        vi x(n+1), y(n+1);
        for(int i=0; i<=n; ++i) scanf("%d%d", &x[i], &y[i]);
        printf("%d\n", interp1(x, y, k));
    }
    return 0;
}
```

快速傅里叶变换

```
void fft(vc<C> &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vc<C> rt(2, 1);
    static vc<complex<long double>> R(2, 1);
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n), rt.resize(n);
        auto x = polar(1.L, acos(-1.L) / k);
        for (int i = k; i < k * 2; ++i)
            rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vc<int> rev(n);
    for (int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for (int i = 0 ; i < n; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += k * 2) {
            auto it1 = &a[i], it2 = it1 + k;
            for (int j = 0; j < k; ++j, ++it1, ++it2) {
                auto x = (double *)&rt[j + k], y = (double *)it2;
                C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
                *it2 = *it1 - z;
                *it1 += z;
            }
        }
}

vd conv(vd &a, vd&b) {
    if (a.empty() || b.empty()) return vd();
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res) - 1), n = 1 << L;
    vc<C> in(n), out(n);
    copy(all(a), begin(in));
    for (int i = 0; i < sz(b); ++i) in[i].imag(b[i]);
    fft(in);
    for (C &x: in) x *= x;
    for (int i = 0; i < n; ++i) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    for (int i = 0; i < sz(res); ++i) res[i] = imag(out[i]) / (n * 4);
    return res;
}

int n, m;

int main()
{
    scanf("%d%d", &n, &m);
    vd a(n+1), b(m+1);
    for(int i=0; i<=n; ++i) scanf("%lf", &a[i]);
    for(int i=0; i<=m; ++i) scanf("%lf", &b[i]);
    a=conv(a, b);
    for(double e: a) printf("%d ", int(e+0.5));
}
```

多项式运算

```
template<class T> using vc = vector<T>;
using ll = int64_t;
constexpr ll md = 998244353, root = 62, LIM = 1 << 18;
struct Mod {
  ll x;
  Mod(ll x = 0): x(x) {}
  Mod operator+(Mod b) {ll y=x+b.x;return y<md ? y : y - md; }
  Mod operator-(Mod b) { return x - b.x + (x < b.x ? md : 0); }
  Mod operator*(Mod b) { return x * b.x % md; }
  void operator += (Mod b) { x += b.x; x < md ?: x -= md; }
  void operator *= (Mod b) { (x *= b.x) %= md; }
  void operator -= (Mod b) { x -= b.x; -x < 0 ?: x += md; }
};
Mod qpow(Mod b, ll e) {
    Mod res = 1;
    for (; e; b *= b, e /= 2)
        if (e & 1) res *= b;
    return res;
}

vc<Mod> inv(LIM), fac(LIM), ifac(LIM);

void init()
{
    inv[0]=inv[1]=ifac[0]=fac[0]=1;
    for(int i=2; i<LIM; ++i) inv[i]=inv[md%i]*(md-md/i);
    for(int i=1; i<LIM; ++i) {
        ifac[i]=ifac[i-1]*inv[i];
        fac[i]=fac[i-1]*i;
    }
}

void ntt(vc<Mod> &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vc<Mod> rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, ++s) {
        rt.resize(n);
        array<Mod, 2> z{1, qpow(root, md >> s)};
        for (int i = k; i < k * 2; ++i)
```

```cpp
            rt[i] = rt[i / 2] * z[i & 1];
    }
    vc<int> rev(n);
    for (int i = 0; i < n; ++i)
        rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for (int i = 0; i < n; ++i)
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0 ; i < n; i += k * 2) {
        auto it1 = &a[i], it2 = it1 + k;
        for (int j = 0; j < k; ++j, ++it1, ++it2) {
            Mod z = rt[j + k] * *it2;
            *it2 = *it1 - z, *it1 += z;
        }
    }
}

vc<Mod> conv(vc<Mod> a, vc<Mod> b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, n = 1 << (32 - __builtin_clz(s - 1));
    Mod iv = md - (md - 1) / n;
    vc<Mod> out(n);
    a.resize(n), b.resize(n);
    ntt(a), ntt(b);
    for (int i = 0; i < n; ++i)
        out[-i & (n - 1)] = a[i] * b[i] * iv;
    ntt(out);
    return {out.begin(), out.begin() + s};
}

void invIter (vc<Mod> &a, vc<Mod> &in, vc<Mod> &b) {
    int n = sz(in);
    vc<Mod> out(n);
    copy(a.begin(), a.begin() + min(sz(a), n), out.begin());
    auto conv = [&] {
        ntt(out);
        for (int i = 0; i < n; ++i) out[i] *= in[i];
        ntt(out), reverse(out.begin() + 1, out.end());
    };
    conv(), fill(out.begin(), out.begin() + sz(b), 0), conv();
    b.resize(n);
    Mod iv = md - (md - 1) / n; iv *= iv;
    for (int i = n / 2; i < n; ++i)
        b[i] = out[i].x ? iv * (md - out[i].x) : 0;
}

vc<Mod> polyInv (vc<Mod> a) {
    if (a.empty()) return {};
    vc<Mod> b{qpow(a[0], md - 2)};
    b.reserve(sz(a));
    while (sz(b) < sz(a)) {
      vc<Mod> in(sz(b) * 2);
      copy(all(b), in.begin()), ntt(in);
      invIter(a, in, b);
    }
    return {b.begin(), b.begin() + sz(a)};
}

vc<Mod> polyMod (vc<Mod> a, vc<Mod> b) {
    if (sz(a) < sz(b)) return a;
    int n = sz(a) - sz(b) + 1;
    vc<Mod> da(a.rbegin(), a.rend()), db(b.rbegin(), b.rend());
    da.resize(n), db.resize(n);
    da = conv(da, polyInv(db));
    da.resize(n), reverse(all(da));
    auto c = conv(da, b);
    a.resize(sz(b) - 1);
    for (int i = 0; i < sz(a); ++i) a[i] -= c[i];
    return a;
}

vc<Mod> deri (vc<Mod> a) {
    for (int i = 1; i < sz(a); ++i) a[i - 1] = a[i] * i;
    a.pop_back();
    return a;
}

