Team Note of ConForza

Compiled on 2025년 4월 19일

차 례

1 Have you...

1.1 tried...

- **Reading the problem once more?**
- doubting “obvious” things?
- writing obivous things?
- radical greedy approach?
- thinking in reverse direction?
- a greedy algorithm?
- network flow when your greedy algorithms stuck?
- a dynamic programming?
- checking the range of answer?
- random algorithm?
- graph modeling using states?
- inverting state only on odd indexes?
- calculating error bound on a real number usage?

1.2 checked...

- **you have read the statement correctly?**
- typo copying the team note?
- initialization on multiple test case problem?
- additional information from the problem?
- undefined behavior?
- overflow?
- function without return value?
- real number error?
- implicit conversion?
- comparison between signed and unsigned integer?

2 Algorithmic Idea Note

I. Complete Search: Backtracking & Pruning
II. Math
  A. Number Theory
    1. Prime Number
      i) Sieve of Eratosthenes, Prime Factorization
      ii) Fast Prime Verdict; Millar-Rabin
      iii) Fast Prime Factorization; Pollad Rho
    2. Extended Euclidean Algorithm; Diophantos Equation
    3. Chinese Remainder Theorem
    4. Harmonic Lemma
    5. Floor Sum (Sum of Rational Arithmetic Sequence)
    6. Several Sieves
  B. Linear Programming
    1. Solve (some) LP with Shortest Path
  C. FFT & Polynomials
    1. FFT : Convolution
      i) High precision FFT with modulo 1e9+7
    2. NTT : Number Theoretic Tranform
    3. Quotient Ring (Formal Power Series)
      i) Multiplication
      ii) FPS : Inverse / Division
      iii) Integration / Differentiation
      iv) FPS : Logarithm / Exponentiation
      v) FPS : Power of Polynomial
      vi) Division - Quotient & Remainder
      vii) Polynomial Taylor Shift
      viii) Multipoint Evaluation
  D. Combinatorics
    1. Labeled Combinatorial Target
    2. The Twelvefold Way (12정도)
    3. Generating Function
III. Linear Algebra
IV. Geometry
  A. Basic Tools
    1. Outer Product (CCW)
    2. Sorting by Polar
    3. Segment Intersection
    4. Closest Point
    5. Furthest Point
  B. Convex Polygon (Convex Hull)
    1. Convex Hull Construction
    2. Convex Layer
    3. Rotating Calipers
    4. Point Containment
    5. Tangent to convex polygon
    6. Inner and Outer Tangent of two Convex’s
  C. General Polygon
  D. Half Plane Intersection
  E. Delaunay Triangulation : Voronoi diagram
V. Greedy

2.1 Some Rules

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

#define getint(n) int n; scanf("%d%*c", &n)
#define getll(n) long long n; scanf("%lld%*c", &n)
#define getchar(n) char n; scanf("%c%*c", &n);
#define intab getint(a); getint(b)

#define forr(i, n) for(int i=1;i<=(n);i++)
#define fors(i, s, e) for(int i=(s); i<=(e); i++)
#define fore(i, e, s) for(int i=(e); i>=(s); i--)

#define fi first
#define se second
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()
#define pb push_back

using namespace std;
using ll = long long;          using lll = __int128_t;
using pii = pair<int,int>;     using pll = pair<ll,ll>;
using vi = vector<int>;        using vl = vector<ll>;
using vii = vector<pii>;       using vll = vector<pll>;
```

3 Math

3.1 Prime Number

3.1.1 Distribution of Prime Number

1e2	25	1e6	78,498	1e10	<5e8
1e3	168	1e7	664,579	1e11	<5e9
1e4	1,229	1e8	<6e6	1e12	<4e10
1e5	9,592	1e9	<6e7	1e13	<4e11

3.1.2 Prime Gap

$$2 \cdot 10^5 \text{ 이하의 소수 간극 } \le 100$$
$$2^{32} \text{ 이하의 소수 간극 } \le 464$$
$$2^{64} \text{ 이하의 소수 간극 } \le 1550$$

3.2 Miller-Rabin Algorithm

Usage: `is_p(X)` : returns true if X is prime, otherwise false.

When $X \le 2^{32}, D = \{2, 7, 61\}$ is sufficient;
$X \le 2^{64}, D = \{p | p \text{ is prime}, p \le 37\}$ is sufficient.

Time Complexity: $\mathcal{O}(\log^3 X)$

```
ll pow(ll a, ll b, ll mod)
{
    ll ret = 1;
    for(int st=0; (1LL<<st) <= b; st++)
    {
        if((1LL<<st) & b) ret=(lll)ret*a%mod;
        a=(lll)a*a%mod;
    }
    return ret;
}
bool miller(ll n, ll a)
{
    if(n == a) return true;
    ll x = n-1;
    if(pow(a, x, n) != 1) return false;
    while(x%2==0)
    {
        x/=2;
        ll t = pow(a, x, n);
        if(t!=1 and t!=n-1) return false;
        if(t==n-1) return true;
    }
    return true;
}
bool is_p(ll n)
{
    if(n<=2) return n==2;
    vi D = {2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37};
    for(auto i:D) if(!miller(n, i)) return false;
    return true;
}
```

3.3 Pollad Rho Algorithm

Usage: `po_rho(N)` : returns array of prime factors of X.
Time Complexity: $\mathcal{O}(N^{1/4})$

```
void fact(ll n, vl& ret)
{
    if(n == 1) return;
    if(n%2 == 0)
    {
        ret.pb(2);
        fact(n/2, ret);
        return;
    }
    if(is_p(n))
    {
        ret.pb(n);
        return;
    }

    ll a, b, c, g = n;
```

```cpp
    auto f = [&c, &n](ll x)->ll{return (c+(lll)x*x)%n;};
    do
    {
        if(g == n) a=b=rand()%(n-2)+2, c=rand()%20+1;
        a=f(a); b=f(f(b));
        g = gcd(a-b, n);
    }while(g == 1);
    fact(g, ret); fact(n/g, ret);
}
vl po_rho(ll n)
{
    vl ret;
    fact(n, ret);
    sort(all(ret));
    return ret;
}
```

3.4 Diopantos Equation(Extended Euclidian Algorithm)

Usage: `diophantos(a, b)` : return one integer solution of $ax+by=1$, satisfying $0 \le x < b$.

Time Complexity: $\mathcal{O}\left(\log\left(\max(a,b)\right)\right)$

```cpp
pll diophantos(ll a, ll b)
{
    assert(a>0 and b>=0);
    if(b == 0) return {1, 0};
    auto [y, x] = diophantos(b, a%b); y = y-(a/b)*x;
    if(x < 0 or x >= b)
    {
        ll t = x/b;
        if(x%b < 0) t--;

        x -= b*t; y += a*t;
    }
    return {x, y};
}
```

3.5 Chinese Remainder Theorem

Usage: `crt(pll p, pll q)` : return `pll r`, satisfying follows:

$$\begin{aligned} x &\equiv \texttt{p.fi} \mod \texttt{p.se} \\ \text{and } x &\equiv \texttt{q.fi} \mod \texttt{q.se} \\ \leftrightarrow x &\equiv \texttt{r.fi} \mod \texttt{r.se} \end{aligned}$$

If there's no such `r`, return $\{-1,-1\}$.

Time Complexity: $\mathcal{O}(\log A)$

