Contents

1 Setting

1.1 Default code

```cpp
#include<bits/stdc++.h>
#include<format>
#pragma warning(disable:4996)
#pragma comment(linker, "/STACK:336777216")
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx,avx2,fma")
using namespace std;
using ll = long long;
using pll = pair<ll,ll>;
using tlll = tuple<ll,ll,ll>;
using ld = long double;
using pld = pair<ld,ld>;
typedef unsigned long long ull;
typedef __int128 LL;
typedef pair<ll, ll> pii;
typedef vector<ll> vi;
typedef pair<ll, ll> pll;
typedef vector<ll> vl;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;
template <typename T> using ordered_set = tree<T, null_type, less<>, rb_tree_tag,
  tree_order_statistics_node_update>;
template <typename T> using ordered_multiset = tree<T, null_type, less_equal<>, rb_tree_tag,
  tree_order_statistics_node_update>;
#define pb(x) push_back(x)
#define all(x) (x).begin(), (x).end()
#define rep(i,a,b) for (auto i = (a); i < (b); i++)
#define each(x, a) for (auto& x: a)

#define debug if constexpr (!ndebug) cout << "[DEBUG] "
#define debugv(x) if constexpr (!ndebug) cout << "[DEBUG] " << #x << " == " << x << '\n';
#define debugc(c) if constexpr (!ndebug) { cout << "[DEBUG] "<< #c << ": "; for (const auto& elem
  : c) cout << elem << ", "; cout << '\n'; }

#ifdef ONLINE_JUDGE
constexpr bool ndebug = true;
#else
constexpr bool ndebug = false;
#endif

ll gcd(ll a, ll b){return b?gcd(b,a%b):a;}
ll lcm(ll a, ll b){if(a&&b)return a*(b/gcd(a,b)); return a+b;}
ll POW(ll a, ll b, ll rem){ll p=1;a%=rem;for(;b;b>>=1,a=(a*a)% rem)if(b&1)p=(p*a)%rem;return p;}

void setup() {
    if(!ndebug) {
        freopen("input.txt", "r", stdin);
        freopen("output.txt", "w", stdout);
    }
    else {
        ios_base::sync_with_stdio(0);
        cin.tie(0);
        cout.tie(0);
    }
}

void preprocess() {

}

void solve(ll testcase){

}

int main() {
    setup();
    preprocess();
    ll t = 1;
    // cin >> t; cin.ignore();
    for (ll testcase = 1; testcase <= t; testcase++){
        solve(testcase);
    }
    return 0;
}
```

2 Math

2.1 Extended Euclidean Algorithm

```cpp
// ax+by=g, return (g,x,y)
tuple<ll, ll, ll> extended_gcd(ll a, ll b){
  if (a == 0) {b, 0, 1};
  auto [g, x, y] = extended_gcd(b % a, a);
  return {g, y - (b / a) * x, x};
}
```

```cpp
// find x in [0,m) s.t. ax === gcd(a, m) (mod m)
ll modinverse(ll a, ll m) {
    return (get<1>(extended_gcd(a, m))%m+m)%m;
}
```

2.2 Primality Test

```cpp
// O(logn*Logn)
bool is_prime(ll n) {
  if (n < 2 || n % 2 == 0 || n % 3 == 0) return n == 2 || n == 3;
  ll k = __builtin_ctzll(n - 1), d = n - 1 >> k;
  for (ll a : { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 }) {
    ll p = modpow(a % n, d, n), i = k;
    while (p != 1 && p != n - 1 && a % n && i--) p = modmul(p, p, n);
    if (p != n - 1 && i != k) return 0;
  }
  return 1;
}
```

2.3 Integer Factorization (Pollard's rho)

```cpp
ll pollard(ll n) {
  auto f = [n](ll x) { return modadd(modmul(x, x, n), 3, n); };
  ll x = 0, y = 0, t = 30, p = 2, i = 1, q;
  while (t++ % 40 || gcd(p, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if (q = modmul(p, abs(x - y), n)) p = q;
    x = f(x), y = f(f(y));
  }
  return gcd(p, n);
}
// integer factorization
// O(n^0.25 * logn)
vector<ll> factor(ll n) {
  if (n == 1) return {};
  if (is_prime(n)) return { n };
  ll x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  sort(l.begin(), l.end());
  return l;
}
```