// initialize array inv
vc<Mod> inte (vc<Mod> a) {
    for (int i = sz(a) - 1; i >= 1; --i) a[i] = a[i - 1] * inv[i];
    a[0] = 0;
    return a;
}

vc<Mod> polyLn (vc<Mod> &a) {
    if (a.empty()) return {};
    int n = 1 << (32 - __builtin_clz(2 * sz(a) - 2));
    Mod iv = md - (md - 1) / n;
    vc<Mod> b = polyInv(a), c = deri(a);
    b.resize(n), c.resize(n);
    ntt(b), ntt(c);
    for (int i = 0; i < n; ++i) b[i] = b[i] * c[i] * iv;
    ntt(b), reverse(b.begin() + 1, b.end());
    b = inte(b);
    return {b.begin(), b.begin() + sz(a)};
}

vc<Mod> polyExp (vc<Mod> &a) {
```

```cpp
    if (a.empty()) return {};
    vc<Mod> b{1}, ib{1};
    b.reserve(sz(a)), ib.reserve(sz(a));
    auto conv = [&](vc<Mod> &a, vc<Mod> &b) {
        ntt(a);
        for (int i = 0; i < sz(a); ++i) a[i] *= b[i];
        ntt(a), reverse(a.begin() + 1, a.end());
    };
    while (sz(b) < sz(a)) {
        int h = sz(b), n = h * 2;
        Mod iv = md - (md - 1) / n;
        vc<Mod> db(n), dib(n), A(deri(b)), B(n);
        copy(all(ib), dib.begin()), ntt(dib);
        copy(all(b), db.begin()), ntt(db);
        A.resize(n), conv(A, dib);
        for (int i = 0; i < n; ++i) B[i] = db[i] * dib[i];
        ntt(B), reverse(B.begin() + 1, B.end());
        fill(B.begin(), B.begin() + h, 0);
        vc<Mod> da(deri(vc<Mod>(a.begin(), a.begin() + h)));
        da.resize(n), ntt(da), conv(B, da);
        for (int i = min(n, sz(a)) - 1; i >= h; --i)
            A[i] = (A[i - 1] - B[i - 1] * iv) * inv[i] * iv - a[i];
        fill(A.begin(), A.begin() + h, 0), conv(A, db);
        b.resize(n);
        for (int i = h; i < n; ++i)
            b[i] = A[i].x ? iv * (md - A[i].x) : 0;
        if (sz(b) < sz(a)) invIter(b, dib, ib);
    }
    return {b.begin(), b.begin() + sz(a)};
}

vc<Mod> polyPow (vc<Mod> &a, ll k) {
    vc<Mod> b = polyLn(a);
    for (Mod &e: b) e *= k;
    return polyExp(b);
}
```

分治 FFT

```cpp
// f[0]=1 且满足递推关系 f[n]=\sum_i^n g[i]*f[n-i]
int n;
vc<Mod> f, g;

void divide(int l, int r)
{
    if(l==r) return;
    int mid=(l+r)>>1;
    divide(l, mid);
    vc<Mod> a{f.begin()+l, f.begin()+mid+1};
    vc<Mod> b{g.begin(), g.begin()+r-l+1};
    a=conv(a, b);
    for(int i=mid+1; i<=r; ++i) f[i]+=a[i-l];
    divide(mid+1, r);
}
```

分治乘法

```
// 求 vec 内多所有项式的卷积
vc<poly> vec;

poly solve(int l, int r)
{
    if(l==r) return vec[l];
    /* 找重心分治
    int mid=l, sum1=0, sum2=sz(vec[mid]);
    for(int i=l; i<=r; ++i) sum1+=sz(vec[l]);
    while(mid+1<r && sum2+sz(vec[mid+1])<=sum1/2) sum2+=sz(vec[++mid]);
    */
    int mid=(l+r)/2;
    return conv(solve(l, mid), solve(mid+1, r));
}

int main()
{
    // 读入个多项式
    // 中点分治需随机打乱
    vc<Mod> f = solve(0, vec.size()-1);
}
```

线性递推

```
// f[0]=P-1 且有递推关系 a[n]=\sum_i^k f[i]*a[n-i]
// 求 a[m]
int recurrence(vc<Mod> f, vc<Mod> a, ll m)
{
    int k=a.size(), n=1<<(32-__builtin_clz(2*k-2));
    vc<Mod> g=polyInv(vc<Mod>(f.begin(), f.begin()+k-1));
    reverse(all(f));
    g.resize(n), f.resize(n);
    ntt(g), ntt(f);
    auto combine = [&](vc<Mod> a, vc<Mod> b) -> vc<Mod> {
        Mod iv = md - (md - 1) / n;
        vc<Mod> c(n), d(n);
        a=conv(a, b);
        copy(a.rbegin(), a.rbegin()+k-1, c.begin());
        ntt(c);
        for (int i=0; i<n; ++i) d[-i&(n-1)]=c[i]*g[i]*iv;
```