```cpp
pll crt(pll p, pll q)
{
    if(p.fi > q.fi) swap(p, q);
    auto [a, A] = p;
    auto [b, B] = q;

    ll g = gcd(A, B);
    if((b-a)%g != 0) return {-1, -1};

    ll i = A, j = B, k = b-a;
    i/=g; j/=g; k/=g;
    auto [x, y] = diophantos(i, j);
    return {(ll)((a+(lll)A*k*x)%(A*B/g)), A*B/g};
}
```

3.6 Harmonic Lemma

Usage: `f(N)` : return the value

$$\sum_{i=1}^{N}\left\lfloor\frac{N}{i}\right\rfloor = \left\lfloor\frac{N}{1}\right\rfloor + \left\lfloor\frac{N}{2}\right\rfloor + \cdots + \left\lfloor\frac{N}{N}\right\rfloor$$

Time Complexity: $\mathcal{O}(\sqrt{N})$

```cpp
ll f(int n)
{
    ll ans = 0;
    for(int i = 1; i <= n; i = n/(n/i)+ 1)
        ans += (ll)(n/(n/i)-i+1)*(n/i);
    return ans;
}
```

3.7 Floor Sum (Sum of Floor of Ratinoal Arithmetic Sequence)

Usage: `floor_sum(A, B, C, N)` : retun the value

$$\sum_{x=1}^{N}\left\lfloor\frac{Ax+B}{C}\right\rfloor$$

Time Complexity: $\mathcal{O}(\log N)$

```cpp
ll floor_sum(ll a, ll b, ll c, ll n)
{
    if(a>=c or b>=c) return n*(n-1)/2 * (a/c) + n * (b/c) +
    floor_sum(a%c, b%c, c, n);
    if(a == 0) return b/c*n;

    ll m = (a*(n-1)+b)/c;
    return m*(n-1) - floor_sum(c, c-b-1, a, m);
}
```

3.8 FFT - Convolution

Time Complexity: $\mathcal{O}(N\log N)$

```cpp
using cpx = complex<double>;
using vcpx = vector<cpx>;
void fft(vcpx &a, bool inv = false)
{
    int n = a.size(), j = 0; assert((n&-n) == n);
    for(int i=1; i<n; i++)
    {
        int bit = (n >> 1);
        while(j >= bit)
        {
            j -= bit;
            bit >>= 1;
        }
        j += bit;
        if(i < j) swap(a[i], a[j]);
    }

    vcpx roots(n/2);
    prec c = 2 * pi * (inv ? -1 : 1);
```

```
    for(int i=0; i<n/2; i++)
        roots[i] = cpx(cosl(c * i / n), sinl(c * i / n));

    for(int i=2; i<=n; i<<=1)
    {
        int step = n / i;
        for(int j=0; j<n; j+=i)
        {
            for(int k=0; k<i/2; k++)
            {
                cpx u = a[j+k], v = a[j+k+i/2]*roots[step*k];
                a[j+k] = u+v;
                a[j+k+i/2] = u-v;
            }
        }
    }

    if(inv) for(int i=0; i<n; i++) a[i] /= n;
}

ll mod = 1e9+7;
vl conv(const vl& AA,const vl& BB)
{
    const ll G = 1<<15;
    int n = AA.size()+BB.size()-1;
    int m = 1; while(m < n) m<<=1;

    int a = AA.size(), b = BB.size();

    vcpx A(m), B(m), C(m), D(m);
    fors(i, 0, a-1) A[i] = cpx(AA[i]/G, AA[i]%G);
    fors(i, 0, b-1) B[i] = cpx(BB[i]/G, BB[i]%G);

    fft(A); fft(B);

    fors(i, 0, m-1)
    {
        int j = i?m-i:0;
        cpx A1 = (A[i]+conj(A[j]))*cpx(0.5, 0);
        cpx A2 = (A[i]-conj(A[j]))*cpx(0, -0.5);

        cpx B1 = (B[i]+conj(B[j]))*cpx(0.5, 0);
        cpx B2 = (B[i]-conj(B[j]))*cpx(0, -0.5);

        C[i] = A1*B1 + A2*B2*cpx(0, 1);
        D[i] = A2*B1 + A1*B2*cpx(0, 1);
    }

    fft(C, true); fft(D, true);


    vl ret(m); ll G1 = G%mod, G2 = (lll)G*G%mod;
    fors(i, 0, m-1)
    {
        ll p = ll(C[i].real()+0.5);
        ll q = ll(D[i].real()+0.5) + ll(D[i].imag()+0.5);
        ll r = ll(C[i].imag()+0.5);
        p %= mod; q %= mod; r %= mod;
        ret[i] = (((lll)p*G2)%mod+((lll)q*G1)%mod+r%mod)%mod;
    }
    ret.resize(n);
    return ret;
}
```

3.9 NTT - Number Theoretic Transform

Usage: helloworld

```
namespace GMS
{
    template<ll mod>
    ll pow(ll a, ll b)
    {
        static_assert(mod <= (ll)2e9, "mod should be less than
        2e9");
        a %= mod;
        ll ret = 1;
        while(b != 0)
        {
            if(b&1) ret = ret*a%mod;
            a = a*a%mod; b>>=1;
        }

        return ret;
    }
    template<ll mod, ll w>
    void ntt(vector<ll> &a, bool inv = false)
    {
        static_assert(mod <= (ll)2e9, "mod should be less than
        2e9");
        int n = a.size(), j = 0;

        assert((n & -n) == n);
        assert((mod-1)%n == 0);

        for(int i=1; i<n; i++)
        {
            int bit = (n >> 1);
            while(j >= bit){
                j -= bit;
                bit >>= 1;
            }
            j += bit;
            if(i < j) swap(a[i], a[j]);
        }

        static vector<ll> root[30], iroot[30];
        for(int st=1; (1<<st) <= n; st++)
        {
            if(root[st].empty())
            {
                ll t = pow<mod>(w, (mod-1)/(1<<st));

                root[st].pb(1);
                for(int i=1; i<(1<<(st-1)); i++)
                    root[st].pb(root[st].back()*t%mod);
            }
```

```
            if(iroot[st].empty())
            {
                ll t = pow<mod>(w, (mod-1)/(1<<st));
                t = pow<mod>(t, mod-2);

                iroot[st].pb(1);
                for(int i=1; i<(1<<(st-1)); i++)
                    iroot[st].pb(iroot[st].back()*t%mod);
            }
        }

        vector<ll>* r = (inv?root:iroot);

        for(int st = 1; (1<<st) <= n; st++)
        {
            int i = 1<<st; //int step = n / i;
            for(int j=0; j<n; j+=i)
            {
                for(int k=0; k<i/2; k++)
                {
                    ll u = a[j+k], v = a[j+k+i/2] * r[st]
                    [k]%mod;
                    a[j+k] = (u+v)%mod;
                    a[j+k+i/2] = (mod+u-v)%mod;
                }
            }
        }
        if(inv)
        {
            ll in = pow<mod>(n, mod-2);
            for(int i=0; i<n; i++) a[i] = a[i]*in%mod;
        }
    }

    template<ll mod, ll w>
    vl conv(vl A, vl B)
    {
        int n = A.size(), m = B.size();
        int t = 1; while(t < n+m-1) t*=2;
        A.resize(t); B.resize(t);

        ntt<mod, w>(A); ntt<mod, w>(B);

        fors(i, 0, t-1) A[i] = A[i]*B[i]%mod;

        ntt<mod, w>(A, true);
        A.resize(n+m-1);

        return A;
    }
} // namespace GMS
```

3.9.1 Good prime numbers to run NTT

595 591 169	71<<23|1	$\omega = 3$
645 922 817	77<<23|1	
897 581 057	107<<23|1	
998 244 353	119<<23|1	
1 300 234 241	155<<23|1	
1 224 736 769	73<<24|1	
2 130 706 433	127<<24|1	
167 772 161	5<<25|1	
469 762 049	7<<26|1	

3.10 Polynomial (Formal Power Series)

