GMAC Cheatsheet

Universitas Gadjah Mada

Latest Update: PolyMod - 27/10/25:19.12

Math

1. PHI

```
const long double PHI = acos(-1);
```

2. RNG

```
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());
```

3. GCD

```
int gcd(int a, int b){ return b ? gcd(b, a % b) : a; }
```

4. Extended Euclid

```
ll euclid(ll a, ll b, ll &x, ll &y){
  if (!b) return x = 1, y = 0, a;
  ll d = euclid(b, a % b, y, x);
  return y -= a/b * x, d;
}
```

5. Fast Mod (Barrett Reduction)

```
using ull = unsigned long long;
using L = __uint128_t;
ull mod, m;
m = ull((L(1) << 64) / mod);
ull reduce(ull x) {
  ull q = ull((L(m) * x) >> 64);
  ull r = x - q * mod;
  return r >= mod ? r - mod : r;
}
```

6. Miller-Rabin Primality Test

```
bool prima(LL x, int k = 5) {
  if(x == 1) return 0;
  if(x == 2 || x == 3) return 1;
  if(x % 2 == 0) return 0;
  LL d = x - 1;
  while(~d&1) d >>= 1;
  while(k--) {
    LL a = rng() % (x-4) + 2;
    LL y = pangkat(a,d,x);
    if(y == 1) continue;
    for(LL i = d; i != x-1 && y != 1 && y != x-1; i <<= 1) {
      y = mul(y,y,x);
    }
    if(y != x-1) return 0;
  }
  return 1;
}
```

7. Pangkat

```
ll pang(ll x, ll y){
    if(!y) return 1;
    return pang(x * x % MOD, y / 2) * (y & 1 ? x : 1ll) % MOD;
}
```

8. Number Theoritic Transform (NTT)

```
const ll MOD = 998244353, root = 3;
void resz(vector<ll> &p){ //Resize jadi 2^n
    p.resize(1 << (32 - __builtin_clz(p.size())));
}

void ntt(vector<ll> &p){ //Hasil NTT dari vector
    ll n = p.size(), L = 31 - __builtin_clz(n);
    static vector<ll> rt(2, 1);
    for(static ll k = 2, s = 2; k < n; k <<= 1ll, s++){
        rt.resize(n);
        ll z[] = {1, pang(root, MOD >> s)};
        for(ll i = k; i < 2*k; i++) rt[i] = rt[i / 2] * z[i & 1]
     % MOD;
  }
  vector<ll> rev(n);
  for(ll i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L)
     / 2;
  for(ll i = 0; i < n; i++) if (i < rev[i]) swap(p[i], p[rev[i]])
     ;
  for (ll k = 1; k < n; k *= 2)
    for (ll i = 0; i < n; i += 2ll * k) for(ll j = 0; j < k; j++)
      {
      ll z = rt[j + k] * p[i + j + k] % MOD, &ai = p[i + j];
      p[i + j + k] = ai - z + (z > ai ? MOD : 0);
      ai += (ai + z >= MOD ? z - MOD : z);
    }
}

void intt(vector<ll> &p){
    ntt(p);
    reverse(p.begin() + 1, p.end());
    ll inv = pang(p.size(), MOD - 2);
    for(ll &x: p) x = x * inv % MOD;
}

vector<ll> conv(const vector<ll> &a, const vector<ll> &b){
    if(a.empty() || b.empty()) return {};
    ll s = a.size() + b.size() - 1, n = 1 << (32 - __builtin_clz(
     s));
    ll inv = pang(n, MOD - 2);
    vector<ll> L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    for(ll i = 0; i < n; i++) out[-i & (n - 1)] = L[i] * R[i] %
     MOD * inv % MOD;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

// Masukan 2 Vector of anysize, Output Vector at 2^n *size*

9. FFT, Convolution Mod, Convolution

```
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef complex<double> C;
typedef vector<double> vd;

void fft(vector<C>& a) {
    int n = (int)a.size();
    int L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);  // (^ 10% faster if double)

    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        for (int i = k; i < 2 * k; ++i)
            rt[i] = R[i] = (i & 1) ? R[i / 2] * x : R[i / 2];
    }

    vi 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 += 2 * k)
            for (int j = 0; j < k; ++j) {
                // C z = rt[j+k] * a[i+j+k]; //in (25% faster if
     hand-rolled)
                auto x = (double *)&rt[j + k]; //ex
                auto y = (double *)&a[i + j + k]; //ex
                C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1]
      * y[0]); //ex
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
}

//Saran dari chatgpt line resize diganti 2*k
typedef vector<ll> vl;

template<int M>
vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res((int)a.size() + (int)b.size() - 1);
```

```
    int B = 32 - __builtin_clz((int)res.size());
    int n = 1 << B;
    int cut = (int)sqrt(M);

    vector<C> L(n), R(n), outs(n), outl(n);

    for (int i = 0; i < (int)a.size(); ++i)
        L[i] = C((int)a[i] / cut, (int)a[i] % cut);

    for (int i = 0; i < (int)b.size(); ++i)
        R[i] = C((int)b[i] / cut, (int)b[i] % cut);

    fft(L);
    fft(R);

    for (int i = 0; i < n; ++i) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = {imag(outs[j]), -real(outs[j])};
    }

    fft(outl);
    fft(outs);

    for (int i = 0; i < (int)res.size(); ++i) {
        ll av = (ll)(real(outl[i]) + 0.5);
        ll cv = (ll)(imag(outs[i]) + 0.5);
        ll bv = (ll)(imag(outl[i]) + 0.5) + (ll)(real(outs[i]) +
 0.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }

    return res;
}
```

Normal conv

```
typedef long long ll;

typedef std::complex<double> C;
typedef std::vector<double> vd;

vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};

    vd res(a.size() + b.size() - 1);
    int L = 32 - __builtin_clz((int)res.size());
    int n = 1 << L;

    std::vector<C> in(n), out(n);
    std::copy(a.begin(), a.end(), in.begin());
    for (int i = 0; i < (int)b.size(); ++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)] - std::conj(in[i]);

    fft(out);
    for (int i = 0; i < (int)res.size(); ++i)
        res[i] = std::imag(out[i]) / (4 * n);

    return res;
}
```

10. BigInt

```
const int BASE_LENGTH = 2;
const int BASE = (int) pow(10, BASE_LENGTH);
const int MAX_LENGTH = 500;

string int_to_string(int i, int width, bool zero) {
    string res = "";
    while (width--) {
        if (!zero && i == 0) return res;
        res = (char)(i%10 + '0') + res;
        i /= 10;
    }
    return res;
}

struct bigint {
    int len, s[MAX_LENGTH];

    bigint() {
        memset(s, 0, sizeof(s));
        len = 1;
    }

    bigint(unsigned long long num) {
        len = 0;
        while (num >= BASE) {
            s[len] = num % BASE;
            num /= BASE;
            len ++;
        }
        s[len++] = num;
    }

    bigint(const char* num) {
        int l = strlen(num);
        len = l/BASE_LENGTH;
        if (l % BASE_LENGTH) len++;
        int index = 0;
        for (int i = l - 1; i >= 0; i -= BASE_LENGTH) {
            int tmp = 0;
            int k = i - BASE_LENGTH + 1;
            if (k < 0) k = 0;
            for (int j = k; j <= i; j++) {
                tmp = tmp*10 + num[j] - '0';
            }
            s[index++] = tmp;
        }
    }

    void clean() {
        while(len > 1 && !s[len-1]) len--;
    }

    string str() const {
        string ret = "";
        if (len == 1 && !s[0]) return "0";
        for(int i = 0; i < len; i++) {
            if (i == 0) {
                ret += int_to_string(s[len - i - 1], BASE_LENGTH,
  false);
            } else {
                ret += int_to_string(s[len - i - 1], BASE_LENGTH,
  true);
            }
        }
        return ret;
    }

    unsigned long long ll() const {
        unsigned long long ret = 0;
        for(int i = len-1; i >= 0; i--) {
            ret *= BASE;
            ret += s[i];
        }
        return ret;
    }

    bigint operator + (const bigint& b) const {
        bigint c = b;
        while (c.len < len) c.s[c.len++] = 0;
        c.s[c.len++] = 0;
        bool r = 0;
        for (int i = 0; i < len || r; i++) {
            c.s[i] += (i<len)*s[i] + r;
            r = c.s[i] >= BASE;
            if (r) c.s[i] -= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator - (const bigint& b) const {
        if (operator < (b)) throw "cannot do subtract";
        bigint c = *this;
        bool r = 0;
        for (int i = 0; i < b.len || r; i++) {
            c.s[i] -= b.s[i];
            r = c.s[i] < 0;
            if (r) c.s[i] += BASE;
```