```
            ntt(d);
            copy(d.rend()-k+1, d.rend(), c.begin());
            fill(c.begin()+k-1, c.end(), 0);
            ntt(c);
            for (int i=0; i<n; ++i) d[-i&(n-1)]=c[i]*f[i]*iv;
            ntt(d);
            for(int i=0; i<k; ++i) a[i]-=d[i];
            return {a.begin(), a.begin()+k};
        };
        vc<Mod> b(k), c(k);
        b[0]=1, c[1]=1;
        for(; m; m>>=1) {
            if(m&1) b=combine(b, c);
            c=combine(c, c);
        }
        Mod ans;
        for(int i=0; i<k; ++i) ans+=a[i]*b[i];
        return ans.x;
    }
```

单位根反演

$\omega_a^b = g^{(P-1)\cdot b/a}$

$[n|a] = \frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{ak}$

$[a \equiv b \mod n] = \frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{ak}\omega_n^{-bk}$

第 K 大反演

$k^{th}max(s) = \sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}min(T)$

Burnside 引理

l 等价类个数

G 置换群

$c(p)$ 置换 p 中不动点个数

$l = \frac{1}{G}\sum_{p\in G}c(p)$

数论

线性筛

```
int n, t, cnt, pri[MAXN], vis[MAXN];
int mu[MAXN], phi[MAXN]; //莫比乌斯函数和欧拉函数

void sieve(int lim)
{
    //phi[i]=mu[i]=1;
    for(int i=2; i<=lim; ++i) {
        if(!vis[i]) pri[++cnt]=i; // phi[i]=i-1, mu[i]=-1;
        for(int j=1; j<=cnt && pri[j]*i<=lim; ++j) {
            vis[i*pri[j]]=1;
            if(i%pri[j]==0) break;
            /*
            if(i%pri[j]==0) {
                mu[pri[j]*i]=0;
                phi[i*pri[j]]=pri[j]*phi[i];
                break;
            } else {
                mu[pri[j]*i]=-mu[i];
                phi[i*pri[j]]=(pri[j]-1)*phi[i];
            }
            */
        }
    }
}
```

质因数分解

```
const LL base[]={2, 325, 9375, 28178, 450775, 9780504, 1795265022};

int T;
LL n;
map<LL, int> mp;

LL qpow(LL x, LL y, LL p)
{
    LL num=1;
    while(y) {
        if(y&1) num=(I128)num*x%p;
        y>>=1, x=(I128)x*x%p;
    }
    return num;
}

bool MR(LL x)
```

```
{
    if(x<=2 || !(x%2)) return x==2;
    LL d=x-1, r=0;
    while(!(d%2)) d>>=1, r++;
    for(LL e: base) {
        LL v=qpow(e, d, x);
        if(v<=1 || v==x-1) continue;
        for(int i=0; i<r-1; ++i) {
            v=(I128)v*v%x;
            if(v==x-1 || v==1) break;
        }
        if(v!=x-1) return false;
    }
    return true;
}

LL PR(LL x)
{
    LL l=0, r=0, val=2, tmp;
    auto f=[x](LL y) {return ((I128)y*y+1)%x;};
    for(int i=0; ; ++i) {
        if(!(i%60) && __gcd(val, x)>1) break;
        if(l==r) l=rand()%(x-1)+1, r=f(l);
        if(tmp=(I128)val*abs(r-l)%x) val=tmp;
        l=f(l), r=f(f(r));
    }
    return __gcd(val, x);
}

void find(LL x, int num)
{
    if(x<=1) return;
    if(MR(x)) {
        mp[x]+=num;
        return;
    }
    LL y=x, cnt=0;
    while(y==x) y=PR(x);
    while(x%y==0) x/=y, cnt++;
    find(x, num), find(y, cnt*num);
}

int main()
{
    srand(time(0));
    scanf("%d", &T);
    while(T--) {
        mp.clear();
        scanf("%lld", &n);
        find(n, 1); //指数形式分解质因数
        for(auto e: mp) printf("%lld %d\n", e.first, e.second);
    }
    return 0;
}
```

BSGS

```
int a, b, x, p;

unordered_map<int, int> mp;

int qpow(int x, int y)
{
    int num=1;
    while(y) {
        if(y&1) num=1LL*num*x%p;
        x=1LL*x*x%p; y>>=1;
    }
    return num;
}

int bsgs(int a, int b)
{
    if(a%p==0) return (b==0)?1:-1;
    mp.clear();
    int siz=(int)(sqrt(p)+0.5), tmp=1LL*a*b%p;
    for(int i=1; i<=siz; ++i) {
        mp[tmp]=i;
        tmp=1LL*tmp*a%p;
    }
    int base=qpow(a, siz);
    tmp=1;
    for(int i=1; i<=siz; ++i) {
        tmp=1LL*tmp*base%p;
        if(mp[tmp]) return i*siz-mp[tmp];
    }
    return -1;
}

int main()
{
    scanf("%d%d%d", &a, &b, &p);
    x=bsgs(a, b); //a^x = b mod p
    if(x==-1) printf("No solution\n");
    else printf("%d\n", x);
}
```

杜教筛

```
int T, n, tot, vis[MAXN], pri[MAXN];
LL mu[MAXN], phi[MAXN];

unordered_map<int, LL> muf, phif;

void init()
{
    mu[1]=phi[1]=1;
    for(int i=2; i<=5e6; ++i)
    {
        if(!vis[i]) pri[++tot]=i, phi[i]=i-1, mu[i]=-1;
        for(int j=1; j<=tot && pri[j]*i<=5e6; ++j)
        {
            vis[pri[j]*i]=1;
            if(i%pri[j]==0)
            {
                phi[i*pri[j]]=phi[i]*pri[j];
                mu[i*pri[j]]=0;
                break;
            }
            phi[i*pri[j]]=phi[i]*(pri[j]-1);
            mu[i*pri[j]]=-mu[i];
        }
    }
    for(int i=1; i<=5e6; ++i) phi[i]+=phi[i-1], mu[i]+=mu[i-1];
}

LL mus(int x)
{
    if(x<=5e6) return mu[x];
    if(muf[x]) return muf[x];
    LL sum=1;
    for(int l=2, r; l<=x; l=r+1)
    {
        r=x/(x/l);
        sum-=1LL*(r-l+1)*mus(x/l);
    }
    muf[x]=sum;
    return sum;
}