2.4 Chinese Remainder Theorem

```cpp
// x = r_i mod m_i
// (y, m) 'x = y mod m' 'm = lcm(m_i)', if not exists return (0, 0)
auto crt = [](auto r, auto m) {
  const int n = r.size(); i64 r0 = 0, m0 = 1;
  for (int i = 0; i < n; i++) {
    i64 r1 = r[i], m1 = m[i];
    if (m0 < m1) swap(r0, r1), swap(m0, m1);
    if (m0 % m1 == 0 && r0 % m1 != r1) return pair(0LL, 0LL);
    if (m0 % m1 == 0) continue;
    i64 g = gcd(m0, m1);
    if ((r1 - r0) % g) return pair(0LL, 0LL);
    i64 u0 = m0 / g, u1 = m1 / g;
    i64 x = (r1 - r0) / g % u1 * modinv(u0, u1) % u1;
    r0 += x * m0, m0 *= u1; if (r0 < 0) r0 += m0;
  }
  return pair(r0, m0);
};
```

2.5 Query of nCr mod M in $O(Q+M)$

```cpp
auto sol_p_e = [](int q, const auto& qs, const int p, const int e, const int mod) {
  // qs[i] = {n, r}, nCr mod p^e in O(p^e)
  vector dp(mod, 1);
  for (int i = 0; i < mod; i++) {
    if (i) dp[i] = dp[i - 1];
    if (i % p == 0) continue;
    dp[i] = mul(dp[i], i);
  }
  auto f = [&](i64 n) {
    i64 res = 0;
    while (n /= p) res += n;
    return res;
  };
  auto g = [&](i64 n) {
    auto rec = [&](const auto& self, i64 n) -> int {
      if (n == 0) return 1;
      int q = n / mod, r = n % mod;
      int ret = mul(self(self, n / p), dp[r]);
      if (q & 1) ret = mul(ret, dp[mod - 1]);
      return ret;
    };
    return rec(rec, n);
  };
  auto bino = [&](i64 n, i64 r) {
    if (n < r) return 0;
    if (r == 0 || r == n) return 1;
    i64 a = f(n) - f(r) - f(n - r);
    if (a >= e) return 0;
    int b = mul(g(n), modinv(mul(g(r), g(n - r)), mod));
    return mul(pow(p, a), b);
  };
  vector res(q, 0);
  for (int i = 0; i < q; i++) {
    auto [n, r] = qs[i];
    res[i] = bino(n, r);
  }
  return res;
};
auto sol = [](int q, const auto& qs, const int mod) {
  vector fac = factor(mod);
  vector r(q, vector(fac.size(), 0));
  vector m(fac.size(), 1);
  for (int i = 0; i < fac.size(); i++) {
    auto [p, e] = fac[i];
    for (int j = 0; j < e; j++) m[i] *= p;
    auto res = sol_p_e(q, qs, p, e, m[i]);
    for (int j = 0; j < q; j++) r[j][i] = res[j];
  }
  vector res(q, 0);
  for (int i = 0; i < q; i++) res[i] = crt(r[i], m).first;
  return res;
};
```

2.6 pelindrome number

```cpp
ll peli(string n) {
    ll len = n.size(), cnt = 0;
    for (int i = 1; i < len; i++) cnt += 9 * pow(10, (i - 1) / 2);
    string half = n.substr(0, (len + 1) / 2);
    ll halfNum = stoll(half), base = pow(10, (len - 1) / 2);
    cnt += halfNum - base;
    string rev = half.substr(0, len / 2);
    reverse(rev.begin(), rev.end());
    string full = half + rev;
    if (full <= n) cnt++;
    return cnt;
```

```
}
```

2.7 Catalan, Derangement, Partition, 2nd Stirling

$C_n = \frac{1}{n+1}\binom{2n}{n}, C_0 = 1, C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}, C_{n+1} = \frac{2(2n+1)}{n+2} C_n$
$D_n = (n-1)(D_{n-1} + D_{n-2}) = n! \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i!}$
$P(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}} (-1)^{k+1} P(n - k(3k-1)/2)$
$= P(n-1) + P(n-2) - P(n-5) - P(n-7) + P(n-12) + P(n-15) - P(n-22) - \cdots$
$P(n,k) = P(n-1,k-1) + P(n-k,k)$, $S(n,k) = S(n-1,k-1) + k\cdot S(n-1,k)$

