Berlekamp Massey

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((ll)(x).size())
typedef vector<ll> VI;
typedef pair<ll,ll> PII;
const ll mod = 1e9 + 7;

ll powmod(ll a,ll b) {
    ll res=1;a%=mod;
    assert(b>=0);
    for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;
}

namespace linear_seq {
    const int N=20010;
    ll res[N],base[N],_c[N],_md[N];

    vector<ll> Md;
    void mul(ll *a,ll *b,ll k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (ll i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    ll solve(ll n,VI a,VI b) {
        // b[n + 1] = a[0] * b[n] + a[1] * b[n - 1] + ...
        ll ans=0,pnt=0;
        ll k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (ll p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (ll i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
```

```
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        ll L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    ll gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};
```

ReedSloane

```
#include <cstdio>
#include <vector>
#include <cassert>
#include <functional>
#include <algorithm>
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
// given first m items init[0..m-1] and coefficents trans[0..m-1] or
// given first 2 *m items init[0..2m-1], it will compute trans[0..m-1]
// for you. trans[0..m] should be given as that
//      init[m] = sum_{i=0}^{m-1} init[i] * trans[i]
struct LinearRecurrence {
    using vec = vector<ll>;

    static void extand(vec &a, ll d, ll value = 0) {
```

```
    if (d <= a.size()) return;
    a.resize(d, value);
  }

  static vec BerlekampMassey(const vec &s, ll mod) {
    std::function<ll(ll)> inverse = [&](ll a) {
      return a == 1 ? 1 : (ll)(mod - mod / a) * inverse(mod % a) % mod;
    };
    vec A = {1}, B = {1};
    ll b = s[0];
    for (size_t i = 1, m = 1; i < s.size(); ++i, m++) {
      ll d = 0;
      for (size_t j = 0; j < A.size(); ++j) {
        d += A[j] * s[i - j] % mod;
      }
      if (!(d %= mod)) continue;
      if (2 * (A.size() - 1) <= i) {
        auto temp = A;
        extand(A, B.size() + m);
        ll coef = d * inverse(b) % mod;
        for (size_t j = 0; j < B.size(); ++j) {
          A[j + m] -= coef * B[j] % mod;
          if (A[j + m] < 0) A[j + m] += mod;
        }
        B = temp, b = d, m = 0;
      } else {
        extand(A, B.size() + m);
        ll coef = d * inverse(b) % mod;
        for (size_t j = 0; j < B.size(); ++j) {
          A[j + m] -= coef * B[j] % mod;
          if (A[j + m] < 0) A[j + m] += mod;
        }
      }
    }
    return A;
  }

    static void exgcd(ll a, ll b, ll &g, ll &x, ll &y) {
        if (!b) x = 1, y = 0, g = a;
        else {
            exgcd(b, a % b, g, y, x);
            y -= x * (a / b);
        }
    }
    static ll crt(const vec &c, const vec &m) {
        ll n = c.size();
        ll M = 1, ans = 0;
        for (ll i = 0; i < n; ++i) M *= m[i];
        for (ll i = 0; i < n; ++i) {
            ll x, y, g, tm = M / m[i];
            exgcd(tm, m[i], g, x, y);
            ans = (ans + tm * x * c[i] % M) % M;
        }
        return (ans + M) % M;
    }
    static vec ReedsSloane(const vec &s, ll mod) {
        auto inverse = [] (ll a, ll m) {
            ll d, x, y;
```