LL phis(int x)
{
    if(x<=5e6) return phi[x];
    if(phif[x]) return phif[x];
    LL sum=1LL*x*(x+1)/2;
    for(int l=2, r; l<=x; l=r+1)
    {
        r=x/(x/l);
        sum-=1LL*(r-l+1)*phis(x/l);
    }
    phif[x]=sum;
    return sum;
}

int main()
{
    init();
    scanf("%d", &T);
    while(T--)
    {
        scanf("%d", &n);
        printf("%lld %lld\n", phis(n), mus(n));
    }
    return 0;
}
```

Min25 筛

积性函数 $f(x)$, $f(p^k) = p^k(p^k - 1)$, 求 $\sum_i^n f(i)$

```
int LIM, INV2=500000004, INV6=166666668;
int cnt1, cnt2, vis[MAXN], pri[MAXN], g1[MAXN], g2[MAXN], h1[MAXN], h2[MAXN],
id1[MAXN], id2[MAXN];
ll n, w[MAXN];

void sieve(int lim)
{
    for(int i=2; i<=lim; ++i) {
        if(!vis[i]) {
            pri[++cnt1]=i;
            g1[cnt1]=(g1[cnt1-1]+i)%P;
            h1[cnt1]=(h1[cnt1-1]+1ll*i*i)%P;
        }
        for(int j=1; j<=cnt1 && pri[j]*i<=LIM; ++j) {
            vis[i*pri[j]]=1;
            if(i%pri[j]==0) break;
        }
    }
}

int cal(ll x, int y)
{
    if(pri[y]>=x) return 0;
    int id=(x<=LIM?id1[x]:id2[n/x]);
    int val=sub(sub(h2[id], g2[id]), sub(h1[y], g1[y]));
    for(int i=y+1; i<=cnt1; ++i) {
        ll tmp=pri[i];
        if(tmp*tmp>x) break;
        for(int k=1; tmp<=x; k++, tmp*=pri[i]) {
            ll tmp1=tmp%P; tmp1=tmp1*(tmp1-1)%P;
            val=(val+tmp1*(cal(x/tmp, i)+(k>1)))%P;
        }
    }
    return val;
}

int main()
{
    scanf("%lld", &n);
    LIM=sqrt(n);
    sieve(LIM);
    for(ll l=1, r; l<=n; l=r+1) {
        r=n/(n/l); w[++cnt2]=n/l;
        ll tmp=(n/l)%P;
        g2[cnt2]=sub(tmp*(tmp+1)%P*INV2%P, 1);
        h2[cnt2]=sub(tmp*(tmp+1)%P*(2*tmp+1)%P*INV6%P, 1);
        if(n/r<=LIM) id1[n/r]=cnt2;
        else id2[r]=cnt2;
    }
    for(int i=1; i<=cnt1; ++i) {
        ll tmp=pri[i];
        for(int j=1; j<=cnt2 && tmp*tmp<=w[j]; ++j) {
            ll id=w[j]/pri[i]; id=(id<=LIM?id1[id]:id2[n/id]);
            g2[j]=sub(g2[j], tmp*sub(g2[id], g1[i-1])%P);
            h2[j]=sub(h2[j], tmp*tmp%P*sub(h2[id], h1[i-1])%P);
        }
    }
    printf("%d\n", add(cal(n, 0), 1));
}
```

数据结构

树状数组

```
int n, t[MAXN];

void add(int x, int y) {for(; x<=n; x+=(x&-x)) t[x]+=y;} //add(x, y) 位置 x 加 y

int sum(int x) {int y=0; for(; x; x-=(x&-x)) y+=t[x]; return y;} //sum(x) 1~x 区间和
```

线段树

```
int n, m, val[MAXN*4], tag[MAXN*4];

void up(int root)
{
    val[root]=val[ls]+val[rs];
}

void down(int root, int l, int r)
{
    if(!tag[root]) return;
    tag[ls]+=tag[root];
    tag[rs]+=tag[root];
    val[ls]+=tag[root]*(mid-l+1);
    val[rs]+=tag[root]*(r-mid);
    tag[root]=0;
}

void build(int root, int l, int r)
{
    if(l==r)
    {
        scanf("%d", &val[root]);
        return ;
    }
    build(ls, l, mid);
    build(rs, mid+1, r);
    up(root);
}

void add(int root, int l, int r, int x, int y, int k)
{
    if(x>r || y<l) return;
    if(l>=x && r<=y)
    {
        val[root]+=k*(r-l+1);
        tag[root]+=k;
        return;
    }
    down(root, l, r);
    add(ls, l, mid, x, y, k);
    add(rs, mid+1, r, x, y, k);
    up(root);
}

int query(int root, int l, int r, int x, int y)
{
    if(l>y || r<x) return 0;
    if(l>=x && r<=y) return val[root];
    down(root, l, r);
    return query(ls, l, mid, x, y)+query(rs, mid+1, r, x, y);
}
```

主席树

```
int n, m, tot, a[MAXN], root[MAXN];

struct Node {int ls, rs, val;} t[MAXN*40];

void update(int &rt1, int rt2, int l, int r, int x)
{
    rt1=++tot;
    t[rt1]=t[rt2], t[rt1].val++;
    if(l==r) return;
    if(x<=mid) update(t[rt1].ls, t[rt2].ls, l, mid, x);
    else update(t[rt1].rs, t[rt2].rs, mid+1, r, x);
}

int query(int rt1, int rt2, int l, int r, int k)
{
    if(l==r) return l;
    int temp=t[t[rt2].ls].val-t[t[rt1].ls].val;
    if(temp>=k) return query(t[rt1].ls, t[rt2].ls, l, mid, k);
    else return query(t[rt1].rs, t[rt2].rs, mid+1, r, k-temp);
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i=1; i<=n; ++i)
    {
        scanf("%d", &a[i]);
            update(root[i], root[i-1], 1, n, a[i]);
    }
    for(int i=1; i<=m; i++)
    {
        //query(root[x-1], root[y], 1, n, k) 区间 x~y 第k大
    }
}
```

点分治