```
        }
        c.clean();
        return c;
    }

    bigint operator * (const bigint& b) const {
        bigint c;
        c.len = len + b.len;
        for(int i = 0; i < len; i++)
            for(int j = 0; j < b.len; j++)
                c.s[i+j] += s[i] * b.s[j];
        for(int i = 0; i < c.len-1; i++){
            c.s[i+1] += c.s[i] / BASE;
            c.s[i] %= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator / (const int b) const {
        bigint ret;
        int down = 0;
        for (int i = len - 1; i >= 0; i--) {
            ret.s[i] = (s[i] + down * BASE) / b;
            down = s[i] + down * BASE - ret.s[i] * b;
        }
        ret.len = len;
        ret.clean();
        return ret;
    }

    bool operator < (const bigint& b) const {
        if (len < b.len) return true;
        else if (len > b.len) return false;
        for (int i = 0; i < len; i++)
            if (s[i] < b.s[i]) return true;
            else if (s[i] > b.s[i]) return false;
        return false;
    }

    bool operator == (const bigint& b) const {
        return !(*this<b) && !(b<(*this));
    }

    bool operator > (const bigint& b) const {
        return b < *this;
    }
};
```

11. Matrix Operation

```
typedef double mat[MAX][MAX];

/* LU-Decomposition
Decomposes an n x n matrix A into LU.*/
void LU_decompose(int n, mat A, mat LU) {
  REP(i, n) REP(j, N) LU[i][j] = A[i][j];
  for (int i = 0; i < n; i++) {
    for (int j = i+1; j < n; j++) {
    double temp = LU[j][i]/LU[i][i];
    for (int k = i; k < n; k++)
        LU[j][k] -= LU[i][k] * temp;
      LU[j][i] = temp;
    }
  }
}

/*Back Substitution
Solves matrix equation Ux = b.*/
void back_substitute(int n, mat U, double x[], double b[]){
  for (int i = n-1; i >= 0; i--){
    x[i] = b[i];
    for (int j = i+1; j < n; j++)
      x[i] -= x[j]*U[i][j];

    x[i] /= U[i][i];
  }
}

/*Forward Substitution
Solves matrix equation Lx = b.*/
void forward_substitute(int n, mat L, double x[], double b[]){
  for (int i = 0; i < n; i++){
    x[i] = b[i];
    for (int j = 0; j < i; j++)
      x[i] -= x[j]*L[i][j];

/*Linear System
Solves matrix equation Ax = b.*/
void solve(int n, mat A, double x[], double b[]){
  mat LU;
  REP(i, n) REP(j, n) LU[i][j] = A[i][j];
  for (int i = 0; i < n; i++)
    LU[i][n] = b[i];
  for (int i = 0; i < n; i++){
    int pivot = i;
    for (int j = i+1; j < n; j++)
      if (fabs(LU[j][i]) > fabs(LU[pivot][i]))
        pivot = j;
    for (int j = i; j <= n; j++)
      swap(LU[i][j], LU[pivot][j]);
    for (int j = i+1; j < n; j++){
      double temp = LU[j][i]/LU[i][i];
      for (int k = i; k <= n; k++)
        LU[j][k] -= LU[i][k] * temp;
      LU[j][i] = temp;
    }
  }
  for (int i = 0; i < n; i++)
    b[i] = LU[i][n];
  back_substitute(n, LU, x, b);
}

/*Matrix Inverse
Compute the inverse Ac of matrix A.*/
void inverse(int n, mat A, mat Ac) {
  mat LU;
  LU_decompose(n, A, LU);
  double x[MAX], e[MAX], y[MAX];
  REP(j, n) {
    REP(i, n) e[i] = 0;
    e[j] = 1;
    forward_substitute(n, LU, y, e);
    back_substitute(n, LU, x, y);
    REP(i, n) Ac[i][j] = x[i];
  }
}

/*Matrix Determinant
Returns the determinant of matrix A.*/
double det(int n, mat A) {
  mat LU;
  LU_decompose(n, A, LU);
  double res = 1;
  REP(i, n) res *= LU[i][i];
  return res * (n % 2 ? -1 : 1);
}

/*
  * Find inverse using gaussian elimination:
    - Pad N more column, which represents identity matrix
    - Do gaussian elimination until original matrix become
     identity matrix
    - The padded matrix is now the inverse

  * Find determinant using gaussian elimination:
    - Do gaussian elimination. keep variable, say d = 1
    - if swap row operation, d = -1 * d
    - if multiply row by C, d = C * d
    - at the end, determinant is (product of main diagonal) / d
*/
```

12. CRT

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
pair<int, int> chinese_remainder_theorem(int x, int a, int y, int
      b) {
  int s, t;
  int d = extended_euclid(x, y, s, t);
  // printf("%lld %lld %lld %lld %lld\n",x,a,y,b,d);
  if (a % d != b % d) return make_pair(0, -1);
  int md = x * y;
  int res = (s * b) % md;
  res = (res * x) % md;
  int res2 = (t * a) % md;
  res2 = (res2 * y) % md;
  return make_pair(mod(res + res2, md) / d, x * y / d);
}
```

```
// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
pair<int, int> chinese_remainder_theorem(const vector<int> &x,
     const vector<int> &a) {
  pair<int, int> ret = make_pair(x[0], a[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = chinese_remainder_theorem(ret.first, ret.second, x[i],
     a[i]);
    swap(ret.first, ret.second);
    if (ret.first == -1) break;
  }

  return ret;
}
```

// Note: Given $x = a_i$ *mod* m_i *for* $i = 1, \cdots, n$. *Let* $M = Prod(m_i)$ *and* $M_i = M/m_i$, *construct* $y_i = M_i^{-1}$ *(mod* m_i*) then* $x = \left(\sum_{i=1}^{k} a_i M_i y_i\right) \pmod M$

Polynomial Operation

1. Polynomial Inverse
   $\frac{1}{P(x)}$

```
vector<ll> pinv(vector<ll> p, ll n){
    vector<ll> inv(1, pang(p[0], MOD - 2));
    for(ll i = 2; i < 2 * n; i *= 2){
        vector<ll> cur(2 * i);
        for(ll j = 0; j < i; j++) cur[j] = j < (ll)p.size() ? p[j
     ] : 0;
        inv.resize(2 * i);
        ntt(inv), ntt(cur);
        ll invSz = pang(2 * i, MOD - 2);
        for (ll j = 0; j < 2 * i; j++) inv[j] = inv[j] * (2 - cur
     [j] * inv[j] % MOD + MOD) % MOD * invSz % MOD;
        reverse(inv.begin() + 1, inv.begin() + 2 * i);
        ntt(inv);
        fill(inv.begin() + i, inv.begin() + 2 * i, 0);
    }
    return {inv.begin(), inv.begin() + n};
}
```

// n = sampai elemen ke x^n *aja*

2. Polynomial Logarithm
   Menghitung $\ln(P(x))$

```
    //n = 2^...
vector<ll> plog(vector<ll> p, ll n){
    p.resize(n);
    p[0] = 1;
    auto q = pinv(p,n);
    for(ll i = 0; i < n - 1; i++){
        p[i] = (i + 1) * p[i + 1] % MOD;
    }
    p = conv(p,q);
    for(ll i = n - 1; i >= 1; i--){
        p[i] = p[i - 1] * pang(i, MOD - 2) % MOD;
    }
    p[0] = 0;
    return p.resize(n), p;
}
```

// n harus power of 2

3. Polynomial Exponentiation
   $e^{P(x)}$

```
vector<ll> pexp(vector<ll> p, ll n){
    vector<ll> r(1, 1);
    p.resize(n);
    for(ll m = 2; m <= n; m *= 2){
        vector<ll> x = r;
        x = plog(x, m);
        for(ll i = 0; i < m; i++) x[i] = (p[i] - x[i] + MOD) %
     MOD;
        x[0] = (x[0] + 1) % MOD;
        r.resize(m);
        x.resize(m);
        r = conv(r,x);
        r.resize(m);
    }
    return r;
}
```

// n harus power of 2

4. Polynomial modulo

```
vector<ll> pmod(vector<ll> q, vector<ll> p){
  ll m = p.size(), n = q.size();
  if(n < m) return q;
  vector<ll> prev = p, qrev = q;
  reverse(prev.begin(), prev.end());
  reverse(qrev.begin(), qrev.end());
  prev = pinv(prev, n - m + 1);
  prev = conv(qrev, prev);
  prev.resize(n - m + 1);
  reverse(prev.begin(), prev.end());
  prev = conv(p, prev);
  vector<ll> res(m - 1, 0);
  for(int i = 0; i < m - 1; i++){
    res[i] = (((q[i] - prev[i]) % MOD + MOD) % MOD);
  }
  return res;
}
```