```
namespace GMS
{
    template<ll T, ll mod, ll w>
    struct Qring : public vl
    {
        using poly = Qring<T, mod, w>;
        Qring()                 : vl(1, 0){}
        Qring(ll c)             : vl(1, c){}
        Qring(ll c, int n)      : vl(n, c){}
        Qring(const vl& cp)     : vl(cp){}

        ll& operator[](ll idx)
        {
            if((unsigned)idx < size()) return vl::operator[]
            (idx);
            this->resize(idx+1); return vl::operator[](idx);
        }
        ll operator[](ll idx) const
        {
            if((unsigned)idx < size()) return vl::operator[]
            (idx);
            return 0LL;
        }

        void adjust()
        {
            while(size() > T) pop_back();
            while(size() > 1 and back() == 0) pop_back();
        }
        void adjust(int n){resize(n, 0);}

        ll operator()(ll x)
        {
            x %= mod;
            ll ret = 0;
            for(auto it=rbegin(); it!=rend(); it++)
                ret = (ret*x+*it)%mod;

            return ret;
        }
```

```
friend poly operator%(const poly& A, int B) //
remainder by x^B
{
    poly ret(A);
    ret.resize(B, 0);
    return ret;
}
friend poly operator%(poly&& A, int B) // remainder by
x^B
{
    A.resize(B, 0);
    return A;
}

friend poly operator+(const poly& A, const poly& B)
{
    int n = max(A.size(), B.size());
    poly ret(0, n);
    fors(i, 0, n-1) ret[i] = (A[i]+B[i])%mod;
    ret.adjust();
    return ret;
}
friend poly operator+(poly&& A, const poly& B)
{
    int n = B.size();
    fors(i, 0, n-1) A[i] = (A[i]+B[i])%mod;
    A.adjust();
    return A;
}

friend poly operator-(const poly& A)
{
    int n = A.size();
    poly ret(0, n);
    fors(i, 0, n-1) ret[i] = A[i]?mod-A[i]:0;
    return ret;
}
friend poly operator-(poly&& A)
{
    int n = A.size();
    fors(i, 0, n-1) A[i] = A[i]?mod-A[i]:0;
    return A;
}

friend poly operator-(const poly& A, const poly& B)
{
    int n = max(A.size(), B.size());
    poly ret(0, n);
    fors(i, 0, n-1) ret[i] = (mod+A[i]-B[i])%mod;
    ret.adjust();
    return ret;
}
friend poly operator-(poly&& A, const poly& B)
{
    int n = B.size();
    fors(i, 0, n-1) A[i] = (mod+A[i]-B[i])%mod;
    A.adjust();
    return A;
}

friend poly operator*(ll x, const poly& B)
{
    poly ret(B); x %= mod;
    for(auto &i : ret) i = (i*x)%mod;
    ret.adjust();
    return ret;
}

friend poly operator*(const poly& A, const poly& B)
{
    poly ret(conv<mod, w>(A, B));
    // ACL : poly ret(atcoder::convolution<mod>(A,
    B));
    ret.adjust();
    return ret;
}
//friend poly operator/(const poly& A, const poly& B)
{return A*inv(B);}

friend poly inv(const poly& A){return inv(A, T);}
friend poly inv(const poly& A, int t)
{
    assert(A[0] != 0);
    poly g = pow<mod>(A[0], mod-2);

    int st=1;
    while(st <= t)
    {
        st <<=1;
        g = (-A%st*g%st+2)*g%st;
    }
    g.adjust(t);
    return g;
}

friend poly diff(const poly& A)
{
    int n = A.size();
    poly ret(0, n-1);
    forr(i, n-1) ret[i-1] = i*A[i]%mod;
    return ret;
}
friend poly inte(const poly& A)
{
    static ll inv[T] = {0, };
    int n = A.size();
    poly ret(0, n+1);
    forr(i, n+1) if(inv[i] == 0) inv[i] = pow<mod>(i,
    mod-2);
    forr(i, n+1) ret[i] = inv[i]*A[i-1]%mod;
    return ret;
}

friend poly log(const poly& A){return log(A, T);}
friend poly log(const poly& A, int t)
{
```

```
    assert(A[0] == 1);
    poly ret = inte(diff(A) * inv(A, t)%t);
    ret.adjust(t);
    return ret;
}

friend poly exp(const poly& A){return exp(A, T);}
friend poly exp(const poly& A, int t)
{
    assert(A[0] == 0);

    poly g = 1;
    int st = 1;
    while(st < t)
    {
        st <<= 1;
        g = (A%st-log(g, st)+1)*g%st;
    }
    g.adjust(t);
    return g;
}

friend poly pow(const poly& A, ll b, ll t)
{
    poly ret(A); ret.adjust();
    if(ret.size() == 1)
    {
        ret[0] = pow<mod>(ret[0], b);
        ret.adjust(t);
        return ret;
    }

    ll idx = 0; while(ret[idx] == 0) idx++;
    if((__int128_t) idx * b >= t) return poly(0, t);

    ll c = ret[idx]; ll ic = pow<mod>(ret[idx],
    mod-2); poly g;
    int n = ret.size();
    fors(i, idx, n-1) g[i-idx] = ret[i]*ic%mod;
    g.resize(t-idx*b);

    g = exp(b * log(g, t-idx*b), t-idx*b);
    c = pow<mod>(c, b);

    ret = poly(0, t);
    fors(i, idx*b, t-1) ret[i] = g[i-idx*b] * c % mod;

    return ret;
}

//Only just Polynomial, not Qring
void rev()
{
    int n=size();
    poly& F = *this;
    for(int i=0; i<n/2; i++) std::swap(F[i],
    F[n-i-1]);
}
```

```
friend poly div_quot(poly F, poly G)
{
    F.adjust(); G.adjust();
    ll df = F.size(), dg = G.size();
    if(df < dg) return poly(0);

    F.rev(); G.rev();

    F.resize(df-dg+1);
    F = F * inv(G, df-dg+1);
    F.resize(df-dg+1);