```
int n, m, tot, rt, f[MAXN], siz[MAXN], k[MAXN], ans[MAXN], vis[MAXN];

vector<pair<int,int>> g[MAXN];
vector<int> vec;

void find(int x, int fa)
{
    siz[x]=1; f[x]=0;
    for(auto [to, d]: g[x]) {
        if(to==fa || vis[to]) continue;
        find(to, x);
        siz[x]+=siz[to];
        f[x]=max(f[x], siz[to]);
    }
    f[x]=max(f[x], tot-siz[x]);
    if(!rt || f[x]<f[rt]) rt=x;
}

void dfs(int x, int fa, int dis)
{
    vec.push_back(dis);
    for(auto [to, d]: g[x]) {
        if(to==fa || vis[to]) continue;
        dfs(to, x, dis+d);
    }
}

void divide(int x)
{
    vis[x]=1;
    set<int> st; st.insert(0);
    for(auto [to, d]: g[x]) {
        if(vis[to]) continue;
        vec.clear();
        dfs(to, x, d);
        for(int i=1; i<=m; ++i)
            for(int e: vec)
                if(st.count(k[i]-e)) ans[i]=1;
        for(int e: vec) st.insert(e);
    }
    for(auto [to, d]: g[x]) {
        if(vis[to]) continue;
        tot=siz[to], rt=0;
        find(to, 0);
        divide(rt);
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i=1; i<n; ++i) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        g[x].push_back({y, z});
        g[y].push_back({x, z});
    }
    for(int i=1; i<=m; ++i) scanf("%d", &k[i]);
    tot=n; rt=0;
    find(1, 0);
    divide(rt);
    for(int i=1; i<=m; ++i) {
        if(ans[i]) printf("AYE\n");
        else printf("NAY\n");
    }
}
```

莫队

```
int n, q, ans[MAXN], bol[MAXN];

struct Q {int l, r, id;} a[MAXN];

bool CMP(Q x, Q y)
{
    if(bol[x.l]==bol[y.l]) {
        if(bol[x.l]&1) return x.r<y.r;
        else return x.r>y.r;
    }
    return x.l<y.l;
}

int main()
{
    scanf("%d", &n);
    int l=1, r=0, siz=sqrt(n);
    for(int i=1; i<=n; ++i) bol[i]=(i-1)/siz+1;
    scanf("%d",  &q);
    for(int i=1; i<=q; ++i) {
        scanf("%d%d", &a[i].l, &a[i].r);
        a[i].id=i;
    }
```

```cpp
    sort(a+1, a+q+1, CMP);
    for(int i=1; i<=q; ++i) {
        while(r<a[i].r) {
            r++;
            //update(r, 1);
        }
        while(r>a[i].r) {
            //update(r, -1);
            r--;
        }
        while(l<a[i].l) {
            //update(l, -1);
            l++;
        }
        while(l>a[i].l) {
            l--;
            //update(l, 1);
        }
        //ans[a[i].id]=query();
    }
    for(int i=1; i<=q; ++i) printf("%d\n", ans[i]);
    return 0;
}
```

图论

最短路

```cpp
int n, m, S, dis[MAXN], vis[MAXN];

struct Node
{
    int id, dis;
    bool friend operator < (Node x, Node y)
    {
        return x.dis>y.dis;
    }
};

vector<int> g1[MAXN], g2[MAXN];
priority_queue<Node> q;

void dijkstra()
{
    memset(dis, 0x3f, sizeof(dis));
    dis[S]=0;
    q.push(Node{S, 0});
    while(!q.empty())
    {
        int x=q.top().id; q.pop();
        if(vis[x]) continue;
        vis[x]=1;
        for(int i=0; i<g1[x].size(); ++i)
        {
            int to=g1[x][i];
            if(dis[to]>dis[x]+g2[x][i])
            {
                dis[to]=dis[x]+g2[x][i];
                q.push(Node{to, dis[to]});
            }
        }
    }
}

int main()
{
    scanf("%d%d%d", &n, &m, &S);
    for(int i=1; i<=m; ++i)
    {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        g1[x].pb(y), g2[x].pb(z);
    }
    dijkstra();
    //dis[i] 为 s 到 i 的最短路距离
    return 0;
}
```

最小生成树

```
int n, m, ans, f[MAXN];

struct Edge {int x, y, dis;} edge[MAXM];

bool CMP(Edge x, Edge y)
{
    return x.dis<y.dis;
}

int find(int x)
{
    if(f[x]!=x) f[x]=find(f[x]);
    return f[x];
}

void kruskal()
{
    for(int i=1; i<=n; ++i) f[i]=i;
    sort(edge+1, edge+m+1, CMP);
    for(int i=1; i<=m; ++i)
    {
        int fx=find(edge[i].x), fy=find(edge[i].y);
        if(fx!=fy)
        {
            ans+=edge[i].dis;
            f[fx]=fy;
        }
    }
}

int main()
{
    scanf("%d%d", &n, &m); //n 个点 m 条边
    for(int i=1; i<=m; ++i)
        scanf("%d%d%d", &edge[i].x, &edge[i].y, &edge[i].dis);
    kruskal();
    //ans 为最小生成树边权之和
    return 0;
}
```

最近公共祖先