```
            exgcd(a, m, d, x, y);
            return d == 1 ? (x % m + m) % m : -1;
        };
        auto L = [] (const vec &a, const vec &b) {
            ll da = (a.size() > 1 || (a.size() == 1 && a[0])) ? (ll)a.size() - 1
: -1000;
            ll db = (b.size() > 1 || (b.size() == 1 && b[0])) ? (ll)b.size() - 1
: -1000;
            return max(da, db + 1);
        };
        auto prime_power = [&] (const vec &s, ll mod, ll p, ll e) {
            // linear feedback shift register mod p^e, p is prime
            vector<vec> a(e), b(e), an(e), bn(e), ao(e), bo(e);
            vec t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);;

            pw[0] = 1;
            for (ll i = pw[0] = 1; i <= e; ++i) pw[i] = pw[i - 1] * p;
            for (ll i = 0; i < e; ++i) {
                a[i] = {pw[i]}, an[i] = {pw[i]};
                b[i] = {0}, bn[i] = {s[0] * pw[i] % mod};
                t[i] = s[0] * pw[i] % mod;
                if (t[i] == 0) {
                    t[i] = 1, u[i] = e;
                } else {
                    for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i]);
                }
            }
            for (ll k = 1; k < s.size(); ++k) {
                for (ll g = 0; g < e; ++g) {
                    if (L(an[g], bn[g]) > L(a[g], b[g])) {
                        ao[g] = a[e - 1 - u[g]];
                        bo[g] = b[e - 1 - u[g]];
                        to[g] = t[e - 1 - u[g]];
                        uo[g] = u[e - 1 - u[g]];
                        r[g] = k - 1;
                    }
                }
                a = an, b = bn;
                for (ll o = 0; o < e; ++o) {
                    ll d = 0;
                    for (ll i = 0; i < a[o].size() && i <= k; ++i) {
                        d = (d + a[o][i] * s[k - i]) % mod;
                    }
                    if (d == 0) {
                        t[o] = 1, u[o] = e;
                    } else {
                        for (u[o] = 0, t[o] = d; t[o] % p == 0; t[o] /= p,
++u[o]);
                        ll g = e - 1 - u[o];
                        if (L(a[g], b[g]) == 0) {
                            extand(bn[o], k + 1);
                            bn[o][k] = (bn[o][k] + d) % mod;
                        } else {
                            ll coef = t[o] * inverse(to[g], mod)%mod*pw[u[o]-
uo[g]]% mod;
                            ll m = k - r[g];
                            extand(an[o], ao[g].size() + m);
                            extand(bn[o], bo[g].size() + m);
```

```
                            for (ll i = 0; i < ao[g].size(); ++i) {
                                an[o][i + m] -= coef * ao[g][i] % mod;
                                if (an[o][i + m] < 0) an[o][i + m] += mod;
                            }
                            while (an[o].size() && an[o].back() == 0)
an[o].pop_back();
                            for (ll i = 0; i < bo[g].size(); ++i) {
                                bn[o][i + m] -= coef * bo[g][i] % mod;
                                if (bn[o][i + m] < 0) bn[o][i + m] -= mod;
                            }
                            while (bn[o].size() && bn[o].back() == 0)
bn[o].pop_back();
                        }
                    }
                }
            }
            return make_pair(an[0], bn[0]);
        };

        vector<tuple<ll, ll, ll>> fac;
        for (ll i = 2; i * i <= mod; ++i) if (mod % i == 0) {
                ll cnt = 0, pw = 1;
                while (mod % i == 0) mod /= i, ++cnt, pw *= i;
                fac.emplace_back(pw, i, cnt);
            }
        if (mod > 1) fac.emplace_back(mod, mod, 1);
        vector<vec> as;
        ll n = 0;
        for (auto &&x: fac) {
            ll mod, p, e;
            vec a, b;
            tie(mod, p, e) = x;
            auto ss = s;
            for (auto &&x: ss) x %= mod;
            tie(a, b) = prime_power(ss, mod, p, e);
            as.emplace_back(a);
            n = max(n, (ll) a.size());
        }
        vec a(n), c(as.size()), m(as.size());

        for (ll i = 0; i < n; ++i) {
            for (ll j = 0; j < as.size(); ++j) {
                m[j] = get<0>(fac[j]);
                c[j] = i < as[j].size() ? as[j][i] : 0;
            }
            a[i] = crt(c, m);
        }
        return a;
    }