5. ANY Polynomial Function (Theory)
   Take n any power of 2. Misal yang mau kita hitung G(P(x)).
   Terus, $Q(x) = G(P(x)) \rightarrow P = G^{-1}(Q(x))$
   Generate F(Q) $= G^{-1}(Q(x)) - P$. Maka

$$Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} (\text{mod } x^{2^{k+1}})$$

   Return Q_n

Graph

1. DSU on Tree

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e5 + 1;

int st[maxn], ft[maxn], sz[maxn], col[maxn];
vector <int> euler;
vector <vector <int>> v;
int timer = 0;
void init(int now, int pr){
    st[now] = timer++;
    sz[now] = 1;
    for(auto p : v[now]){
        if(p == pr) continue;
        init(p, now);
        sz[now] += sz[p];
    }
    ft[now] = timer;
}

int cnt[maxn];
void dfs(int now, int pr, bool keep){
    int mx = -1, bigChild = -1;
    for(auto p : v[now])
       if(p != pr && sz[p] > mx)
          mx = sz[p], bigChild = p;
    for(auto p : v[now])
        if(p != pr && p != bigChild)
            dfs(p, now, 0);  // run a dfs on small childs and
     clear them from cnt
    if(bigChild != -1)
        dfs(bigChild, now, 1);  // bigChild marked as big and not
      cleared from cnt
    for(auto p : v[now])
        if(p != pr && p != bigChild)
            for(int i = st[p]; i < ft[p]; i++) // dari p dan anak
      anaknya
                cnt[ col[ euler[i] ] ]++;
    cnt[ col[now] ]++;
    //now cnt[c] is the number of vertices in subtree of vertex v
      that has color c. You can answer the queries easily.
    if(keep == 0)
        for(int i = st[now]; i < ft[now]; i++)
      cnt[ col[ euler[i] ] ]--;
}
```

2. Strongly Connected Component [Kosaraju]

```
stack<int> st;
void dfs(int curr) {
  if(vis[curr]) return;
  vis[curr] = 1;
  for(auto to: adj[curr]) {
    dfs(to);
  }
  st.push(curr);
}
void dfs2(int curr) {
  if(vis[curr]) return;
  vis[curr] = 1;
  for(auto to: radj[curr]) {
    dfs2(to);
  }
  // curr is part of new SCC
  cout << curr << " ";
}
int main() {
  ...
  while(!st.empty()) {
    curr = st.top();
    st.pop();
    if(!vis[curr]) {
      // new SCC
      dfs2(curr);
    }
}
```

3. Strongly Connected Component [Tarjan]

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
#define fi first
#define se second

const int maxn = 1e5 + 1;
vector <vector <int>> v(maxn);
vector <vector <int>> cc;
bitset <maxn> vis;
int tt = 1;
int low[maxn], dis[maxn], id[maxn], in[maxn], out[maxn];
stack <int> st;
bitset <maxn> contain = 0;

void dfs(int a){
  low[a] = dis[a] = tt++;
  vis[a] = 1;
  st.push(a);
  contain[a] = 1;
  for(auto p : v[a]){
    if(!vis[p]){
      dfs(p);
      low[a] = min(low[a], low[p]);
    }
    else if(contain[p]){
      low[a] = min(low[a], dis[p]);
    }
  }

  if(low[a] == dis[a]){
    id[a] = cc.size();
    cc.push_back({a});
    while(!st.empty() && st.top() != a){
      cc.back().push_back(st.top());
      id[st.top()] = id[a];
      contain[st.top()] = 0;
      st.pop();
    }
    if(!st.empty()) st.pop();
  }
}

void compress(int n){
  for(int a = 1; a <= n; ++a){
    for(auto p : v[a]){
      if(id[a] != id[p]) in[id[p]]++, out[id[a]]++;
    }
  }
}

int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;
  int i, j;
  for(int a = 1; a <= m; ++a){
    cin >> i >> j;
    v[i].push_back(j);
  }
  for(int a = 1; a <= n; ++a){
    if(!vis[a]) tt = 1, dfs(a);
  }

  compress(n);

  int ou = 0, ii = 0;
  for(int a = 0; a < cc.size(); ++a){
    if(in[a] == 0) ii++;
    if(out[a] == 0) ou++;
  }

  bool ans = (ou == 1 && ii == 1);
  cout << (ans?"YA":"TIDAK") << '\n';

  return 0;
}
```

4. Articulation and Bridge

```
void tarjanAPB(int u){
  dlow[u] = dnum[u] = nxt++;
  for ( int i = 0; i < adlis[u].size(); i++ ){
      int v = adlis[u][i];
      if ( dnum[v] == -1 ) {
        dpar[v] = u;
        if ( u == dfs_root ) child_root++;
        tarjanAPB(v);
        if ( dlow[v] >= dnum[u] ) {
          isAP[u] = true;
        }
        if ( dlow[v] > dnum[u] ) {
          is_bridge[u][v] = true;
        }
      dlow[u] = min(dlow[u], dlow[v]);
      }
      else if ( v != dpar[u] ) {
        dlow[u] = min(dlow[u], dnum[v]);
      }
  }
}...
nxt=0;
RESET(dnum,-1);
RESET(dlow,-1);
RESET(dpar,-1);
RESET(isAP,0);
RESET(is_bridge,0);
for ( int i=0; i < nvert; i++ ){
  if ( dnum[i] == -1 ){
      dfs_root = i;
      child_root = 0;
      tarjanAPB(i);
      is_AP[dfs_root] = (child_root > 1);
  }
}
```

5. Heavy Light Decomposition

```
/*
  This is currently one of my favourite algorithm so far
  Heavy Light Decomposition
*/

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
#define fi first
#define se second

const int maxn = 2e5 + 1;
vector <vector <int>> graf(maxn);
int sz[maxn];  // store the size of subtree with node i is the
```

```
    top most node on the subtree
int hv[maxn];  // store the heavy child of each node
int par[maxn]; // store the parent of each node
int st[maxn];  // store the index of each node in segment tree
int dep[maxn]; // store the depth of each node
int top[maxn]; // store the top-most node in a single range in
    segment tree from each node
int h[maxn];   // store the value of each node
vector <int> euler; // this array store the node order in the
    segment tree

void dfs(int a, int pr){
  // set up everything
  sz[a] = 1;
  par[a] = pr;
  hv[a] = -1;
  int mx = -1;

  // dfs
  for(auto child : graf[a]){
    if(child == pr) continue;
    dep[child] = dep[a] + 1;
    dfs(child, a);

    // getting heavy child
    if(sz[child] > mx){
      hv[a] = child;
      mx = sz[child];
    }
    sz[a] += sz[child];
  }
}

int tt = 0;
void decompose(int a, int pr, int tp){
  // decompose
  st[a] = tt++;
  top[a] = tp;
  euler.push_back(a);

  // decompose to heavy child
  if(hv[a] != -1) decompose(hv[a], a, tp);

  // decompose to another child
  for(auto child : graf[a]){
    if(child == pr || child == hv[a]) continue;
    decompose(child, a, child);
  }
}

// classic segment tree
int seg[4 * maxn];

int merge(int a, int b){
  return a + b;
}

void build(int l, int r, int v){
  if(l == r){
    seg[v] = h[euler[l]];
  }
  else{
    int m = (l + r)/2;
    build(l, m, 2 * v);
    build(m + 1, r, 2 * v + 1);
    seg[v] = merge(seg[2 * v], seg[2 * v + 1]);
  }
}

int query(int l, int r, int v, int lq, int rq){
  if(lq <= l && r <= rq) return seg[v];
  if(l > rq || r < lq) return 0;
  int m = (l + r)/2;
  return merge(query(l, m, 2 * v, lq, rq), query(m + 1, r, 2 * v
    + 1, lq, rq));
}

int lca(int a, int b){ // Complexity O(log^2(N)) beacuse of
    segment tree
  // with bin-lift
  int res = 0;
  while(top[a] != top[b]){
    if(dep[top[a]] > dep[top[b]]) swap(a, b);
    res = merge(res, query(0, euler.size() - 1, 1, st[top[b]], st
        [b]));
    b = par[top[b]];
  }
  if(st[a] > st[b]) swap(a, b);
  res = merge(res, query(0, euler.size() - 1, 1, st[a], st[b]));
  return res;
}

int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  dfs(1, -1);
  decompose(1, -1, 1);
  build(0, euler.size() - 1, 1);

  // if there is a query from certain node to another
  int i, j;
  cin >> i >> j;
  cout << lca(i, j) << '\n';

  return 0;
}
```