    F.rev();
    return F;
}
friend poly div_rem(poly F, poly G)
            {return F-G*div_quot(F, G);}

friend poly shift(const poly& F, ll c)
{
    ll n = F.size(); c %= mod;

    poly A(0, n); ll fac = 1;
    fors(i, 0, n-1) A[i] = F[i]*fac%mod, fac =
    fac*(i+1)%mod;
    A.rev();

    poly C(1, n);
    fors(i, 1, n-1) C[i] = C[i-1]*c%mod;

    ll facc = fac = pow<mod>(fac, mod-2)*n%mod;
    fore(i, n-1, 0) C[i] = C[i]*fac%mod, fac =
    fac*i%mod;

    poly B = C*A; B.resize(n);
    B.rev();

    fore(i, n-1, 0) B[i] = B[i]*facc%mod, facc =
    facc*i%mod;

    return B;
}
friend void calcG(vector<poly>& G, int i, int l, int
r, const vl& p)
{
    if(l == r)
    {
        ll g = p[l]?mod-p[l]:0;
        G[i] = vl({g, 1});
        return;
    }
    int mid = (l+r)/2;
    calcG(G, i*2, l, mid, p);
    calcG(G, i*2+1, mid+1, r, p);

    G[i] = G[i*2]*G[i*2+1];
}
```

```
        friend void eval(const vector<poly>& G, int i, int l,
        int r, poly&& F, vl& ret)
        {
            if(l == r)
            {
                ret[l] = F[0];
                return;
            }

            int mid=(l+r)/2;
            eval(G, i*2, l, mid, div_rem(F, G[i*2]), ret);
            eval(G, i*2+1, mid+1, r, div_rem(F, G[i*2+1]),
            ret);
        }
        friend vl multipoint_eval(const poly& A, const vl& B)
        {
            int m = B.size();
            vector<poly> G(4*m);
            calcG(G, 1, 0, m-1, B);

            vl ret(m, 0);
            eval(G, 1, 0, m-1, div_rem(A, G[1]), ret);

            return ret;
        }
    };
}  // namespace GMS

const ll mod = 998244353, w = 3, T = 1<<20;
using poly = GMS::Qring<T, mod, w>;
```

4 Linear Algebra

4.1 Matrix Multiplication

k-i-j 순서가 가장 Cache-Friendly 함.

5 Geometry

5.1 Mindset

Time Complexity: $\mathcal{O}(N?)$

```
using pii=pair<int,int>;
pii operator+(pii A, pii B){return {A.fi+B.fi, A.se+B.se};}
pii operator-(pii A, pii B){return {A.fi-B.fi, A.se-B.se};}
ll operator*(pii A, pii B){return
(ll)A.fi*B.fi+(ll)A.se*B.se;} // inner product
ll operator/(pii A, pii B){return
(ll)A.fi*B.se-(ll)A.se*B.fi;} // outer product

// 각도 정렬 (O = pii(0, 0))
sort(P+1, P+1+n, [](pii A, pii B){return
(A<O)!=(B<O)?l<r:l/r>0;});

// 선분 : pair<pii A, pii B> -> 2D-vector B from A
// 선분 교차 판정
bool isintersect(const pii &a, const pii &b, const pii &u,
const pii &v){
  if( b/v != 0 ) return sign((u-a)/b) * sign((u+v-a)/b) <= 0
  && sign((a-u)/v) * sign((a+b-u)/v) <= 0;
  else return (a-u)/v == 0 && (0 <= v*(a-u) && v*(a-u) <= v*v
  || 0 <= b*(u-a) && b*(u-a) <= b*b);
}
```

6 Greedy

7 DP

8 String

8.1 F, Z, M, SA(Suffix Array), LCP(Longest Common Prefix)

Usage: For string s(1-indexed) of length N;

$$
\begin{aligned}
\texttt{F[i]} &= \text{maximum } k < i & &\text{s.t. } s[1\ldots k] = s[i-k+1\ldots i]\\
\texttt{Z[i]} &= \text{maximum } k & &\text{s.t. } s[1\ldots k] = s[i\ldots i+k-1]\\
\texttt{M[i]} &= \text{maximum } k & &\text{s.t. } s[i-k+1\ldots i+k-1] \text{ is palindrom.}\\
\texttt{SA[i]} &= k & &\text{s.t. } s[k\ldots N] \text{ is the } i^{th} \text{ smallest of}\\
& & &\{s[1\ldots N],\ s[2\ldots N],\ \cdots,\ s[N\ldots N]\}\\
\texttt{LCP[i]} &= \text{maximum } k & &\text{s.t. } s[SA[i-1]\ldots SA[i-1]+k-1]\\
& & &= s[SA[i]\ldots SA[i]+k-1]
\end{aligned}
$$

Time Complexity: $\mathcal{O}(N), \mathcal{O}(N), \mathcal{O}(N), \mathcal{O}(N\log N), \mathcal{O}(N)$, respectively

```
const int N = 1e5+7;
char s[N];

int F[N], Z[N], M[N];
int sa[N]; int ord[N], tmp[N], cnt[N];
int lcp[N];
int main()
{
    scanf("%s", s+1);
    int n = strlen(s+1);

    // KMP - fail function
    {
        F[1] = 0; int j = 0;
        for(int i=2; i<=n;i++)
        {
            while(j > 0 and s[i] != s[j+1]) j = F[j];
            F[i] = j+=(s[i] == s[j+1]);
        }
    }

    //Z - Z array
    {
        Z[1] = n; int j = 1, r = 0;
        for(int i=2; i<=n; i++)
        {
            Z[i] = i<j+r?min(Z[i-j+1], j+r-i):0;
            while(s[1+Z[i]] == s[i+Z[i]]) Z[i]++;
            if(j+r < i+Z[i]) j = i, r = Z[i];
        }
    }

    //Manacher - M array
```

```
{
    M[1] = 0; int j = 1, r = 0;
    for(int i=2; i<=n; i++)
    {
        M[i] = i<j+r?min(M[2*j-i], j+r-i):0;
        while(1 <= i-M[i]-1 && i+M[i]+1 <= n
               && s[i-M[i]-1] == s[i+M[i]+1]) M[i]++;
        if(j+r < i+M[i]) j = i, r = M[i];
    }
}

//Suffix Array - SA
{
    int t = 1; ord[n+1] = 0; tmp[0] = 0; sa[0] = 0;

    auto cmp = [&t, &n](int i,int j)
    {
           return ord[i] == ord[j]
                  ?ord[min(i+t, n+1)]<ord[min(j+t, n+1)]
                  :ord[i]<ord[j];
    };

    forr(i, n) ord[i] = s[i], sa[i] = i;
    sort(sa+1, sa+1+n, [](int i,int j){return ord[i]
    <ord[j];});

    forr(i, n)  tmp[sa[i]] = tmp[sa[i-1]] + (ord[sa[i-1]]
    <ord[sa[i]]);
    swap(tmp, ord);

    while(t < n)
    {
        fors(i, 0, n) cnt[i] = 0;
        forr(i, n) cnt[ord[min(i+t, n+1)]]++;
        forr(i, n) cnt[i] += cnt[i-1];
        fore(i, n, 1) tmp[cnt[ord[min(i+t, n+1)]]--] = i;

        fors(i, 0, n) cnt[i] = 0;
        forr(i, n) cnt[ord[i]]++;
        forr(i, n) cnt[i] += cnt[i-1];
        fore(i, n, 1) sa[cnt[ord[tmp[i]]]--] = tmp[i];

        forr(i, n) tmp[sa[i]] = tmp[sa[i-1]] +
        cmp(sa[i-1], sa[i]);
        swap(ord, tmp);

        t<<=1;
        if(ord[sa[n]] == n) break;
    }
}

//LCP array
{
    int k = 0;
    forr(i, n) if(ord[i] != 1)
    {
        int j = sa[ord[i]-1];
        while(s[i+k] == s[j+k]) k++;
            lcp[ord[i]] = k;

            if(k > 0) k--;
        }
    }

    printf("\nF : "); forr(i, n) printf("%d ", F[i]);
    printf("\nZ : "); forr(i, n) printf("%d ", Z[i]);
    printf("\nM : "); forr(i, n) printf("%d ", M[i]);
    printf("\nSA : "); forr(i, n) printf("%d ", sa[i]);
    printf("\nLCP : x "); fors(i, 2, n) printf("%d ", lcp[i]);


    printf("\n"); forr(i, n) printf("%s\n", s+sa[i]);
}
```

9 Graph

9.1 SCC - Tarjan Algorithm

Usage: `scn[i]` : SCC number of node i, `nscc` : the number of SCCs