    LinearRecurrence(const vec &s, const vec &c, ll mod):
        init(s), trans(c), mod(mod), m(s.size()) {}

    LinearRecurrence(const vec &s, ll mod, bool is_prime = true): mod(mod) {
        vec A;
        if(is_prime) A = BerlekampMassey(s,mod);
        else A = ReedsSloane(s, mod);
        if (A.empty()) A = {0};
```

```
        m = A.size() - 1;
        trans.resize(m);
        for (ll i = 0; i < m; ++i) {
            trans[i] = (mod - A[i + 1]) % mod;
        }
        reverse(trans.begin(), trans.end());
        init = {s.begin(), s.begin() + m};
    }

    ll calc(ll n) {
        if (mod == 1) return 0;
        if (n < m) return init[n];
        vec v(m), u(m << 1);

        ll msk = !!n;
        for (ll m = n; m > 1; m >>= 1LL) msk <<= 1LL;
        v[0] = 1 % mod;
        for (ll x = 0; msk; msk >>= 1LL, x <<= 1LL) {
            fill_n(u.begin(), m * 2, 0);
            x |= !!(n & msk);
            if (x < m) u[x] = 1 % mod;
            else { // can be optimized by fft/ntt
                for (ll i = 0; i < m; ++i) {
                    for (ll j = 0, t = i + (x & 1); j < m; ++j, ++t) {
                        u[t] = (u[t] + v[i] * v[j]) % mod;
                    }
                }
                for (ll i = m * 2 - 1; i >= m; --i) {
                    for (ll j = 0, t = i - m; j < m; ++j, ++t) {
                        u[t] = (u[t] + trans[j] * u[i]) % mod;
                    }
                }
            }
            v = {u.begin(), u.begin() + m};
        }
        ll ret = 0;
        for (ll i = 0; i < m; ++i) {
            ret = (ret + v[i] * init[i]) % mod;
        }
        return ret;
    }

    vec init, trans;
    ll mod;
    ll m;
};
const int NN = 2e3 + 7;
vector < ll > init, fib;
const ll mod = 1e9;

ll power_mod(ll a, ll b) {
    ll ret = 1;
    for(a %= mod; b; b>>=1, a=(ll)a*a%mod)
        if(b&1) ret = (ll) ret*a%mod;
    return ret;
}

int main(){
```

```
#ifdef local
    freopen("in.txt", "r", stdin);
#endif
/*
    ll t, cs = 1;

    cin >> t;
    for(cs = 1; cs <= t; cs++){

        init.clear();

        scanf("%lld %lld", &k, &mod);
        assert(k>=1 && k<=50);
        assert(mod>=1 && mod<(1LL<<31));

        for(ll i = 0; i < 2*k; i++){
            scanf("%lld", &x);
            init.push_back(x);
            assert(x>=0 && x<mod);
        }

        bool isPrime = true;
        for (ll i = 2; i * i <= mod; ++i) if (mod % i == 0) isPrime = false;

        LinearRecurrence lr(init, mod, isPrime);
        printf("%lld\n", lr.calc(2 * k));
    }*/
    int n, m; cin >> n >> m;
    init.resize(NN), fib.resize(NN);
    fib[0] = 0, fib[1] = 1;
    for(int i = 2; i < NN; i++) fib[i] = (fib[i-1] + fib[i-2]) % mod;
    for(int i = 1; i < NN; i++) {
        init[i] = power_mod(fib[i], m);
        init[i] = (init[i] +  init[i-1]) % mod;
    }

    LinearRecurrence lr(init, mod, 0);
    printf("%lld\n", lr.calc(n));

    return 0;
}
```

Fast Fourier Transformation

```
#include<bits/stdc++.h>
using namespace std;
typedef double db;
typedef long long ll;
#define all(x) (x).begin(), (x).end()
const db pi = acos(-1.0);
const int p = 1e9 + 9;
int power_mod(int a, int b) {
    int ret = 1;
    for(a %= p; b; b>>=1,a=(ll)a*a%p)
```