6. Bipartite Matching

```
#include <bits/stdc++.h>
using namespace std;

struct Random : std::mt19937_64 {
    using std::mt19937_64::mt19937_64;
    using std::mt19937_64::operator();
    static int64_t gen_seed() {
        return std::chrono::steady_clock::now().time_since_epoch
     ().count();
    }
    Random() : std::mt19937_64(gen_seed()) {}
    template <class Int>
    auto operator()(Int a, Int b)
        -> std::enable_if_t<std::is_integral_v<Int>, Int> {
        return std::uniform_int_distribution<Int>(a, b)(*this);
    }
    template <class Int>
    auto operator()(Int a) -> std::enable_if_t<std::is_integral_v
     <Int>, Int> {
        return std::uniform_int_distribution<Int>(0, a - 1)(*this
     );
    }
    template <class Real>
    auto operator()(Real a, Real b)
        -> std::enable_if_t<std::is_floating_point_v<Real>, Real>
      {
        return std::uniform_real_distribution<Real>(a, b)(*this);
    }
};

template <bool ToShuffle = false>
struct bipartite_matching {
    int n_left, n_right, flow = 0;
    vector<vector<int>> g;
    vector<int> match_from_left, match_from_right;

    bipartite_matching(int _n_left, int _n_right)
        : n_left(_n_left),
          n_right(_n_right),
          g(_n_left),
          match_from_left(_n_left, -1),
          match_from_right(_n_right, -1),
          dist(_n_left) {}

    void add(int u, int v) { g[u].push_back(v); }

    vector<int> dist;

    void bfs() {
        queue<int> q;
        for (int u = 0; u < n_left; ++u) {
            if (!~match_from_left[u])
                q.push(u), dist[u] = 0;
            else
                dist[u] = -1;
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
```

```
            for (auto v : g[u])
                if (~match_from_right[v] && !~dist[
    match_from_right[v]]) {
                    dist[match_from_right[v]] = dist[u] + 1;
                    q.push(match_from_right[v]);
                }
        }
    }

    bool dfs(int u) {
        for (auto v : g[u])
            if (!~match_from_right[v]) {
                match_from_left[u] = v, match_from_right[v] = u;
                return true;
            }
        for (auto v : g[u])
            if (dist[match_from_right[v]] == dist[u] + 1 &&
                dfs(match_from_right[v])) {
                match_from_left[u] = v, match_from_right[v] = u;
                return true;
            }
        return false;
    }

    int get_max_matching() {
        if constexpr (ToShuffle) {
            Random rng;
            for (int i = 0; i < n_left; ++i)
                random_shuffle(begin(g[i]), end(g[i]), rng);
        }
        while (true) {
            bfs();
            int augment = 0;
            for (int u = 0; u < n_left; ++u)
                if (!~match_from_left[u]) augment += dfs(u);
            if (!augment) break;
            flow += augment;
        }
        return flow;
    }
};

int main(){
  bipartite_matching<false> matching(M, N);

  // Masukin E edge
  // i harus di rentang 0 <= i < M
  // j harus di rentang 0 <= j < N
  for (auto p : E) matching.add(p.first, p.second);

  // dapetin jumlah matching
  cout << matching.get_max_matching() << '\n';

  return 0;
}
```

7. Bipartite Matching GFG (Hopcroft Karp)

```
/**
 * Author: Chen Xing
 * Date: 2009-10-13
 * License: CC0
 * Source: N/A
 * Description: Fast bipartite matching algorithm. Graph $g$
     should be a list
 * of neighbors of the left partition, and $btoa$ should be a
     vector full of
 * -1's of the same size as the right partition. Returns the size
      of
 * the matching. $btoa[i]$ will be the match for vertex $i$ on
     the right side,
 * or $-1$ if it's not matched.
 * Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
 * Time: O(\sqrt{V}E)
 * Status: stress-tested by MinimumVertexCover, and tested on
     oldkattis.adkbipmatch and SPOJ:MATCHING
 */
#pragma once

// g nya zero based
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    /// Find the starting nodes for BFS (i.e. layer 0).
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    /// Find all layers using bfs.
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

Data Structure

1. Policy Based Data Structures

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;

ordered_set x;
x.insert(p); // memasukkan elemen p ke ordered_set
x.find_by_order(p); // elemen terkecil ke p (mulai dari 0)
x.order_of_key(p); // banyaknya elemen yang < p
if(x.find(p) != x.end()) x.erase(x.find(p)) // hapus elemen p
```

2. Fenwick Tree

```
void update(int x, LL val) {
  for(int i = x; i <= N; i+=i&-i) {
    ft[i] += val;
  }
}

LL query(int x) {
  LL ret = 0;
  for(int i = x; i > 0; i-=i&-i) {
    ret += ft[i];
  }
  return ret;
}
```

3. Treap

```
struct node {
  node *l, *r;
  int prior, sz, key;
  node(int val): prior(rng()), sz(1), key(val), l(NULL), r(NULL)
     {}
};
typedef node* pnode;
int get_sz(pnode curr) {
  return curr ? curr->sz : 0;
}
void upd_sz(pnode curr) {
```

```
  if(curr) curr->sz = get_sz(curr->l) + 1 + get_sz(curr->r);
}
void split(pnode curr, pnode &l, pnode &r, int x) {
  if(!curr) l = r = NULL;
  else if(curr->key <= x) split(curr->r, curr->r, r, x), l = curr
     ;
  else split(curr->l, l, curr->l, x), r = curr;
  upd_sz(curr);
}
void merge(pnode &curr, pnode l, pnode r) {
  if(!l || !r) curr = l ? l : r;
  else if(l->prior > r->prior) merge(l->r, l->r, r), curr = l;
  else merge(r->l, l, r->l), curr = r;
  upd_sz(curr);
}
```

4. Treap (Implicit Key)

```
void split(pnode curr, pnode &l, pnode &r, int x, int add = 0) {
  if(!curr) return void(l = r = NULL);
  int curr_key = add + get_sz(curr->l);
  if(curr_key <= x) split(curr->r, curr->r, r, x, curr_key+1), l
     = curr;
  else split(curr->l, l, curr->l, x, add), r = curr;
  upd_sz(curr);
}
void merge(pnode &curr, pnode l, pnode r) {
  if(!l || !r) curr = l ? l : r;
  else if(l->prior > r->prior) merge(l->r, l->r, r), curr = l;
  else merge(r->l, l, r->l), curr = r;
  upd_sz(curr);
}
```

5. Li-Chao Tree

```
// min implementation
struct line {
  LL m, c;
  LL operator()(LL x) {
    return m*x + c;
  }
} lines[4*MX];
void add(line nw, int idx = 0, int sl = 0, int sr = MX) {
  int sm = sl + sr >> 1;
  bool lef = nw(sl) < lines[idx](sl);
  bool mid = nw(sm) < lines[idx](sm);
  if(mid) swap(lines[idx], nw);
  if(sr - sl == 1) return;
  else if(lef != mid) add(nw, 2*idx + 1, sl, sm);
  else add(nw, 2*idx + 2, sm, sr);
}
LL get(int x, int idx = 0, int sl = 0, int sr = MX) {
  int sm = sl + sr >> 1;
  if(sr - sl == 1) return lines[idx](sl);
  else if(x < sm) return min(lines[idx](x), get(x, 2*idx+1, sl,
     sm));
  else return min(lines[idx](x), get(x, 2*idx+2, sm, sr));
}
```