```
int n, m, s, dep[MAXN], f[MAXN][20];

vector<int> g[MAXN];

void dfs(int x, int fa)
{
    f[x][0]=fa, dep[x]=dep[fa]+1;
    for(int i=0; i<g[x].size(); ++i)
    {
        int to=g[x][i];
        if(to==fa) continue;
        dep[to]=dep[x]+1;
        dfs(to, x);
    }
}

int lca(int x, int y)
{
    if(dep[x]>dep[y]) swap(x, y);
    for(int i=19; i>=0; --i)
        if(dep[f[y][i]]>=dep[x]) y=f[y][i];
    if(x==y) return x;
    for(int i=19; i>=0; --i)
        if(f[x][i]!=f[y][i]) x=f[x][i], y=f[y][i];
    return f[x][0];
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i=1; i<n; ++i)
    {
        int x, y;
        scanf("%d%d", &x, &y);
        g[x].push_back(y);
        g[y].push_back(x);
    }
    dfs(1, 0);
    for(int i=1; i<=19; ++i)
        for(int j=1; j<=n; ++j) f[j][i]=f[f[j][i-1]][i-1];
    //lca(x,y) 为 x,y 的最近公共祖先
    return 0;
}
```

缩点

```
int n, m, cnt, ans, tot, dfn[MAXN], low[MAXN], ins[MAXN], id[MAXN];

```

```
queue<int> q;
stack<int> sta;
vector<int> g1[MAXN], g2[MAXN];

void tarjan(int x)
{
    dfn[x]=low[x]=++cnt;
    sta.push(x);
    ins[x]=1;
    for(int i=0; i<g1[x].size(); ++i)
    {
        int to=g1[x][i];
        if(!dfn[to])
        {
            tarjan(to);
            low[x]=min(low[x], low[to]);
        }
        else if(ins[to])
            low[x]=min(low[x], dfn[to]);
    }
    if(low[x]==dfn[x])
    {
        tot++;
        while(!sta.empty() && dfn[sta.top()]>=dfn[x])
        {
            int top=sta.top(); sta.pop();
            ins[top]=0;
            id[top]=tot;
        }
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i=1; i<=m; ++i)
    {
        int x, y;
        scanf("%d%d", &x, &y);
        g1[x].push_back(y);
    }
    for(int i=1; i<=n; ++i)
        if(!dfn[i]) tarjan(i);
    //id[x] 为 x 所在强连通分量的编号
    return 0;
}
```

割点

```
int n, m, cnt, ans, dfn[MAXN], low[MAXN], val[MAXN];
vector<int> g[MAXN];

void tarjan(int x)
{
    dfn[x]=low[x]=++cnt;
    for(int i=0; i<g[x].size(); ++i)
    {
        int to=g[x][i];
        if(!dfn[to])
        {
            tarjan(to);
            low[x]=min(low[x], low[to]);
            if(low[to]>=dfn[x]) val[x]++;
        }
        else low[x]=min(low[x], dfn[to]);
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i=1; i<=m; ++i)
    {
        int x, y;
        scanf("%d%d", &x, &y);
        g[x].push_back(y);
        g[y].push_back(x);
    }
    for(int i=1; i<=n; ++i)
        if(!dfn[i])
        {
            ans++;
            tarjan(i);
            if(val[i]) val[i]--;
        }
    //val[i]>0 代表 i 为割点
    return 0;
}
```

Dinic

```
int n, m, S, T, cnt=1, head[MAXN], dis[MAXN];
LL maxflow;

struct Edge {
    int next, to;
    LL flow;
```

```cpp
} edge[MAXM*2];

queue<int> q;

inline void addedge(int from, int to, int flow)
{
    edge[++cnt].next=head[from];
    edge[cnt].to=to;
    edge[cnt].flow=flow;
    head[from]=cnt;
}

bool bfs()
{
    for(int i=1; i<=n; ++i) dis[i]=0;
    dis[S]=1;
    q.push(S);
    while(!q.empty())
    {
        int x=q.front(); q.pop();
        for(int i=head[x]; i; i=edge[i].next)
        {
            int to=edge[i].to;
            if(dis[to] || !edge[i].flow) continue;
            dis[to]=dis[x]+1;
            q.push(to);
        }
    }
    return dis[T]>0;
}

LL dfs(int x, LL flow)
{
    if(x==T) return flow;
    LL add=0;
    for(int i=head[x]; i && flow; i=edge[i].next)
    {
        int to=edge[i].to;
        if(dis[to]!=dis[x]+1 || !edge[i].flow) continue;
        LL f=dfs(to, min(edge[i].flow, flow));
        edge[i].flow-=f, edge[i^1].flow+=f;
        add+=f; flow-=f;
    }
    if(!add) dis[x]=0;
    return add;
}

int main()
{
    scanf("%d%d%d%d", &n, &m, &S, &T);
    for(int i=1; i<=m; i++)
    {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        addedge(u, v, w);
        addedge(v, u, 0);
    }
    while(bfs()) maxflow+=dfs(S, INF);
    printf("%lld\n", maxflow);
}
```

EK费用流