```
        if(b&1) ret=(ll)ret*a%p;
    return ret;
}
namespace FFT {
    struct Comp {
        db x, y;
        Comp(db _x = 0, db _y = 0) : x(_x), y(_y){}
        Comp operator + (const Comp &r) { return Comp(x + r.x, y + r.y);}
        Comp operator - (const Comp &r) { return Comp(x - r.x, y - r.y);}
        Comp operator * (const Comp &r) { return Comp(x*r.x-y*r.y,
x*r.y+y*r.x);}
        Comp operator / (const db &r) { return Comp(x/r, y/r);}
        friend Comp conj(const Comp &r) { return Comp(r.x, -r.y);}
    };
    Comp Exp(const db &r) { return Comp(cos(r), sin(r));}
    const int L = 18, N = 1 << L;
    int rev[N];
    Comp roots[N*2];
    void fft_init() {
        for(int i = 0; i < N; i++)
            rev[i] = (rev[i>>1]>>1)|((i&1)<<L-1);
        roots[1] = {1, 0};
        for(int i = 1; i < L+1; i++) {
            db angle = 2*pi/(1<<i+1);
            for(int j = 1<<i-1; j<1<<i; j++) {
                roots[j<<1] = roots[j];
                roots[j<<1|1] = Exp((j*2+1-(1<<i))*angle);
            }
        }
    }
    void fft(Comp *y, int n, int on = 0) {
        assert((n & (n - 1)) == 0);
        int zeros = __builtin_ctz(n), shift = L - zeros;
        for(int i = 0; i < n; i++)
            if(i < (rev[i]>>shift))
                swap(y[i], y[rev[i]>>shift]);
        for(int k = 1; k < n; k <<= 1) {
            for(int i = 0; i < n; i += k * 2) {
                for(int j = 0; j < k; j++) {
                    Comp z = y[i+j+k]*(on?conj(roots[j+k]):roots[j+k]);
                    y[i+j+k] = y[i+j]-z, y[i+j] = y[i+j]+z;
                }
            }
        }
        if(on) for(int i = 0; i < n; i++) y[i] = y[i]/n;
    }

    int gl(int x) { while(x&(x-1)) x+=x&-x; return x;}

    const Comp half = 0.5, zero = 0;
    const int M = (1 << 15) - 1;
    Comp fa[N], fb[N];
    void conv(int *a, int *b, int *c, int l1, int l2, int eq=0) {
        static ll d[4];
        int l = gl(l1 + l2 - 1);
        for(int i = 0; i < l; i++)
            fa[i] = i<l1? Comp{ a[i]&M, a[i]>>15}:zero;
        fft(fa, l, 0);
```