6. Treap (Full Implementation)

```
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
     count());

struct node{
  ll prior, size, lazy;
  // value, task_num
  pair <ll, ll> key, min;
  node *L, *R;
  node(int val, int task_num){
    prior = rng();
    size = 1;
    key = min = make_pair(val, task_num);
    lazy = 0;
    L = R = NULL;
  }
};

typedef node* pnode;

struct Treap{
  // 0 based
  pnode treap;

  // mendapatkan size
  int getsz(pnode T){
    return T ? T->size : 0;
  }

  // mendapatkan ukuran tatal
  int getsz(){
    return getsz(treap);
  }

  // mendapatkan minimum
  pair <ll, ll> getmin(pnode T){
    return T ? make_pair(T->min.fi + T->lazy, T->min.se) :
     make_pair(LLONG_MAX, 0ll);
  }

  // update T
  void upd_sz(pnode T){
    if(!T) return;
    T->size = getsz(T->L) + 1 + getsz(T->R);
  }

  // propagate
  void prop(pnode T){
    if(!T || T->lazy == 0) return;
    T->key.fi += T->lazy;
    T->min.fi += T->lazy;
    if(T->L) T->L->lazy += T->lazy;
    if(T->R) T->R->lazy += T->lazy;
    T->lazy = 0;
  }

  // operation (ngemerge)
  void operate(pnode T){
    if(!T) return;
    T->min = T->key;
    T->min = min(T->min, getmin(T->L));
    T->min = min(T->min, getmin(T->R));
  }

  // split treap (0...X, X+1...N)
  void split(pnode T, pnode &L, pnode &R, int x, int add = 0){
    prop(T);
    if(!T){
      L=R=NULL;
      return;
    }
    int cur_idx = add + getsz(T->L);
    if(cur_idx <= x) split(T->R, T->R, R, x, cur_idx + 1), L = T;
    else split(T->L, L, T->L, x, add), R = T;
    upd_sz(T), operate(T);
  }

  // merge treap
  void merge(pnode &T, pnode L, pnode R){
    prop(L), prop(R);
    if(!L || !R) T = L?L:R; // base case
    else if(L->prior > R->prior) merge(L->R, L->R, R), T = L;
    else merge(R->L, L, R->L), T = R;
    upd_sz(T), operate(T);
  }

  // insert val, with task_num, in position pos (0 based)
  void insert(ll val, int task_num, int pos){
    pnode tmp, L, R, X = new node(val, task_num);;
    split(treap, L, R, pos - 1);
    merge(tmp, L, X);
    merge(treap, tmp, R);
  }

  // erase position pos (0 based)
  void erase(int pos){
    pnode L1, R1, L2, R2;
    split(treap, L1, R1, pos);
    split(L1, L2, R2, pos - 1);
    merge(treap, L2, R1);
  }

  // increase all value
  void add(ll val){
    if(val == 0) return;
    if(!treap) return;
    treap->lazy += val;
    prop(treap);
  }
```

```
  // cut some prefix, and increase (cut 0...pos, pos+1....)
  void cutAddPush(ll val, int pos){
    if(!treap) return;
    pnode L, R;
    split(treap, L, R, pos);
    if(L && val){
      L->lazy += val;
      prop(L);
    }
    merge(treap, R, L);
  }

  // find element with minimum value and index
  pair <ll, ll> dfs(pnode T, int add = 0){
    prop(T);
    int cur_idx = add + getsz(T->L);
    ll minT = T->key.fi;
    ll minL = getmin(T->L).fi;
    ll minR = getmin(T->R).fi;
    if(T->L && minL <= minR && minL <= minT) return dfs(T->L, add
     );
    if(T->R && minR < minL && minR < minT) return dfs(T->R,
     cur_idx + 1);
    return make_pair(minT, cur_idx);
  }

  // find element with minimum value and index
  pair <ll, ll> getMinTask(){
    if(treap)
      return dfs(treap);
    else
      return make_pair(LLONG_MAX, LLONG_MAX);
  }

  // find element with specified index
  pair <ll, ll> dfs2(pnode T, int pos, int add = 0){
    prop(T);
    int cur_idx = add + getsz(T->L);
    if(cur_idx > pos) return dfs2(T->L, pos, add);
    if(cur_idx < pos) return dfs2(T->R, pos, cur_idx + 1);
    return T->key;
  }

  // get task at position pos
  pair <ll, ll> getTask(int pos){
    if(treap)
      return dfs2(treap, pos);
    else
      return make_pair(LLONG_MAX, LLONG_MAX);
  }

  // print
  void print(pnode T){
    if(!T) return;
    prop(T);
    if(getsz(T) == 1){
      cout << "{" << T->key.fi << " " << T->key.se << "}" << " ";
    }
    else{
      if(T->L) print(T->L);
      cout << "{" << T->key.fi << " " << T->key.se << "}" << " ";
      if(T->R) print(T->R);
    }
  }
};
```

7. gp hash table (faster map)

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

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
     count());

struct chash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock
            ::now().time_since_epoch().count();
           return splitmix64(x + FIXED_RANDOM);
        }
    };

    gp_hash_table <ll, ll, chash> gp;
```

Dynamic Segment Tree

```
struct Vertex {
    int left, right;
    int sum = 0;
    Vertex *left_child = nullptr, *right_child = nullptr;

    Vertex(int lb, int rb) {
        left = lb;
        right = rb;
    }

    void extend() {
        if (!left_child && left + 1 < right) {
            int t = (left + right) / 2;
            left_child = new Vertex(left, t);
            right_child = new Vertex(t, right);
        }
    }

    void add(int k, int x) {
        extend();
        sum += x;
        if (left_child) {
            if (k < left_child->right)
                left_child->add(k, x);
            else
                right_child->add(k, x);
        }
    }

    int get_sum(int lq, int rq) {
        if (lq <= left && right <= rq)
            return sum;
        if (max(left, lq) >= min(right, rq))
            return 0;
        extend();
        return left_child->get_sum(lq, rq) + right_child->get_sum(lq
     , rq);
    }
};
```

Persistent Segment Tree

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;

struct Node {
  ll val;
  Node *l, *r;

  Node(ll x) : val(x), l(nullptr), r(nullptr) {}
  Node(Node *ll, Node *rr) {
    l = ll, r = rr;
    val = 0;
    if (l) val += l->val;
    if (r) val += r->val;
  }
  Node(Node *cp) : val(cp->val), l(cp->l), r(cp->r) {}
};

int n, cnt = 1;
ll a[200001];
Node *roots[200001];

Node *build(int l = 1, int r = n) {
  if (l == r) return new Node(a[l]);
  int mid = (l + r) / 2;
  return new Node(build(l, mid), build(mid + 1, r));
}

Node *update(Node *node, int val, int pos, int l = 1, int r = n) {
  if (l == r) return new Node(val);
  int mid = (l + r) / 2;
  if (pos > mid) return new Node(node->l, update(node->r, val, pos,
     mid + 1, r));
  else return new Node(update(node->l, val, pos, l, mid), node->r);
}
```

```
ll query(Node *node, int a, int b, int l = 1, int r = n) {
  if (l > b || r < a) return 0;
  if (l >= a && r <= b) return node->val;
  int mid = (l + r) / 2;
  return query(node->l, a, b, l, mid) + query(node->r, a, b, mid +
     1, r);
}

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int q;
  cin >> n >> q;
  for (int i = 1; i <= n; i++) cin >> a[i];
  roots[cnt++] = build();

  while (q--) {
    int t;
    cin >> t;
    if (t == 1) {
      int k, i, x;
      cin >> k >> i >> x;
      roots[k] = update(roots[k], x, i);
    } else if (t == 2) {
      int k, l, r;
      cin >> k >> l >> r;
      cout << query(roots[k], l, r) << '\n';
    } else {
      int k;
      cin >> k;
      roots[cnt++] = new Node(roots[k]);
    }
  }
  return 0;
}
```