2.8 Matrix Operations

```
inline bool is_zero(ld a) { return abs(a) < eps; }
// returns {det(A), A^-1, rank(A), tr(A)}
// A becomes invalid after call this O(n^3)
tuple<ld,vector<vector<ld>>,ll,ll> inv_det_rnk(auto A) {
  ld n=A.size(); ld det = 1; vector out(n, vector<ld>(n)); ld tr=0;
  for (int i = 0; i < n; i++) {
    out[i][i] = 1; tr+=A[i][i];
  }
  for (int i = 0; i < n; i++) {
    if (is_zero(A[i][i])) {
      ld maxv = 0;
      int maxid = -1;
      for (int j = i + 1; j < n; j++) {
        auto cur = abs(A[j][i]);
        if (maxv < cur) {
          maxv = cur;
          maxid = j;
        }
      }
      if (maxid == -1 || is_zero(A[maxid][i])) return {0, out, i, tr};
      for (int k = 0; k < n; k++) {
        A[i][k] += A[maxid][k]; out[i][k] += out[maxid][k];
      }
    }
    det *= A[i][i];
    ld coeff = 1.0 / A[i][i];
    for (int j = 0; j < n; j++) A[i][j] *= coeff,out[i][j] *= coeff;
    for (int j = 0; j < n; j++) if (j != i) {
      ld mp = A[j][i];
      for (int k = 0; k < n; k++) A[j][k] -= A[i][k] * mp;
      for (int k = 0; k < n; k++) out[j][k] -= out[i][k] * mp;
    }
  }
  return {det, out, n, tr};
}
```

2.9 Gaussian Elimination

```
const double EPS = 1e-10;
typedef vector<vector<double>> VVD;

// Gauss-Jordan elimination with full pivoting.
// solving systems of linear equations (AX=B)
// INPUT:    a[][] = an n*n matrix
//           b[][] = an n*m matrix
// OUTPUT:   X      = an n*m matrix (stored in b[][])
//           A^{-1} = an n*n matrix (stored in a[][])
// O(n^3)
bool gauss_jordan(VVD& a, VVD& b) {
    const int n = a.size();
    const int m = b[0].size();
    vector<int> irow(n), icol(n), ipiv(n);
    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
        if (fabs(a[pj][pk]) < EPS) return false; // matrix is singular
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        irow[i] = pj;
        icol[i] = pk;

        double c = 1.0 / a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }
    for (int p = n - 1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return true;
}
```

2.10 Permutation and Combination

```
//Permutation
int arr[5] = {1,2,3,4,5};
do{
    for(int i=0;i<5;i++)
        cout << arr[i] << ' ';
    cout <<'\n';
}while(next_permutation(arr,arr+5));
//also prev_permutation exist


//Combination
int arr[5] = {0, 0, 0, 1, 1}; // total : total cnt, 0 cnt : choose cnt
do{
    for(int i=0;i<5;i++)
        if(arr[i] == 0)
            cout << i+1 << ' ';
    cout <<'\n';
}while(next_permutation(arr,arr+5));
```

2.11 Lifting The Exponent

For any integers x, y a positive integer n, and a prime number p such that $p \nmid x$ and $p \nmid y$, the following statements hold:

- When p is odd:
  - If $p \mid x - y$, then $\nu_p(x^n - y^n) = \nu_p(x - y) + \nu_p(n)$.
  - If n is odd and $p \mid x + y$, then $\nu_p(x^n + y^n) = \nu_p(x + y) + \nu_p(n)$.
- When $p = 2$:
  - If $2 \mid x - y$ and n is even, then $\nu_2(x^n - y^n) = \nu_2(x - y) + \nu_2(x + y) + \nu_2(n) - 1$.
  - If $2 \mid x - y$ and n is odd, then $\nu_2(x^n - y^n) = \nu_2(x - y)$.
  - Corollary:
    - If $4 \mid x - y$, then $\nu_2(x + y) = 1$ and thus $\nu_2(x^n - y^n) = \nu_2(x - y) + \nu_2(n)$.
- For all p:

- If $\gcd(n,p)=1$ and $p \mid x-y$, then $\nu_p(x^n-y^n)=\nu_p(x-y)$.
- If $\gcd(n,p)=1$, $p \mid x+y$ and n odd, then $\nu_p(x^n+y^n)=\nu_p(x+y)$.