```
vi adj[N];
int in[N], c = 0;
stack<int> p;
bool fin[N]; int scn[N], nscc = 0;

int dfs(int s)
{
    in[s] = ++c;
    p.push(s);

    int m = c;
    for(auto i : adj[s])
    {
        if(in[i] == 0) m = min(m, dfs(i));
        else if(!fin[i]) m = min(m, in[i]);
    }

    if(m == in[s])
    {
        nscc++;
        while(p.top() != s)
        {
            int i = p.top(); p.pop();
            scn[i] = nscc; fin[i] = true;
        }
        p.pop();
        scn[s] = nscc; fin[s] = true;
    }
    return m;
}

forr(i, n) if(!fin[i]) dfs(i);
```

9.2 Bipartite Matching - with DFS

Usage: Let's say that graph is bipartite. And Let's say that one group is A, and the other graph is B. $|A| = N$, $|B| = M$. `matching(c = s)` : add one matching from $s \in A$. If successfully matched, return true; otherwise return false. `selby[i] =` store $s \in A$, s.t. $i \in B$ is matched

with s.
(e.g.) `forr(i, n) ans += matching(c=i);`

Time Complexity: $\mathcal{O}(VE)$

```
vector<int> sideadj[N];
int selby[M];
int chk[M], c;
bool matching(int s)
{
    for(auto i : sideadj[s])
    {
        if(chk[i] == c) continue;
        chk[i] = c;

        if(selby[i] and !matching(selby[i])) continue;

        selby[i] = s;
        return true;
    }
    return false;
}
```

9.2.1 Minimum Vertex Cover on Bipartite Graph(Kőnig's Therorem)

On bipartite graph,

$$|\texttt{Minimum Vertex Cover}| = |\texttt{Maximum Matching}|$$

To find Minimum Vertex Cover, (Should be added.)

9.2.2 Maximum Independent Set on Bipartite Graph

On bipartite graph,

$$|\texttt{Maximum Independent Set}| = |V| - |\texttt{Maximum Matching}|$$

* Note : Complement of the Vertex Cover is the Independent Set.

9.2.3 Minimum Path Cover on DAG

Let's think about the bipartite graph, with vertex set A and B, satisfying follow property:

- If there's edge from node i to node j on DAG, then there's edge connecting i^{th} node of A and j^{th} node of B, and vice versa.

Then following holds:
$|\texttt{Minimum Path Cover of DAG}| = |\texttt{Maximum Matching on Bipartite Graph}|$

9.2.4 Maximum Antichain on DAG(Dilworth's Theorem)

On DAG,

$$|\texttt{Minimum Path Cover}| = |\texttt{Maximum Antichain}|$$

9.3 Network Flow - Dinic

Usage: Construct graph with `connect(from, to, capacity, isDirected);`. Find the flow from S to T with `flow(S, T);`.

Time Complexity: $\mathcal{O}(V^2E)$, but it works like magic.

```
struct Edge
{
    int to, cap, now;
    Edge* rev;
    Edge(int to,int cap):to(to), cap(cap), now(0){}
    int left(){return cap - now;}
    void flow(int f){now += f; rev->now -= f;}
    void reset(){now = 0;}
};
vector<Edge*> adj[N];

int lv[N]; bool chk[N];
bool bfs(int S, int T)
{
    queue<int> q;

    q.push(S); lv[S] = 0; chk[S] = true;

    while(!q.empty())
    {
        int s = q.front(); q.pop();
        for(auto i : adj[s])
        {
            if(i->left() and !chk[i->to])
            {
                lv[i->to] = lv[s]+1; chk[i->to] = true;
                q.push(i->to);
            }
        }
    }
    return chk[T];
}

Edge* hist[N]; int last[N];
bool dfs(int s, int T)
{
    if(s == T) return true;

    for(int &j=last[s]; j < adj[s].size(); j++)
    {
        int i = adj[s][j]->to;
        if(adj[s][j]->left() == 0 or lv[i] != lv[s]+1)
        continue;
        hist[i] = adj[s][j];

        if(dfs(i, T)) return true;
    }
    return false;
}

ll flow(int S,int T)
{
    ll ans = 0;
    while(bfs(S, T))
    {
        while(dfs(S, T))
        {
            int m = 2e9;

            int now = T;
            while(S != now)
            {
```

```
            m = min(m, hist[now]->left());
            now = hist[now]->rev->to;
        }
        now = T;
        while(S != now)
            hist[now]->flow(m), now = hist[now]->rev->to;
        ans += m;
    }

    memset(last, 0, sizeof last);
    memset(chk, 0, sizeof chk);
  }
  return ans;
}
// isDir : isDirected => 양방향 간선이면 false
void connect(int from, int to, int cap, bool isDir = true)
{
    Edge *fw, *bw;
    fw = new Edge(to, cap);
    bw = new Edge(from, !isDir ? cap : 0);
    fw->rev = bw; bw->rev = fw;
    adj[from].push_back(fw);
    adj[to].push_back(bw);
}
```

9.4 MCMF - with SPFA

Usage: Construct graph with `connect(from, to, capacity, cost);`. Find the maximum flow and corresponding minimum cost from S to T with `flow(S, T);`.

Time Complexity: $\mathcal{O}(VEf)$, but it works like magic.

```
struct Edge
{
    int to, cap, now;
    ll cost;
    Edge* rev;
    Edge(int to,int cap, ll cost)
        :to(to), cap(cap), now(0), cost(cost){}
    int left(){return cap - now;}
    ll flow(int f)
        {now += f; rev->now -= f; return cost * f;}
    void reset(){now = 0;}
};
vector<Edge*> adj[N];
Edge* hist[N]; ll dist[N]; bool inQueue[N], chk[N];
bool spfa(int s, int t)
{
    memset(dist, 0, sizeof(dist));
    memset(chk, 0, sizeof(chk)); chk[s] = true;

    queue<int> q;
    memset(inQueue, 0, sizeof(inQueue));
    q.push(s); inQueue[s] = true;