```
        if(eq) memcpy(fb, fa, l*sizeof(Comp));
        else {
            for(int i = 0; i < l; i++)
                fb[i] = i<l2?Comp{ b[i]&M, b[i]>>15}:zero;
            fft(fb, l, 0);
        }
        for(int i = 0, j = 0; i <= (l>>1); i++, j = l - i) {
            Comp xx = fa[i] * fb[i];
            Comp yy = conj(fa[j]) * fb[i];
            Comp zz = fa[i] * conj(fb[j]);
            Comp ww = conj(fa[j]) * conj(fb[j]);
            if(i != j) {
                Comp _x = fa[j]*fb[j];
                Comp _y = conj(fa[i])*fb[j];
                Comp _z = fa[j]*conj(fb[i]);
                Comp _w = conj(fa[i])*conj(fb[i]);
                fa[j] = (_x + _y) * half;
                fb[j] = (_z - _w) * half;
            }
            fa[i] = (xx + yy) * half;
            fb[i] = (zz - ww) * half;
        }
        fft(fa, l, 1), fft(fb, l, 1);
        int need = l1 + l2 - 1;
        for(int i = 0; i < need; i++) { \\ watch out the sign
            d[0] = fa[i].x + 0.5, d[1] = fa[i].y + 0.5;
            d[2] = fb[i].x + 0.5, d[3] = fb[i].y + 0.5;
            c[i] = (((d[2]%p)<<30) + d[0] + ((d[1]+d[3])%p<<15)) % p;
        }
    }
    void sqr(int *a, int *b, int l) { conv(a, a, b, l, l, 1);}
    void poly_inv(int *a, int *b, int len) {
        static int t[N];
        if(len == 1) return b[0] = power_mod(a[0], p - 2), void(b[1] = 0);
        poly_inv(a, b, len>>1);
        sqr(b, t, len>>1);
        conv(a, t, t, len, len-1);
        for(int i = (len>>1); i < len; i++)
            b[i] = (- t[i] + p) % p;
        //for(int i = len; i < 2*len; i++) b[i] = 0;
    }
    void poly_div(int *a, int *b, int *c, int *d, int n, int m) {
        assert(n >= m);
        static int invb[N], tb[N];
        int lm = gl(n-m+1);
        memcpy(tb, b, m<<2); memset(tb+m, 0, max(lm-m,0)<<2);
        reverse(a, a+n); reverse(tb, tb + m);
        poly_inv(tb, invb, lm);
        //for(int i = 0; i < lm; i++) cout << invb[i] << " \n"[i==lm-1];
        conv(a, invb, c, n - m + 1, n - m + 1);
        reverse(c, c + n - m + 1); reverse(a, a + n);
        conv(c, b, tb, n - m + 1, m);
        for(int i = 0; i < m - 1; i++) d[i] = (a[i] + p - tb[i]) % p;
    }
    int t[N];
    void sqr_mod(int *a, int *Mod, int lm) {
        sqr(a, a, lm-1);
        poly_div(a, Mod, t, a, 2*lm-3, lm);
```

```
    }
    inline void sub(int &x, ll y) { if((x-=y)<0) x+=p;}
    void Pow_mod(int *res, ll n, int *Mod, int lm) {
        int pnt = 0;
        while((1ll<<pnt)<=n) pnt++;
        memset(res, 0, lm<<2); res[0] = 1;
        for(int i = pnt-1; ~i; i--) {
            sqr_mod(res, Mod, lm);
            if(n>>i&1) {
                for(int j = lm-2; j>=0; j--) res[j+1] = res[j]; res[0] = 0;
                for(int j = 0; j < lm-1; j++) sub(res[j], (ll)res[lm-
1]*Mod[j]%p);
            }
        }
    }
}
using namespace FFT;
```

Poly Ln and Exp

```
oid invpoly(int *A, int *B, int len) {
    static int x[N], i;
    if(len == 1) return B[0] = 1, void(B[1] = 0);
    invpoly(A, B, len>>1);
    for(i = 0; i < len; i++) x[i] = A[i], x[i+len]=B[i+len]=0;
    ntt.ntt(x, len<<1, 0), ntt.ntt(B, len<<1, 0);
    for(i = 0; i < 2 * len; i++)
        B[i] = B[i] * (2 + (ll)(P-B[i])*x[i]%P)%P;
    ntt.ntt(B, len<<1, 1);
    for(i = len; i < 2 * len; i++) B[i] = 0;
}
void logpoly(int *A, int *B, int len) {
    static int x[N], i;
    invpoly(A, x, len);
    for(i = 0; i < len; i++) B[i] = A[i+1]*(i+1ll)%P;
    for(i = len-1; i < 2 * len; i++) B[i] = 0;
    ntt.ntt(x, len<<1, 0), ntt.ntt(B, len<<1, 0);
    for(i = 0; i < 2 * len; i++)
        B[i] = (ll)x[i]*B[i]%P;
    ntt.ntt(B, len<<1, 1);
    for(i = len; i < 2 * len; i++) B[i] = 0;
    for(i = len-1; i; i--) B[i] = (ll)B[i-1]*inv[i]%P;
    B[0] = 0;
}
void exppoly(int *A, int *B, int len) {
    static int x[N], i;
    if(len == 1) return B[0] = 1, void(B[1] = 0);
    exppoly(A, B, len>>1);
    logpoly(B, x, len); x[0] = sub(x[0], 1);
    for(i = 0; i < len; i++) x[i] = sub(A[i], x[i]);
    for(i = len; i < 2 * len; i++) x[i] = 0;
    ntt.ntt(B, len<<1, 0), ntt.ntt(x, len<<1, 0);
    for(i = 0; i < 2 * len; i++)
        B[i] = (ll)B[i] * x[i] % P;
```