Dynamic Li Chao Tree

```
typedef long long ll;
#define fi first
#define se second

// minimum lichao tree
// data type, lower bound, upper bound, default value
template <typename T, ll x_low, ll x_high, ll id>
struct DynamicLiChaoTree {
    struct Line {
        T a, b;
        Line(T a, T b) : a(a), b(b) {}
        inline T get(T x) const { return a * x + b; }
    };

    struct Node {
        Line x;
        Node *l, *r;
        Node(const Line& x) : x{x}, l{nullptr}, r{nullptr} {}
        ~Node() {
            if (l) delete l;
            if (r) delete r;
        }
    };

    Node* root;
    DynamicLiChaoTree() : root{nullptr} {}

    void clear() {
        if (root) delete root;
        root = nullptr;
    }

    Node* add_line(Node* t, Line& x, const T& l, const T& r, const T
     & x_l,
                   const T& x_r) {
        if (!t) return new Node(x);

        T t_l = t->x.get(l), t_r = t->x.get(r);

        // MAX CHANGE: if (t_l >= x_l && t_r >= x_r)
        if (t_l <= x_l && t_r <= x_r) {
            return t;
        }
        // MAX CHANGE: else if (t_l <= x_l && t_r <= x_r)
        else if (t_l >= x_l && t_r >= x_r) {
            t->x = x;
            return t;
        } else {
            T m = (l + r) / 2;
            if (m == r) --m;
            T t_m = t->x.get(m), x_m = x.get(m);

            // MAX CHANGE: if (t_m < x_m)
            if (t_m > x_m) {
                swap(t->x, x);
                // MAX CHANGE: if (x_l <= t_l)
                if (x_l >= t_l)
                    t->l = add_line(t->l, x, l, m, t_l, t_m);
                else
                    t->r = add_line(t->r, x, m + 1, r, t_m + x.a,
     t_r);
            } else {
                // MAX CHANGE: if (t_l <= x_l)
                if (t_l >= x_l)
                    t->l = add_line(t->l, x, l, m, x_l, x_m);
                else
                    t->r = add_line(t->r, x, m + 1, r, x_m + x.a,
     x_r);
            }
            return t;
        }
    }

    void add_line(const T& a, const T& b) {
        Line x(a, b);
        root = add_line(root, x, x_low, x_high, x.get(x_low), x.get(
     x_high));
    }

    Node* add_segment(Node* t, Line& x, const T& a, const T& b,
     const T& l,
                      const T& r, const T& x_l, const T& x_r) {
        if (r < a || b < l) return t;
        if (a <= l && r <= b) {
            Line y{x};
            return add_line(t, y, l, r, x_l, x_r);
        }
        if (t) {
            T t_l = t->x.get(l), t_r = t->x.get(r);
            // MAX CHANGE: if (t_l >= x_l && t_r >= x_r) return t;
            if (t_l <= x_l && t_r <= x_r) return t;
        } else {
            t = new Node(Line(0, id));
        }
        T m = (l + r) / 2;
        if (m == r) --m;
        T x_m = x.get(m);
        t->l = add_segment(t->l, x, a, b, l, m, x_l, x_m);
        t->r = add_segment(t->r, x, a, b, m + 1, r, x_m + x.a, x_r);
        return t;
    }

    void add_segment(const T& l, const T& r, const T& a, const T& b)
      {
        Line x(a, b);
        root = add_segment(root, x, l, r - 1, x_low, x_high, x.get(
     x_low),
                           x.get(x_high));
    }

    T query(const Node* t, const T& l, const T& r, const T& x) const
      {
        if (!t) return id;
        if (l == r) return t->x.get(x);
        T m = (l + r) / 2;
        if (m == r) --m;
        if (x <= m)
            // MAX CHANGE: return max(t->x.get(x), query(t->l, l, m,
      x));
            return min(t->x.get(x), query(t->l, l, m, x));
        else
            // MAX CHANGE: return max(t->x.get(x), query(t->r, m +
     1, r, x));
            return min(t->x.get(x), query(t->r, m + 1, r, x));
    }

    T query(const T& x) const { return query(root, x_low, x_high, x)
     ; }
};
const int maxn = 1e9 + 1;
```

```
DynamicLiChaoTree <ll, 0, maxn-1, LLONG_MAX> t;

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n;
    cin >> n;
    int A[n+1], B[n+1];
    for(int i = 1; i <= n; ++i) cin >> A[i];
    for(int i = 1; i <= n; ++i) cin >> B[i];

    t.clear(); // freeing memory
    for(int i = 1; i <= n; ++i){
        if(i == n){
            ll ans = t.query(A[i]);
            cout << ans << '\n';
        }
        else if(i == 1){
            ll qu = 0;
            // cout << qu << '\n';
            t.add_line(B[i], qu);
        }
        else{
            ll qu = t.query(A[i]);
            t.add_line(B[i], qu);
            // cout << qu << '\n';
        }
    }

    return 0;
}
```

Geometry

1. Geometri Template

```
struct titik {
  double x, y;
};
double cross(titik a, titik b, titik c) {
  // | i  j  k |
  // | Ax Ay 0 |
  // | Cx Cy 0 |
  // = Ax . Cy - Ay . Cx
  // >0 = cw
  return (a.x - b.x)*(c.y - b.y) - (a.y - b.y)*(c.x - b.x);
}
bool in_segment(titik a, titik O, titik b) {
  // Apakah O ada pada segmen ab
  return cross(a, b, O) == 0
    && min(a.x, b.x) <= O.x && O.x <= max(a.x, b.x)
    && min(a.y, b.y) <= O.y && O.y <= max(a.y, b.y);
}
bool berpotongan(titik a, titik b, titik c, titik d) {
  // Apakah segmen ab dan cd berpotongan
  double abc = cross(a, b, c);
  double abd = cross(a, b, d);
  double cda = cross(c, d, a);
  double cdb = cross(c, d, b);
  if(abc * abd < 0 && cda * cdb < 0) return 1;
  if(in_segment(a, c, b)) return 1;
  if(in_segment(a, d, b)) return 1;
  if(in_segment(c, a, d)) return 1;
  if(in_segment(c, b, d)) return 1;
  return 0;
}
```

2. Convex Hull

```
typedef long long ll;
typedef double ld;
#define fi first
#define se second

struct pt {
    ld x, y;
};

inline ld cross(const pt& a, const pt& b, const pt& c) {
    return (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x);
}

int orientation(const pt& a, const pt& b, const pt& c) {
    ld v = cross(a, b, c);
    if (v < 0) return -1;
    if (v > 0) return 1;
    return 0;
}

void convex_hull(vector<pt> &a, bool include_collinear = false) {
    if (a.size() <= 1) return;

    sort(a.begin(), a.end(), [](const pt& A, const pt& B) {
        if (A.x != B.x) return A.x < B.x;
        return A.y < B.y;
    });

    vector<pt> lower, upper;

    for (const pt& p : a) {
        while (lower.size() >= 2) {
            int o = orientation(lower[lower.size()-2], lower.back
 (), p);
            if (o > 0 || (include_collinear && o == 0)) break;
            lower.pop_back();
        }
        lower.push_back(p);
    }

    for (int i = (int)a.size() - 1; i >= 0; --i) {
        const pt& p = a[i];
        while (upper.size() >= 2) {
            int o = orientation(upper[upper.size()-2], upper.back
 (), p);
            if (o > 0 || (include_collinear && o == 0)) break;
            upper.pop_back();
        }
        upper.push_back(p);
    }

    lower.pop_back();
    upper.pop_back();

    a = lower;
    a.insert(a.end(), upper.begin(), upper.end());
}
```

Miscellaneous

1. Fast Input

```
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma,popcnt,bmi2,bmi,lzcnt")
#pragma unroll 2
#include <immintrin.h>
uint32_t odd_bits(uint32_t x){return _pext_u32(x, 0x55555555u);}

#define gc getchar

namespace fastio{
  template <typename T> inline void sca(T &angka){
    T kali = 1; angka = 0; char input = gc();
    while (input < '0' || input > '9'){if (input == '-') kali =
     -1; input = gc();}
    while(input >= '0' && input <= '9')angka = (angka << 3) + (
     angka << 1) + input - 48, input = gc();
    angka *= kali;
  }
  template <typename T> inline void scans(T &s){
    s = ""; char input = gc(); while(input != ' ' && input != '\n
     ') s += input, input = gc();
  }
  template <typename T> inline void scanln(T &s){
    s = ""; char input = gc(); while(input != '\n') s += input,
     input = gc();
  }
  template <typename FIRST, typename... REST > inline void scan(
     FIRST& first, REST&... rest ); // utama
  inline void scan() {}
    template <typename FIRST, typename... REST > inline void scan
     ( FIRST& first, REST&... rest ){sca(first);scan(rest...);}
}using namespace fastio;
```

DSU Rollback (Offline)

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
#define fi first
```

```
#define se second

struct info{
    int par, sz;
    info(int pr = -1, int _sz = -1){
        par = pr;
        sz = _sz;
    }
};

struct hist{
    info a, b;
};

const int maxn = 1e5 + 2;
int last[maxn];
vector <pair <int, int>> e = {{-1, -1}};
stack <hist> history;
inline int snapshot() {
    return history.size();
}
int N, M, Q;
vector <pair <int, int>> seg[4 * maxn];

ll cnt = 0;
ll ans[maxn];
info d[maxn];

inline int find(int a){
    while(d[a].par != a) a = d[a].par;
    return a;
}

inline ll calc(ll x){
    return (x * (x - 1)) / 2;
}

inline void join(int a, int b){ // join edge ke i
    int pa = find(a);
    int pb = find(b);
    if(pa == pb) return;

    // temporary
    hist tmp;
    tmp.a = d[pa];
    tmp.b = d[pb];
    history.push(tmp);

    // gabungkan pa ke pb
    if(d[pa].sz > d[pb].sz) swap(pa, pb);
    cnt -= (calc(d[pa].sz) + calc(d[pb].sz));
    d[pb].sz += d[pa].sz;
    cnt += calc(d[pb].sz);
    d[pa].par = pb;
}

inline void rollback(int snap){
    while(snapshot() > snap){
        hist tmp = history.top();
        history.pop();
        d[tmp.a.par] = tmp.a;
        d[tmp.b.par] = tmp.b;
        cnt += (calc(tmp.a.sz) + calc(tmp.b.sz) - calc(tmp.a.sz +
     tmp.b.sz));
    }
}

// tambahkan semua nilai val
void update(int l, int r, int v, int lq, int rq, int i, int j){
    if(l > rq || r < lq) return;
    if(lq <= l && r <= rq){
        seg[v].emplace_back(i, j);
        return;
    }
    int m = (l + r)/2;
    update(l, m, 2 * v, lq, rq, i, j);
    update(m + 1, r, 2 * v + 1, lq, rq, i, j);
}

void run(int l, int r, int v){
    // jalankan
    int snap = snapshot();
    if(seg[v].size())
        for(int i = 0; i < seg[v].size(); ++i)
            join(seg[v][i].fi, seg[v][i].se);