```
int n, m, S, T, cnt=1, head[MAXN];
int maxflow, mincost, vis[MAXN], dis[MAXN];

struct Edge {int next, to, flow, cost;} edge[MAXM*2];
struct Pre {int id, from;} pre[MAXN];

queue<int> q;

void addedge (int from, int to, int flow, int cost)
{
    edge[++cnt].next=head[from];
    edge[cnt].cost=cost;
    edge[cnt].flow=flow;
    edge[cnt].to=to;
    head[from]=cnt;
}

bool spfa()
{
    for(int i=0; i<=n; ++i) vis[i]=0, dis[i]=INF;
    vis[S]=1; dis[S]=0;
    q.push(S);
    while(!q.empty())
    {
        int x=q.front(); q.pop();
        vis[x]=0;
        for(int i=head[x]; i; i=edge[i].next)
        {
            int to=edge[i].to;
            if(dis[to]>dis[x]+edge[i].cost && edge[i].flow)
            {
                dis[to]=dis[x]+edge[i].cost;
                pre[to].from=x, pre[to].id=i;
                if(!vis[to])
                {
                    q.push(to);
                    vis[to]=1;
                }
            }
        }
    }
    return dis[T]<dis[0];
}

int main()
{
    scanf("%d%d%d%d", &n, &m, &S, &T);
    for(int i=1; i<=m; i++)
    {
        int u, v, w, f;
        scanf("%d%d%d%d", &u, &v, &w, &f);
        addedge(u, v, w, f);
        addedge(v, u, 0, -f);
    }
    maxflow=0, mincost=0;
    while(spfa())
    {
        int flow=INF;
        for(int i=T; i!=S; i=pre[i].from) flow=min(flow, edge[pre[i].id].flow);
        for(int i=T; i!=S; i=pre[i].from)
        {
            edge[pre[i].id].flow-=flow;
            edge[pre[i].id^1].flow+=flow;
        }
        maxflow+=flow;
        mincost+=dis[T]*flow;
    }
    printf("%d %d\n", maxflow, mincost);
    return 0;
}
```

集合运算

枚举子集

```cpp
int n;

int main()
{
    scanf("%d", &n);
    // 预处理
    for(int sta=1; sta<(1<<n); ++sta)
        for(int sub=sta; sub; sub=(sub-1)&sta)
        {
            // sub 为 sta 的子集
        }
    return 0;
}
```

SOS DP

```cpp
int n, f[MAXN], g[MAXN];

int main()
{
    scanf("%d", &n);
    for(int i=0; i<1<<n; ++i) scanf("%d%d", &f[i], &g[i]);
    for(int i=0; i<n; ++i)
        for(int sta=0; sta<1<<n; ++sta) {
            if((sta>>i)&1) f[sta]+=f[sta^(1<<i)]; //子集和
            if(!((sta>>i)&1)) g[sta]+=g[sta^(1<<i)]; //母集和
        }
    return 0;
}
```

快速沃尔什变换

```cpp
void or_fwt(vi &a, int op)
{
    for(int n=sz(a), step=1; step<n; step*=2)
        for(int i=0; i<n; i+=2*step) for(int j=i; j<i+step; ++j) {
            int &u=a[j], &v=a[j+step];
            tie(u, v)=op>0?MP(add(u, v), u):MP(v, sub(u, v));
        }
}

void and_fwt(vi &a, int op)
{
    for(int n=sz(a), step=1; step<n; step*=2)
        for(int i=0; i<n; i+=2*step) for(int j=i; j<i+step; ++j) {
            int &u=a[j], &v=a[j+step];
            tie(u, v)=op>0?MP(v, add(u, v)):MP(sub(v, u), u);
        }
}

void xor_fwt(vi &a, int op)
{
    for(int n=sz(a), step=1; step<n; step*=2)
        for(int i=0; i<n; i+=2*step) for(int j=i; j<i+step; ++j) {
            int &u=a[j], &v=a[j+step];
            tie(u, v)=MP(add(u, v), sub(u, v));
        }
    if(op<0) {
        int inv=qpow(sz(a));
        for(int i=0; i<sz(a); ++i) a[i]=1LL*a[i]*inv%P;
    }
}

vi or_conv(vi a, vi b)
{
    or_fwt(a, 1), or_fwt(b, 1);
    for(int i=0; i<sz(a); ++i) a[i]=1LL*a[i]*b[i]%P;
    or_fwt(a, -1);
    return a;
}

vi and_conv(vi a, vi b)
{
    and_fwt(a, 1), and_fwt(b, 1);
    for(int i=0; i<sz(a); ++i) a[i]=1LL*a[i]*b[i]%P;
    and_fwt(a, -1);
    return a;
}

vi xor_conv(vi a, vi b)
{
    xor_fwt(a, 1), xor_fwt(b, 1);
    for(int i=0; i<sz(a); ++i) a[i]=1LL*a[i]*b[i]%P;
    xor_fwt(a, -1);
    return a;
}
```

子集卷积

```
// WC2018 州区划分
int n, m, p, vis[22], w[22], v[MAXN], iv[MAXN];
vi vec[22], f[22], g[22];
```

```cpp

int dfs(int x, int sta)
{
    int cnt=1, deg=0;
    vis[x]=1;
    for(int to: vec[x])
        if((sta>>to)&1) {
            deg++;
            if(vis[to]) continue;
            int tmp=dfs(to, sta);
            if(tmp==-1) return -1;
            else cnt+=tmp;
        }
    return deg%2==0?cnt:-1;
}

int main()
{
    scanf("%d%d%d", &n, &m, &p);
    for(int i=1; i<=m; ++i) {
        int x, y;
        scanf("%d%d", &x, &y);
        vec[x-1].PB(y-1);
        vec[y-1].PB(x-1);
    }
    for(int i=0; i<n; ++i) scanf("%d", &w[i]);
    for(int i=0; i<=n; ++i) f[i].resize(1<<n), g[i].resize(1<<n);
    for(int sta=1; sta<1<<n; ++sta) {
        memset(vis, 0, sizeof(vis));
        int cnt=0;
        for(int i=0; i<n; ++i)
            if((sta>>i)&1) cnt++, v[sta]+=w[i];
        if(!p) v[sta]=1;
        else if(p==2) v[sta]=1LL*v[sta]*v[sta]%P;
        if(dfs(__builtin_ctz(sta), sta)==cnt) g[cnt][sta]=0;
        else g[cnt][sta]=v[sta];
        iv[sta]=qpow(v[sta]);
    }
    for(int i=0; i<=n; ++i) fwt(g[i], 0);
    for(int i=0; i<=n; ++i) {
        if(i==0) {
            f[0][0]=1;
            fwt(f[0], 0);
            for(int sta=0; sta<1<<n; ++sta) f[2][sta]=1LL*f[0][sta]*g[2][sta]%P;
            fwt(f[2], 1);
        } else {
            fwt(f[i], 1);
            for(int sta=0; sta<1<<n; ++sta) {
                //printf("%d %d %d\n", i, sta, f[i][sta]);
                if(__builtin_popcount(sta)==i) f[i][sta]=1LL*f[i][sta]*iv[sta]%P;
                else f[i][sta]=0;
            }
            fwt(f[i], 0);
        }
        for(int j=0; j<=n-i; ++j)
            for(int sta=0; sta<1<<n; ++sta)
                f[i+j][sta]=(f[i+j][sta]+1LL*f[i][sta]*g[j][sta])%P;