```
    ntt.ntt(B, len<<1, 1);
    for(i = len; i < 2 * len; i++) B[i] = 0;
}
```

SA

```
#include <algorithm>
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cmath>
using namespace std;

const int N = 100005;
const int LOGN = 17;

struct SA {
int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
#define pushS(x) sa[cur[s[x]]--] = x
#define pushL(x) sa[cur[s[x]]++] = x
#define inducedSort(v) fill_n(sa, n, -1); fill_n(cnt, m, 0);                  \
    for (int i = 0; i < n; i++) cnt[s[i]]++;                                  \
    for (int i = 1; i < m; i++) cnt[i] += cnt[i-1];                           \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
    for (int i = n1-1; ~i; i--) pushS(v[i]);                                  \
    for (int i = 1; i < m; i++) cur[i] = cnt[i-1];                            \
    for (int i = 0; i < n; i++) if (sa[i] > 0 &&  t[sa[i]-1]) pushL(sa[i]-1); \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
    for (int i = n-1;  ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
    void sais(int n, int m, int *s, int *t, int *p) {
        int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
        for (int i = n-2; ~i; i--)
            t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
        for (int i = 1; i < n; i++)
            rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
        inducedSort(p);
        for (int i = 0, x, y; i < n; i++) {
            if (~(x = rk[sa[i]])) {
                if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y])
                    ch++;
                else {
                    for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++) {
                        if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {
                            ch++;
                            break;
                        }
                    }
                }
                s1[y = x] = ch;
            }
        }
        if (ch+1 < n1)
```

```
            sais(n1, ch+1, s1, t+n, p+n1);
        else {
            for (int i = 0; i < n1; i++)
                sa[s1[i]] = i;
        }
        for (int i = 0; i < n1; i++)
            s1[i] = p[sa[i]];
        inducedSort(s1);
    }
    template<typename T>
    int mapCharToInt(int n, const T *str) {
        int m = *max_element(str, str+n);
        fill_n(rk, m+1, 0);
        for (int i = 0; i < n; i++)
            rk[str[i]] = 1;
        for (int i = 0; i < m; i++)
            rk[i+1] += rk[i];
        for (int i = 0; i < n; i++)
            s[i] = rk[str[i]] - 1;
        return rk[m];
    }
    int sz;
    // Ensure that str[n] is the unique lexicographically smallest character in
str.
    template<typename T>
    void suffixArray(int n, const T *str) {
        sz = n;
        int m = mapCharToInt(++n, str);
        sais(n, m, s, t, p);
        for (int i = 0; i < n; i++)
            rk[sa[i]] = i;
        for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
            int j = sa[rk[i]-1];
            while (i+h < n && j+h < n && s[i+h] == s[j+h])
                h++;
            if (ht[rk[i]] = h)
                h--;
        }
        rmq.init(ht, n);
    }
    struct RMQ {
      int dp[N][LOGN];
      void init(int *a, int n) {
        for(int i = 1; i <= n; i++) dp[i][0] = a[i];
        int lgn = __lg(n);
        for(int i = 1; i <= lgn; i++) {
          for(int j = 1; j <= n-(1<<i)+1; j++)
            dp[j][i] = min(dp[j][i-1], dp[j+(1<<i-1)][i-1]);
        }
      }
      int query(int l, int r) {
        int lgt = __lg(r - l + 1);
        return min(dp[l][lgt], dp[r-(1<<lgt)+1][lgt]);
      }
    } rmq;