    // transverse
    if(l == r){
        ans[l] = cnt;
    }
    else{
        int m = (l + r)/2;
        run(l, m, 2 * v);
        run(m + 1, r, 2 * v + 1);
    }

    // rollback
    rollback(snap);
}

int main(){
    ios_base::sync_with_stdio(0);
    // cin.tie(0);

    cin >> N >> M >> Q;
    for(int i = 1; i <= N; ++i){
        d[i] = info(i, 1);
    }
    for(int a = 1; a <= M; ++a){
        int x, y, z;
        cin >> x >> y >> z;
        e.emplace_back(x, y);
        if(z == 1) last[a] = 1;
        else last[a] = -1;
    }

    for(int a = 1; a <= Q; ++a){
        string t; int i;
        cin >> t >> i;
        if(t == "G"){ // tutup
            // update langsung (dari last[i] - (a-1))
            if(last[i] != a) update(1, Q+1, 1, last[i], a - 1, e[i].
     fi, e[i].se);
            last[i] = -1;
        }
        else{ // buka
            last[i] = a;
        }
    }

    for(int i = 1; i <= M; ++i){
        if(last[i] != -1){
            if(last[i] != Q+1) update(1, Q+1, 1, last[i], Q+1, e[i].
     fi, e[i].se);
            last[i] = -1;
        }
    }

    run(1, Q+1, 1);
    for(int i = 1; i <= Q; ++i){
        cout << ans[i] << " \n"[i != Q+1];
    }

    return 0;
}
```

DSU Rollback (More Convenient Join)

```
class DSU {
  private:
  vector<int> p, sz;
  // stores previous unites
  vector<pair<int &, int>> history;

  public:
  DSU(int n) : p(n), sz(n, 1) { iota(p.begin(), p.end(), 0); }

  int get(int x) { return x == p[x] ? x : get(p[x]); }

  void unite(int a, int b) {
    a = get(a);
    b = get(b);
    if (a == b) { return; }
    if (sz[a] < sz[b]) { swap(a, b); }

    // save this unite operation
    history.push_back({sz[a], sz[a]});
    history.push_back({p[b], p[b]});
```

```
    p[b] = a;
    sz[a] += sz[b];
  }

  int snapshot() { return history.size(); }

  void rollback(int until) {
    while (snapshot() > until) {
      history.back().first = history.back().second;
      history.pop_back();
    }
  }
};
```

DSU Rollback (Online) [Persistent DSU]

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
#define fi first
#define se second

const int maxn = 2e5 + 3;
vector <pair <int, int>> par[maxn];
vector <pair <int, int>> sz[maxn];
int n, m, q;

void update(int i, int t, int val){ // update elemen ke i, di waktu
     t, dengan nilai val
    assert(par[i].back().fi < t);
    par[i].emplace_back(t, val);
}

int get(int i, int t){ // dapatkan elemen ke i, di waktu minimal t
    int id = upper_bound(par[i].begin(), par[i].end(), make_pair(t,
     INT_MAX)) - par[i].begin();
    id--;
    return par[i][id].se;
}

void update_size(int i, int t, int val){
    assert(sz[i].back().fi < t);
    sz[i].emplace_back(t, val);
}

int get_sz(int i, int t){
    int id = upper_bound(sz[i].begin(), sz[i].end(), make_pair(t,
     INT_MAX)) - sz[i].begin();
    id--;
    return sz[i][id].se;
}

int find(int i, int t){
    int pr = get(i, t);
    return (pr == i) ? i : find(pr, t);
}

bool same(int i, int j, int t){
    return find(i, t) == find(j, t);
}

void join(int a, int b, int t){
    int pa = find(a, t);
    int pb = find(b, t);
    if(pa == pb) return;
    int sza = get_sz(pa, t);
    int szb = get_sz(pb, t);
    if(sza > szb){
        swap(pa, pb);
        swap(sza, szb);
    }
    update(pa, t, pb);
    update_size(pb, t, szb + sza);
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> m >> q;
    for(int i = 1; i <= n; ++i){
        par[i].emplace_back(0, i);
        sz[i].emplace_back(0, 1);
    }
    for(int i = 1; i <= m; ++i){
        int a, b;
        cin >> a >> b;
        join(a, b, i);
    }
    for(int i = 1; i <= q; ++i){
        int a, b;
        cin >> a >> b;
        int ans = INT_MAX;
        int l = 0, r = m;
        while(l <= r){
            int mid = (l + r)/2;
            if(same(a, b, mid)){
                ans = min(ans, mid);
                r = mid - 1;
            }
            else{
                l = mid + 1;
            }
        }
        cout << (ans == INT_MAX ? -1 : ans) << '\n';
    }

    return 0;
}
```

Hashing

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
#define fi first
#define se second

const int maxn = 1e5 + 1;
const ll mod = 1e9 + 7; // 1e9 + 9
ll kar = 139; // 131, 137, 149, 151
ll pref[maxn];
ll pg[maxn];

ll pang(ll a, ll b){
  if(b == 0) return 1;
  return pang(a * a % mod, b / 2) * (b & 1 ? a : 1ll) % mod;
}

inline ll get(ll l, ll r){
  ll res = (pref[r] - ((pref[l - 1] * pang(kar, (r - l + 1))) % mod)
     ) % mod;
  if(res < 0) res += mod;
  res %= mod;
  return res;
}

int main(){
  string s;
  cin >> s;
  s = "~" + s;
  int n = s.length();

  pg[0] = 1;
  for(int a = 1; a <= n; ++a){ // pangkat
    pg[a] = pg[a - 1] * kar % mod;
  }

  for(int a = 1; a <= n; ++a){
    pref[a] = (((pref[a - 1] * kar) % mod) + (ll)(s[a] - 'a' + 1)) %
      mod;
    if(pref[a] < 0) pref[a] += mod;
    pref[a] %= mod;
  }

  cout << get(2, 2) << '\n';
  cout << get(1, 2) << '\n';
  cout << get(1, 1) << '\n';

  return 0;
}
```

Z Algorithm

```
/* Generate Z Array
Z[i] is maximum R such that S[0..R-1] == S[i..i+R-1]
Usage:
1. Everything that can be done with KMP
2. Longest Common Prefix in linear time
```

```
*/

vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

Manacher

```
// to find the number of palindrome substring in O(N)
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 0, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = min(r - i, p[l + (r - i)]);
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}
```

Suffix Array

```
// Suffix Array (SA) & Bantuan
int SA[MAXN];   // Suffix Array: SA[i] = indeks awal dari sufiks
     peringkat ke-i.
int tmpS[MAXN]; // Array SA sementara, digunakan selama proses Radix
      Sort.
// Rank Array (RA) & Bantuan
int RA[MAXN];   // Rank Array: RA[i] = peringkat dari sufiks yang
     dimulai di indeks i.
int tmpR[MAXN]; // Array RA sementara, digunakan untuk menghitung
     rank baru di tiap iterasi.
// Radix Sort
int cnt[MAXN];  // Array 'count' untuk Counting Sort (bagian dari
     Radix Sort).
// LCP Array & Bantuan (Kasai's Algorithm)
int phi[MAXN];  // phi[i] = indeks sufiks yang berada sebelum sufiks
      'i' di SA.
int PLCP[MAXN]; // Permuted LCP: PLCP[i] = LCP antara sufiks 'i' dan
      sufiks 'phi[i]'.
int LCP[MAXN];  // LCP Array: LCP[i] = LCP antara sufiks SA[i] dan
     SA[i-1].

int len;
void radixSort(int k){
    int maxx = max(len,300);
    for(int i = 0 ; i < maxx ; i++)
        cnt[i] = 0;
    for(int i = 0 ; i  < len ; i++)
        cnt[i + k < len ? RA[i + k] : 0]++;
    int tot = 0;
    for(int i = 0 ; i < maxx ; i++){
        int tmp = cnt[i];
        cnt[i] = tot;
        tot += tmp;
    }
    for(int i = 0 ; i < len ; i++)
        tmpS[cnt[SA[i] + k < len ? RA[SA[i] + k] : 0]++] = SA[i];
    for(int i = 0 ; i < len ; i++)
        SA[i] = tmpS[i];
}
void buildSA(){
  len = s.length();
  s[len++] = '$';
    for(int i = 0 ; i < len ; i++){
        SA[i] = i;
        RA[i] = s[i];
    }
    for(int lg = 1 ; lg < len ; lg <<= 1){
        Radix_Sort(lg);
        Radix_Sort(0);
        int tot = 0;
        tmpR[SA[0]] = 0;
        for(int i = 1 ; i < len ; i++)
            tmpR[SA[i]] = (RA[SA[i]] == RA[SA[i - 1]] && SA[i] + lg
     < len && SA[i - 1] + lg < len
                                && RA[SA[i] + lg] == RA[SA[i - 1] + lg])
      ? tot : ++tot;

        for(int i = 0 ; i < len ; i++)
            RA[i] = tmpR[i];
        if(RA[SA[len - 1]] == len - 1) break;
    }
}

void buildLCP(){ // Longesst Common Prefix
    phi[SA[0]] = -1;
    for(int i = 1 ; i < len ; i++)
        phi[SA[i]] = SA[i - 1];
    for(int i = 0, l = 0 ; i < len ; i++){
        if(phi[i] == -1)
            PLCP[i] = 0;
        else{
            while(s[i + l] == s[phi[i] + l]) l++;
            PLCP[i] = l;
            l = max(0,l - 1);
        }
    }
    for(int i = 0 ; i < len ; i++)
        LCP[i] = PLCP[SA[i]];
}
```