    while(!q.empty())
    {
        int now = q.front();
        q.pop(); inQueue[now] = false;
        for(auto e : adj[now])
        {
            int next = e->to;
            if(e->left() > 0 and
                  (chk[next] == false
                         or dist[next] > dist[now] + e->cost))
            {
                chk[next] = true;
                dist[next] = dist[now] + e->cost;
                hist[next] = e;
                if(!inQueue[next])
                    q.push(next), inQueue[next] = true;
            }
        }
    }
    return chk[t];
}
// cost가 들어가면 항상 단방향만 가능하다. (양방향 : 2번
connect)
void connect(int from, int to, int cap, ll cost)
{
    Edge *fw, *bw;
    fw = new Edge(to, cap, cost);
    bw = new Edge(from, 0, -cost);
    fw->rev = bw; bw->rev = fw;
    adj[from].push_back(fw);
    adj[to].push_back(bw);
}
//maximum matching & minimum cost
pair<ll, ll> flow(int S,int T)
{
    ll ans = 0; ll cost = 0;
    while(spfa(S, T))
    {
        int m = 2e9;

        int now = T;
        while(S != now)
        {
            m = min(m, hist[now]->left());
            now = hist[now]->rev->to;
        }
        now = T;
        while(S != now)
        {
            cost += hist[now]->flow(m);
            now = hist[now]->rev->to;
        }
        ans += m;
    }
    return {ans, cost};
}
```

10 Tree

10.1 HLD(Heavy Light Decomposition)

```
BOJ 트리와 쿼리 1
```

```
//Segment Tree 코드
const int N = 1e5+7;

vi adj[N]; int par[N]; int sz[N]; int d[N];
void dfs1(int s)
{
    sz[s] = 1;
    for(int i=0;i<adj[s].size();i++)
        if(adj[s][i] == par[s])
            {adj[s].erase(adj[s].begin() + i); break;}
    for(auto &i : adj[s])
    {
        par[i] = s; d[i] = d[s] + 1;
        dfs1(i);
        sz[s] += sz[i];
        if(sz[i] > sz[adj[s][0]]) swap(adj[s][0], i);
    }
}
int in[N], c; int top[N];
void dfs2(int s)
{
    //printf("%d\n", s);
    in[s] = ++c;
    for(auto i : adj[s])
    {
        if(i == adj[s][0])
            top[i] = top[s];
        else top[i] = i;

        dfs2(i);
    }
}

Node *root;
int query(int a,int b)
{
    int ans = 0;
    while(top[a] != top[b])
    {
        if(d[top[a]] > d[top[b]]) swap(a, b);
        ans = max(ans, query(root, in[top[b]], in[b]));
        b = par[top[b]];
    }

    if(d[a] > d[b]) swap(a, b);
    ans = max(ans, query(root, in[a]+1, in[b]));

    return ans;
}
map<pii, int> m;
int arr[N]; pii edge[N];
int main()
{
    getint(n);
    forr(i, n-1)
    {
        intab; adj[a].pb(b); adj[b].pb(a);
        getint(c);
        m[{a,b}] = m[{b, a}] = c;
        edge[i] = {a,b};
    }

    dfs1(1);
    dfs2(1);
    forr(i, n) arr[in[i]] = m[{par[i], i}];
    root = new Node(1, n); init(root, arr);

    getint(Q);
    while(Q--)
    {
        getint(q);
        if(q == 1)
        {
            getint(i); getint(c);
            int a = edge[i].fi;
            int b = edge[i].se;
            if(par[b] == a) a = b;

            update(root, in[a], c, true);
        }
        if(q == 2)
        {
            intab;
            printf("%d\n", query(a, b));
        }
    }
}
```

11 Data Structure

11.1 PBDS - Policy-Based Data Structure

Time Complexity: Equivalent to std::set

```
#include<ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

template<typename T>
using indexed_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

indexed_set<int> s;
s.insert(3); s.insert(2); s.insert(3);
s.insert(9); s.insert(7);
//2 3 7 9

s.insert(5); //2 3 5 7 9
s.erase(5); //2 3 7 9

auto x = s.find_by_order(2); // *x : 7

s.order_of_key(6) // 2
s.order_of_key(7) // 2
s.order_of_key(8) // 3
```

11.2 rope

11.3 Union and Find - Queue Undoing

Time Complexity: $\mathcal{O}(\log^2 N)$

```cpp
struct dsu_pb
{
    const int N;
    vi par; stack<pair<pii, pii> > s;

    dsu_pb(int N):N(N), par(N)
    {
        fors(i, 0, N-1) par[i] = -1;
    }
    int root(int i)
    {
        if(par[i] < 0) return i;
        return root(par[i]);
    }
    bool join(int i, int j)
    {
        i = root(i); j = root(j);
        s.push({{i, par[i]}, {j, par[j]}});

        if(i == j) return false;

        if(-par[i] < -par[j]) swap(i, j);

        par[i] += par[j]; par[j] = i;
        return true;
    }

protected:
    void unjoin()
    {
        assert(!s.empty());
        auto [i, j] = s.top(); s.pop();

        par[i.fi] = i.se;
        par[j.fi] = j.se;
    }
};

struct dsu_pf : public dsu_pb
{
    vector<pair<bool, pii> > st;          // fi==0 -> B type,
    fi==1 -> A type
    vector<pair<bool, pii> > tmp[2];

    int A=0, B=0;

    dsu_pf(int N):dsu_pb(N){}

    bool join(int i, int j)
    {
        st.pb({0, {i, j}}); B++;
        return dsu_pb::join(i, j);
    }
    void pop_front()
    {
        assert(!st.empty());
        if(A == 0)
        {
            forr(i, B) unjoin();

            A = B; B = 0;
            reverse(all(st));
            for(auto &[b, p]:st) b = 1, dsu_pb::join(p.fi,
            p.se);
        }
        else if(st.back().fi == false)
        {
            tmp[st.back().fi].pb(st.back()); st.pop_back();
            unjoin();

            while(tmp[0].size() != tmp[1].size() and
            (unsigned) A != tmp[1].size())
            {
                tmp[st.back().fi].pb(st.back());
                st.pop_back();
                unjoin();
            }

            for(auto i:{0, 1}) reverse(all(tmp[i]));
            for(auto i:{0, 1}) for(auto v:tmp[i])
                st.pb(v), dsu_pb::join(v.se.fi, v.se.se);

            tmp[0].clear(); tmp[1].clear();
        }

        A--; st.pop_back();
        unjoin();
    }
};
```

11.4 Segment Tree Generalization

Time Complexity: $\mathcal{O}(\log N)$

```cpp
namespace GMS
{
    template<typename D, D (*join)(D,D), D _e>
    class Segtree
    {
        class Node
        {
            Node *l, *r;
            int s,e; D v;
        public:
            Node(int s, int e) :l(0), r(0), s(s), e(e), v(_e)
            {};
            ~Node(){delete l; delete r;}

            template<typename Dini>
            friend void init(Node* node, Dini arr[] = NULL)
            {
                int s = node->s, e=node->e, mid=(s+e)/2;
                if(s == e)
                {
                    node->v = D(arr?arr[s]:_e);
                    return;
                }
```

```
            node->l = new Node(s, mid);
            init(node->l, arr);
            node->r = new Node(mid+1, e);
            init(node->r, arr);

            node->v = join(node->l->v, node->r->v);
        }
        friend D _query(Node* node, int a, int b)
        {
            int s=node->s, e=node->e;
            if(a <= s and e <= b) return node->v;
            if(b < s or e < a) return _e;

            return join(_query(node->l, a, b),
            _query(node->r, a, b));
        }
        friend void _update
                (Node* node, int i, function<D(D)> upd)
        {
            int s=node->s, e=node->e;
            if(i < s or e < i) return;
            if(s == e)
            {
                node->v = upd(node->v);
                return;
            }