    int lcp(int i, int j) {
      if(i == j) return sz - i;
```

```
      i = rk[i], j = rk[j];
      if(i > j) swap(i, j);
      return rmq.query(i + 1, j);
    }
#ifdef local
    void debug(const char *s, int n) {
      for(int i = 0; i <= n; i++) cout << i << ' ' << s + sa[i] << endl;
      for(int i = 0; i <= n; i++) cout << i << ' ' << ht[i] << endl;
    }
#else
    #define debug(...)
#endif
} pr, sf;
```

Montgomery Residue

```
#include<cstdio>
#include<algorithm>
#include<map>
using namespace std;
const int N=524288,M=200010,K=18,P=998244353,G=3;
typedef unsigned int uint32;
typedef long long int64;
typedef unsigned long long uint64;
typedef uint32 word;
typedef uint64 dword;
typedef int sword;
const int word_bits=sizeof(word)*8;
word mod,Modinv,r2;
struct UnsafeMod{
  word x;
  UnsafeMod(): x(0) {}
  UnsafeMod(word _x): x(init(_x)) {}
  UnsafeMod& operator += (const UnsafeMod& rhs) {
    (x += rhs.x) >= mod && (x -= mod);
    return *this;
  }
  UnsafeMod& operator -= (const UnsafeMod& rhs) {
    sword(x -= rhs.x) < 0 && (x += mod);
    return *this;
  }
  UnsafeMod& operator *= (const UnsafeMod& rhs) {
    x = reduce(dword(x) * rhs.x);
    return *this;
  }
  UnsafeMod operator + (const UnsafeMod &rhs) const {
    return UnsafeMod(*this) += rhs;
  }
  UnsafeMod operator - (const UnsafeMod &rhs) const {
    return UnsafeMod(*this) -= rhs;
  }
  UnsafeMod operator * (const UnsafeMod &rhs) const {
```

```
    return UnsafeMod(*this) *= rhs;
  }
  UnsafeMod pow(uint64 e) const {
    UnsafeMod ret(1);
    for (UnsafeMod base = *this; e; e >>= 1, base *= base) {
      if (e & 1) ret *= base;
    }
    return ret;
  }
  word get() const {
    return reduce(x);
  }
  static word modulus() {
    return mod;
  }
  static word init(word w) {
    return reduce(dword(w) * r2);
  }
  static void set_mod(word m) {
    mod = m;
    Modinv = mul_inv(mod);
    r2 = -dword(mod) % mod;
  }
  static word reduce(dword x) {
    word y = word(x >> word_bits) - word((dword(word(x) * Modinv) * mod) >>
word_bits);
    return sword(y) < 0 ? y + mod : y;
  }
  static word mul_inv(word n, int e = 6, word x = 1) {
    return !e ? x : mul_inv(n, e - 1, x * (2 - x * n));
  }
};
```

Radix sort

```
const int Base = 1 << 8;
const int H = Base - 1;
int base[Base];
int tt[N];
void rdx_sort(int *s, int *t) {
  int n = t - s;
  int *x = s, *y = tt;
  for(int o = 0; o < 32; o += 4) {
    memset(base, 0, sizeof base);
    for(int i = 0; i < n; i++) base[(x[i]>>o)&H]++;
    for(int i = 1; i < Base; i++) base[i] += base[i-1];
    for(int i = n - 1; ~i; i--) y[--base[(x[i]>>o)&H]] = x[i];
    swap(x, y);
  }
}
```