KMP

```
const int MAXN = 1000005;
#define MAX_N 100010
char T[MAX_N], P[MAX_N]; // T = text, P = pattern
int b[MAX_N], n, m; // b = back table, n = length of T, m = length
     of P
void kmpPreprocess() { // call this before calling kmpSearch()
  int i = 0, j = -1; b[0] = -1; // starting values
  while (i < m) { // pre-process the pattern string P
    while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset
     j using b
    i++; j++; // if same, advance both pointers
    b[i] = j;
} }

void kmpSearch() { // this is similar as kmpPreprocess(), but on
     string T
  int i = 0, j = 0; // starting values
  while (i < n) { // search through string T
    while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset
     j using b
    i++; j++; // if same, advance both pointers
    if (j == m) { // a match found when j == m
      printf("P is found at index %d in T\n", i - j);
      j = b[j]; // prepare j for the next possible match
} } }
```

Aho Corasick

```
const int N = 1e5 + 5;

struct AhoCorasick {
  int trie[N][26];
  int fail[N];
  int saiz;

  AhoCorasick() {
    memset(trie[0],-1,sizeof trie[0]);
    saiz = 0;
```

```
  }

  void add(string str) {
    int cur = 0;
    for(int i = 0 ; i < str.length() ; i++) {
      //checkChar(str[i]);
      int nex = str[i] - 'a';
      if(trie[cur][nex] == -1) {
        trie[cur][nex] = ++saiz;
        memset(trie[saiz],-1,sizeof trie[saiz]);
      }
      cur = trie[cur][nex];
    }
  }

  void build() {
    queue<int> q;
    fail[0] = 0;

    for(int i = 0 ; i < 26 ; i++)
      if(trie[0][i] == -1)
        trie[0][i] = 0;
      else {
        int nex = trie[0][i];
        fail[nex] = 0;
        q.push(nex);
      }

    while(!q.empty()) {
      int now = q.front();
      q.pop();

      for(int i = 0 ; i < 26 ; i++)
        if(trie[now][i] == -1)
          trie[now][i] = trie[fail[now]][i];
        else {
          int nex = trie[now][i];
          fail[nex] = trie[fail[now]][i];
          q.push(nex);
        }
    }
  }

  int getIndex(string str) {
    int cur = 0;
    for(int i = 0 ; i < str.length() ; i++) {
      //checkChar(str[i]);
      cur = trie[cur][str[i] - 'a'];
    }
    return cur;
  }

};
```

Maxflow Edmons-Karp

```
/*
    Notes:
    capacity is a matrix NxN
    adj is an adjacency list
*/

int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

Maxflow Dinic

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap == edges[id].flow)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u])
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[
 id].flow));
            if (tr == 0)
```

```
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

Mod

Modmul and Modpow for big M

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

DP

1. Convex Hull Trick

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+
     m, and query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

2. Sum Over Subset (Naive) $O(4^N)$

```
vector<int> sos(1 << n);

for (int x = 0; x < (1 << n); x++) {
  // iterate over all other sets and checks whether they're a
     subset of x
  for (int i = 0; i < (1 << n); i++) {
    if ((i & x) == i) { sos[x] += a[i]; }
  }
}
```

3. Sum Over Subset $O(3^N)$

```
vector<int> sos(1 << n);

for (int x = 0; x < (1 << n); x++) {
  sos[x] = a[0];
  // iterate over all subsets of x directly
  for (int i = x; i > 0; i = (i - 1) & x) { sos[x] += a[i]; }
}
```

4. Sum Over Subset With DP $O(N \times 2^N)$

```
vector<int> sos(1 << n);
vector<vector<int>> dp(1 << n, vector<int>(n + 1));

for (int x = 0; x < (1 << n); x++) {
  dp[x][0] = a[x];
  for (int i = 0; i < n; i++) {
    dp[x][i + 1] = dp[x][i];
    if (x & (1 << i)) { dp[x][i + 1] += dp[x ^ (1 << i)][i]; }
  }
  sos[x] = dp[x][n];
}
```

5. More Memory Optimization

```
// Since $dp[x][i]$ only depends on $dp[x][i - 1]$, we can reuse
     the DP array.

sos = a;

for (int i = 0; i < n; i++) {
  for (int x = 0; x < (1 << n); x++) {
    if (x & (1 << i)) { sos[x] += sos[x ^ (1 << i)]; }
  }
}
```

Abstract Algebra

(a) Todd-Coxeter Algorithm

```
            struct DSU {
        vector<int> parent;

        DSU(int n) {
            parent.resize(n);
            iota(parent.begin(), parent.end(), 0);
        }

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

        bool join(int a, int b) {
            a = find(a);
            b = find(b);
            if (a != b) {
                parent[a] = b;
                return true;
            }
            return false;
        }
    };

    struct Coset {
        int G; //Generator termasuk invers
        vector<vector<int>> rels; // relasi
        vector<vector<int>> subgens; // generator subgrup

        vector<vector<int>> nxt; // coset table
        DSU dsu;
```

```
Coset(int generators) : G(generators), dsu(1) {
    nxt.push_back(vector<int>(G, -1)); // coset 0
}

int newcoset() {
    int id = nxt.size();
    nxt.push_back(vector<int>(G, -1));
    dsu.parent.push_back(id);
    return id;
}

void edge(int u, int g, int v) {
    nxt[u][g] = v;
    nxt[v][g ^ 1] = u;  // inverse generator
}

void merge(int a, int b) {
    a = dsu.find(a);
    b = dsu.find(b);
    if (a == b) return;

    dsu.parent[b] = a;

    for (int g = 0; g < G; g++) {
        int x = nxt[a][g];
        int y = nxt[b][g];

        if (x != -1 && y != -1) {
            merge(x, y);
        } else if (x == -1 && y != -1) {
            nxt[a][g] = y;
            nxt[y][g ^ 1] = a;
        }
    }
}


void scan(int start, const vector<int>& word) {
    int cur = start;
    for (int g : word) {
        cur = dsu.find(cur);
        if (nxt[cur][g] == -1) {
            int nw = newcoset();
            edge(cur, g, nw);
        }
        cur = dsu.find(nxt[cur][g]);
    }

    merge(start, cur);
}

void enumerate() {

    for (auto &w : subgens)
        scan(0, w);

    bool changed = true;
    while (changed) {
        changed = false;

        int sz = nxt.size();
        for (int c = 0; c < sz; c++) {
            int cc = dsu.find(c);

            for (auto &r : rels) {
                int cur = cc;

                for (int g : r) {
                    cur = dsu.find(cur);
                    if (nxt[cur][g] == -1) {
                        int nw = newcoset();
                        edge(cur, g, nw);
                        changed = true;
                    }
                    cur = nxt[cur][g];
                }

                if (dsu.find(cur) != dsu.find(c)) {
                    merge(cur, c);
                    changed = true;
                }
            }
        }
    }

    int index() {
        set<int> reps;
        for (int i = 0; i < (int)nxt.size(); i++)
            reps.insert(dsu.find(i));
        return reps.size();
    }

    int evaluate(const vector<int>& word) {
        int cur = 0;
        for (int g : word) {
            cur = dsu.find(cur);
            if (nxt[cur][g] == -1) return -1;
            cur = nxt[cur][g];
        }
        return dsu.find(cur);
    }
};

    // Generators:
    // 0: A
    // 1: A^-1 / A
    // 2: B
    // 3: B^-1 / B^2

    Coset grup(4);

    // AA = e
    grup.rels.push_back({0, 0});
    // BBB = e
    grup.rels.push_back({2, 2, 2});
    // s = e
    grup.rels.push_back(convert(s));

    grup.enumerate();
    a
```