            _update(node->l, i, upd);
            _update(node->r, i, upd);

            node->v = join(node->l->v, node->r->v);
        }
    };

    Node *root;
public:

    template<typename Dini>
    Segtree(int s,int e, Dini arr[] = NULL)
    {
        root = new Node(s, e);
        init(root, arr);
    }
    ~Segtree(){delete root;}
    D query(int s, int e)
                    {return _query(root, s, e);}
    void update(int i, function<D(D)> upd)
                    {_update(root, i, upd);}
};


template<typename D, D (*join)(D,D), D _e, typename L, D
(*apply)(D, L, int), L (*give)(L, L), L _l>
class LZSegtree
{
    class Node
    {
        Node *l, *r;
        int s,e;
        D v; L lz;

        void prop()
        {
            v = apply(v, lz, e-s+1);
            if(l) l->lz = give(l->lz, lz);
            if(r) r->lz = give(r->lz, lz);

            lz = _l;
        }

    public:
        Node(int s, int e)
              :l(0), r(0), s(s), e(e), v(_e), lz(_l){};
        ~Node(){delete l; delete r;}

        template<typename Dini>
        friend void init(Node* node, Dini arr[] = NULL)
        {
            int s = node->s, e=node->e, mid=(s+e)/2;
            if(s == e)
            {
                node->v = D(arr?arr[s]:_e);
                return;
            }

            node->l = new Node(s, mid);
            init(node->l, arr);
            node->r = new Node(mid+1, e);
            init(node->r, arr);

            node->v = join(node->l->v, node->r->v);
        }
        friend D _query(Node* node, int a, int b)
        {
            node->prop();
            int s=node->s, e=node->e;
            if(a <= s and e <= b) return node->v;
            if(b < s or e < a) return _e;

            return join(_query(node->l, a, b),
            _query(node->r, a, b));
        }
        friend void _update
           (Node* node, int a, int b, function<L(L)> upd)
        {
            node->prop();
            int s=node->s, e=node->e;
            if(b < s or e < a) return;
            if(a <= s and e <= b)
            {
                node->lz = upd(node->lz);
                node->prop();
                return;
            }

            _update(node->l, a, b, upd);
```

```
                _update(node->r, a, b, upd);

                node->v = join(node->l->v, node->r->v);
            }
        };

        Node *root;
    public:

        template<typename Dini>
        LZSegtree(int s,int e, Dini arr[] = NULL)
        {
            root = new Node(s, e);
            init(root, arr);
        }
        ~LZSegtree(){delete root;}
        D query(int s, int e){return _query(root, s, e);}
        void update(int s, int e, function<L(L)> upd)
        {_update(root, s, e, upd);}
    };
} // namespace GMS

////////////////////////////////////////////////////////
#define data _data

struct data
{
    int m, m_cnt;
    constexpr data(int m):m(m), m_cnt(1){}
    constexpr data(int m, int m_cnt):m(m), m_cnt(m_cnt){}
};
data join(data A, data B)
{
    if(A.m == B.m) return data(A.m, A.m_cnt+B.m_cnt);
    if(A.m < B.m) return A;
    else return B;
}
data apply(data A, int lz, int len)
                     {return {A.m+lz, A.m_cnt};}
int give(int a, int b){return a+b;}

using Seg = GMS::LZSegtree<data, join, {(int)1e9, 0}, int,
apply, give, 0>;
```

11.5 Li-Chao Tree

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

#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()

#define getint(n) int n; scanf("%d%*c", &n)
#define getll(n)  ll n; scanf("%lld%*c", &n)
#define inta int a; scanf("%d%*c", &a)
#define intab int a,b; scanf("%d%*c%d%*c", &a,&b)

#define forr(i, n) for(int i=1; i <= (n); i++)
#define fors(i, s, e) for(int i = (s); i<=(e); i++)
#define fore(i, e, s) for(int i = (e); i >= s; i--)
#define pb push_back
#define fi first
#define se second

using namespace std;

using ll = long long;
using pii = pair<int,int>; using pll = pair<ll, ll>;
using vi = vector<int>; using vii = vector<pii>;
struct Line
{
    ll a=0, b=(ll)2e18+7;
    ll operator()(ll x){return a*x+b;}
    Line():a(0),b((ll)2e18+7){}
    Line(ll a, ll b):a(a), b(b){}
};

ll middle(ll s, ll e){return (s+e+(ll)2e18)/2-(ll)1e18;}
struct Node
{
    Node *l=0, *r=0;
    Line v;
    Node():l(0), r(0), v(Line()){}
};
void insert(Node* node, Line v, ll l, ll r, ll s, ll e)
{
    ll mid=middle(s, e);
    if(e < l or r < s) return;
    if(s == e)
    {
        node->v = (node->v(s) < v(s))?node->v:v;
        return;
    }
    if(!node->l) node->l = new Node();
    if(!node->r) node->r = new Node();

    if(l <= s and e <= r)
    {
        if(node->v(s) >= v(s) and node->v(e) >= v(e)){node->v
        = v; return;}
        if(node->v(s) <= v(s) and node->v(e) <= v(e)) return;
        insert(node->l, v, l, r, s, mid);
        insert(node->r, v, l, r, mid+1, e);
    }
    else
    {
        insert(node->l, v, l, r, s, mid);
        insert(node->r, v, l, r, mid+1, e);
    }
}
ll query(Node* node, ll x, ll s, ll e)
{
    if(!node) return (ll)2e18+7;
    if(x < s or e < x) return (ll)2e18+7;
    if(s == e) return node->v(x);

    ll mid = middle(s, e);
```

```
    return min({query(node->l, x, s,mid), query(node->r, x,
    mid+1, e), node->v(x)});
}
// 전체 구간을 미리 고정해 놓아야 함.
// (const int L = -1e9, R = 1e9);
// Insert : insert(root, Line 객체, l, r, L, R);
// Query : query(root, x, L, R);
int main()
{
    Node *root = new Node();
    getint(n); getint(Q);
    forr(i, n)
    {
        getll(l); getll(r); getll(a); getll(b);
        insert(root, Line(a,b), l,r-1,(ll)-1e9-7, (ll)1e9+7);
    }
    while(Q--)
    {
        getint(q);
        if(q == 0)
        {
            getll(l); getll(r); getll(a); getll(b);
            insert(root, Line(a,b), l,r-1,(ll)-1e9-7,
            (ll)1e9+7);
        }
        if(q == 1)
        {
            getll(x);
            ll ans = query(root, x,(ll)-1e9-7, (ll)1e9+7);
            if(ans == (ll)2e18+7) printf("INFINITY\n");
            else printf("%lld\n", ans);
        }
    }
}
```