    }
    fwt(f[n], 1);
    printf("%d\n", f[n][(1<<n)-1]);
    return 0;
}
```

其他

离散化

```cpp
int n, cnt, a[MAXN], b[MAXN], temp[MAXN*2], suba[MAXN], subb[MAXN];

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; ++i)
    {
        scanf("%d%d", &a[i], &b[i]);
        temp[i*2-1]=a[i], temp[2*i]=b[i];
    }
    sort(temp+1, temp+2*n+1);
    cnt=unique(temp+1, temp+2*n+1)-temp-1;
    for(int i=1; i<=n; ++i)
    {
        suba[i]=lower_bound(temp+1, temp+cnt+1, a[i])-temp;
        subb[i]=lower_bound(temp+1, temp+cnt+1, b[i])-temp;
    }
    // suba subb 离散化后数组
    return 0;
}
```

大数运算

```cpp
//只限两个非负整数相加
string add(string a, string b)
{
    string ans;
    int na[MAXL]={0}, nb[MAXL]={0};
    int la=a.size(), lb=b.size();
    for(int i=0; i<la; i++) na[la-1-i]=a[i]-'0';
    for(int i=0; i<lb; i++) nb[lb-1-i]=b[i]-'0';
    int lmax=la>lb?la:lb;
    for(int i=0; i<lmax; i++) na[i]+=nb[i], na[i+1]+=na[i]/10, na[i]%=10;
    if(na[lmax]) lmax++;
    for(int i=lmax-1; i>=0; i--) ans+=na[i]+'0';
    return ans;
}

//只限大的非负整数减小的非负整数
string sub(string a, string b)
{
    string ans;
    int na[MAXL]={0}, nb[MAXL]={0};
    int la=a.size(), lb=b.size();
    for(int i=0; i<la; i++) na[la-1-i]=a[i]-'0';
    for(int i=0; i<lb; i++) nb[lb-1-i]=b[i]-'0';
    int lmax=la>lb?la:lb;
    for(int i=0; i<lmax; i++)
    {
        na[i]-=nb[i];
        if(na[i]<0) na[i]+=10, na[i+1]--;
    }
    while(!na[--lmax]&&lmax>0)  ;lmax++;
    for(int i=lmax-1; i>=0; i--) ans+=na[i]+'0';
    return ans;
}

//只限非负整数相乘
string mul(string a, string b)
{
    string ans;
    int na[MAXL]={0}, nb[MAXL]={0}, nc[MAXL]={0}, La=a.size(), Lb=b.size(); //na存储被乘数，nb存储乘数，nc存储积
    for(int i=La-1; i>=0; i--) na[La-i]=a[i]-'0'; //将字符串表示的大整形数转成i整形数组表示的大整形数
    for(int i=Lb-1; i>=0; i--) nb[Lb-i]=b[i]-'0';
    for(int i=1; i<=La; i++)
        for(int j=1; j<=Lb; j++)
        nc[i+j-1]+=na[i]*nb[j]; //a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）
    for(int i=1; i<=La+Lb; i++)
        nc[i+1]+=nc[i]/10, nc[i]%=10; //统一处理进位
    if(nc[La+Lb]) ans+=nc[La+Lb]+'0'; //判断第i+j位上的数字是不是0
    for(int i=La+Lb-1; i>=1; i--) ans+=nc[i]+'0';
    return ans;
}

//高精度整数除单精度整数
string div(string a, int b)
{
    string r, ans;
    int d=0;
    for(int i=0; i<a.size(); i++)
    {
        r+=(d*10+a[i]-'0')/b+'0';//求出商
        d=(d*10+(a[i]-'0'))%b;//求出余数
    }
    int p=0;
    for(int i=0; i<r.size(); i++)
        if(r[i]!='0') {p=i; break;}
    return r.substr(p);
}
```

位运算

`__builtin_clz(x)` 返回 x 前导 0 的个数

`__builtin_ctz(x)` 返回 x 末尾 0 的个数

`__builtin_popcount(x)` 返回 x 中 1的个数

bitset

`count()` 返回 1 的个数

`any()` 返回是否有 1

`set()` 全体/某位置 1

`reset()` 全体/某位置 0

`flip()` 全体/某位取反

`_Find_first()` 返回最低位 1 的位置

`_Find_next(p)` 返回第 p 位后第一个 1 的位置

string

`insert (size_t pos, const string& str)` 在位置 pos 上插入串 str

`erase (size_t pos, size_t len = npos)` 删除位置 pos 上长度为 len 的串

`find (const string& str, size_t pos = 0)` 返回位置 pos 后串 str 第一次出现的位置

`substr (size_t pos = 0, size_t len = npos)` 返回在位置 pos 上长度为 len 的子串

对拍

```bash
while true; do
    ./gen > a.in
    ./A < a.in > a.out
    ./std < a.in > a.ans
    if diff a.out a.ans; then
        echo AC
    else
        echo WA
        exit 0
    fi
done
```