11.6 Splay Tree

```
struct Node
{
    Node *p, *l, *r;
    int key, cnt;
    ll val; ll m, M, sum; ll lazy;
    bool flip;

    Node(int key, ll val):p(0),l(0), r(0), key(key), cnt(1),
    val(val), m(val), M(val), sum(val), lazy(0), flip(0){}

    void fix()
    {
        cnt = 1+(l?l->cnt:0)+(r?r->cnt:0);
        sum = val+(l?l->sum:0)+(r?r->sum:0);
        m = min({val, (l?l->m:inf), (r?r->m:inf)});
        M = max({val, (l?l->M:-1), (r?r->M:-1)});
    }
    void prop()
    {
        if(flip)
        {
            swap(l, r);
            if(l) l->flip = !l->flip;
            if(r) r->flip = !r->flip;

            flip = false;
        }
        if(lazy)
        {
            val += lazy; sum += cnt * lazy;
            if(l) l->lazy += lazy;
            if(r) r->lazy += lazy;

            lazy = 0;
        }
    }

} *root;
struct SplayTree
{
    Node *root = NULL, *rp = NULL;

    SplayTree(){}
    SplayTree(Node *rt)
    {
        if(!rt) return;

        root = rt;
        rp = rt->p;
    }

    void mop(Node *node)
    {
        if(node == root) node->prop();
        else mop(node->p);

        if(node->l) node->l->prop();
        if(node->r) node->r->prop();
    }

    void rotate(Node *node)
    {
        if(!root) return;

        if(node->p == rp) return;
        if(node->p->l == node)
        {
            Node *p = node->p, *g = p->p;
            Node *a = node->l, *b = node->r, *c = p->r;

            p->l = b; if(b) b->p = p;
            p->r = c; if(c) c->p = p;

            node->l = a; if(a) a->p = node;
            node->r = p; p->p = node;

            node->p = g; if(g) (g->l == p?g->l:g->r) = node;

            p->fix(); node->fix();
```

```
        if(p == root) root = node;
    }
    else
    {
        Node *p = node->p, *g = p->p;
        Node *a = p->l, *b = node->l, *c = node->r;

        p->l = a; if(a) a->p = p;
        p->r = b; if(b) b->p = p;

        node->l = p; p->p = node;
        node->r = c; if(c) c->p = node;

        node->p = g; if(g) (g->l == p?g->l:g->r) = node;

        p->fix(); node->fix();

        if(p == root) root = node;
    }

}

void splay(Node* node)
{
    if(!root) return;
    assert(node); mop(node);

    while(node->p != rp)
    {
        Node *p, *g;
        p = node->p; g = p->p;

        if(g == rp) rotate(node);
        else if((p->l == node) == (g->l == p))
            rotate(p), rotate(node);
        else rotate(node), rotate(node);
    }

    root = node;
}

Node* insert(int key, ll val)
{
    if(!root)
    {
        root = new Node(key, val);
        return root;
    }
    else
    {
        Node *now = root;
        while(true)
        {
            if(now->key == key) return NULL;
            else if(now->key > key)
            {
                if(!now->l) break;
                now = now->l;
            }
            else
            {
                if(!now->r) break;
                now = now->r;
            }
        }

        Node *ret;
        if(now->key > key)
        {
            ret = now->l = new Node(key, val);
            now->l->p = now;
            splay(now->l);
        }
        else
        {
            ret = now->r = new Node(key, val);
            now->r->p = now;
            splay(now->r);
        }

        return ret;
    }
}

/*Node* find(int key)
{
    Node* now = root;
    if(!now) return NULL;
    while(true)
    {
        if(key == now->key) break;
        else if(key < now->key)
        {
            if(!now->l) break;
            now = now->l;
        }
        else
        {
            if(!now->r) break;
            now = now->r;
        }
    }

    splay(now);
    return key == now->key?now:NULL;
}

void erase(int key)
{
    if(!find(key)) return;
    if(root->l and root->r)
    {
        Node* e = root;
        root = root->l; root->p = NULL;

        Node* now = root;
```

```
        while(now->r) now = now->r;
        now->r = e->r;
        e->r->p = now;
        splay(now);

        delete e;
    }
    else if(root->l)
    {
        Node* now = root;
        root = root->l; root->p = NULL;
        delete now;
    }
    else if(root->r)
    {
        Node* now = root;
        root = root->r; root->p = NULL;
        delete now;
    }
    else
    {
        delete root;
        root = NULL;
    }
}*/

Node* find_kth(int k) // 0-indexed
{
    assert(root);
    assert(root->cnt > k);
    Node *now = root; now->prop();
    while(true)
    {
        while(now->l and now->l->cnt > k) now = now->l,
        now->prop();
        k -= now->l?now->l->cnt:0;

        if(k == 0) break;
        k--; now = now->r;
        now->prop();
    }

    splay(now);
    return now;
}
Node* gather(int s, int e)
{
    find_kth(e+1);
    SplayTree(root->l).find_kth(s-1);

    assert(root->l->r);
    return root->l->r;
}
void update(int i, int j, ll val)
{
    Node *node = gather(i, j);

    node->lazy += val; node->prop();
        node->p->fix(); node->p->p->fix();
    }
    void reverse(int i, int j)
    {
        Node *node = gather(i, j);
        node->flip = !node->flip;
    }

    void p_vals(int n){p_vals(root, 0, false, n);}
    void p_vals(Node* node, ll lz, bool flip, int n)
    {
        lz += node->lazy; flip ^= node->flip;
        if(!flip)
        {
            if(node->l) p_vals(node->l, lz, flip, n);
            if(!(node->key == 0 or node->key == n+1))
            printf("%lld ", node->val+lz);
            if(node->r) p_vals(node->r, lz, flip, n);
        }
        else
        {
            if(node->r) p_vals(node->r, lz, flip, n);
            if(!(node->key == 0 or node->key == n+1))
            printf("%lld ", node->val+lz);
            if(node->l) p_vals(node->l, lz, flip, n);
        }
    }
};
```

12 Numerical Analysis

13 Technic

14 Misc

14.1 Fast Input

Usage: Fast Input with fread. Do not use with scanf, cin, or other input function. Use `forr(i, n) read(arr[i]);` instead of `forr(i, n) scanf("%d", arr+i);`. Use `read(s+1)` instead of `scanf("%s", s+1);`.

```
#define getint(n) int n; read(n)
#define getll(n) ll n; read(n)
#define inta getint(a)
#define intab getint(a); getint(b)
char get()
{
    static char buf[100000], *S=buf, *T=buf;
    if(S == T)
    {
        S = buf;
        T = buf + fread(buf, 1, 100000, stdin);
        if(S == T) return EOF;
    }
    return *S++;
}
void read(int& n)
{
    n = 0;
    char c; bool neg = false;
```

```
    for(c = get(); c < '0'; c=get()) if(c=='-') neg = true;
    for(;c>='0';c=get()) n = n*10+c-'0';
    if(neg) n = -n;
}
void read(ll& n)
{
    n = 0;
    char c; bool neg = false;
    for(c = get(); c < '0'; c=get()) if(c=='-') neg = true;
    for(;c>='0';c=get()) n = n*10+c-'0';
    if(neg) n = -n;
}
int read(char s[])
{
    char c; int p = 0;
    while((c = get()) <= ' ');

    s[p++] = c;
    while((c = get()) >= ' ') s[p++] = c;
    s[p] = '\0';

    return p;
}
```

14.2 MT19937 Random Number

```
const long long rand_L = 1;
const long long rand_R = 10;
mt19937_64
rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dist(rand_L, rand_R);
auto generator = bind(dist, rng);
```

— Document end —