3 Data Structure

3.1 Lazy Segment Tree

```
struct LazySeg {
    ll n;
    vector<ll> data, tree, lazy;
    LazySeg(ll n): n(n), data(n), tree(n<<2), lazy(n<<2) {}
    void seg_init(ll idx, ll s, ll e) {
        if (s == e) {
            tree[idx] = data[s];
            return;
        }
        ll mid = (s + e) >> 1;
        seg_init(idx<<1, s, mid);
        seg_init(idx<<1|1, mid+1, e);
        tree[idx] = tree[idx<<1] + tree[idx<<1|1];
    }
    void update_lazy(ll idx, ll s, ll e) {
        if (lazy[idx] != 0) {
            tree[idx] += (e-s+1) * lazy[idx];
            if (s != e) {
                lazy[idx<<1] += lazy[idx];
                lazy[idx<<1|1] += lazy[idx];
            }
            lazy[idx] = 0;
        }
    }
    void seg_update(ll idx, ll s, ll e, ll l, ll r, ll d) {
        update_lazy(idx, s, e);
        if (l > e || r < s) return;
        if (l <= s && e <= r) {
            tree[idx] += (e-s+1) * d;
            if (s != e) {
                lazy[idx<<1] += d;
                lazy[idx<<1|1] += d;
            }
            return;
        }
        ll mid = (s + e) >> 1;
        seg_update(idx<<1, s, mid, l, r, d);
        seg_update(idx<<1|1, mid+1, e, l, r, d);
        tree[idx] = tree[idx<<1] + tree[idx<<1|1];
    }
    ll seg_query(ll idx, ll s, ll e, ll l, ll r) {
        update_lazy(idx, s, e);
        if (l > e || r < s) return 0;
        if (l <= s && e <= r) return tree[idx];
        ll mid = (s + e) >> 1;
        ll lsum = seg_query(idx<<1, s, mid, l, r);
        ll rsum = seg_query(idx<<1|1, mid+1, e, l, r);
        return lsum + rsum;
    }
    // seg.init(v);
    void init(const vector<ll>&v) {
        data = v;
        seg_init(1, 0, n-1);
    }
    // seg.update(l-1, r-1, d);
    void update(ll l, ll r, ll d) {
        seg_update(1, 0, n-1, l, r, d);
    }
    // seg.query(l-1, r-1);
    ll query(ll l, ll r) {
        if (l > r) return 0;
        return seg_query(1, 0, n-1, l, r);
    }
};
```

3.2 Persistent Segment Tree

```
struct PST{
    ll n;
    vector<ll> data;
    vector<vector<pll>> tree;
    PST(ll n):n(n), data(n), tree(4*n) {}
    void seg_init(ll idx, ll s, ll e){
        if(s==e){
            tree[idx].push_back({0, data[s]});
            return;
        }
        ll mid=(s+e)>>1;
        seg_init(idx<<1, s, mid);
        seg_init(idx<<1|1, mid+1, e);
        tree[idx].push_back({0, tree[idx<<1].back().second+tree[idx<<1|1].back().second});
    }
    void seg_update(ll idx, ll s, ll e, ll pos, ll val, ll ord){
        if(pos<s || pos>e) return;
        if(s==e){
            tree[idx].push_back({ord, val});
            return;
        }
        ll mid=(s+e)>>1;
        seg_update(idx<<1, s, mid, pos, val, ord);
        seg_update(idx<<1|1, mid+1, e, pos, val, ord);
        tree[idx].push_back({ord, tree[idx<<1].back().second+tree[idx<<1|1].back().second});
    }
    ll seg_query(ll idx, ll s, ll e, ll l, ll r, ll ord){
        if(l>e || r<s)return 0;
        if(l<=s && e<=r) {
            return prev(ranges::lower_bound(tree[idx], pll(ord, LLONG_MAX)))->second;
        }
        ll mid=(s+e)>>1;
        return seg_query(idx<<1, s, mid, l, r, ord)
                +seg_query(idx<<1|1, mid+1, e, l, r, ord);
    }
    void init(const vector<ll>&arr){
        data=arr;
        seg_init(1, 0, n-1);
    }
    void update(ll pos, ll val, ll ord){
        seg_update(1, 0, n-1, pos, val, ord);
    }
    ll query(ll l, ll r, ll ord){
        if(l>r)return 0;
        else return seg_query(1, 0, n-1, l, r, ord);
    }
};
```

4 Graph

4.1 Dijkstra

```
// O(ElogV)
vector<ll> dijk(ll n, ll s){
  vector<ll>dis(n,INF);
  priority_queue<pll, vector<pll>, greater<pll> > q; // pair(dist, v)
  dis[s] = 0;
  q.push({dis[s], s});
  while (!q.empty()){
```

```
    while (!q.empty() && visit[q.top().second]) q.pop();
    if (q.empty()) break;
    ll next = q.top().second; q.pop();
    visit[next] = 1;
    for (ll i = 0; i < adj[next].size(); i++)
      if (dis[adj[next][i].first] > dis[next] + adj[next][i].second){
        dis[adj[next][i].first] = dis[next] + adj[next][i].second;
        q.push({dis[adj[next][i].first], adj[next][i].first});}}
  for(ll i=0;i<n;i++)if(dis[i]==INF)dis[i]=-1;
  return dis;
}
```

4.2 LCA

```
const int MAXN = 100;
const int MAXLN = 9;
vector<int> tree[MAXN];
int depth[MAXN];
int par[MAXLN][MAXN];

void dfs(int nod, int parent) {
    for (int next : tree[nod]) {
        if (next == parent) continue;
        depth[next] = depth[nod] + 1;
        par[0][next] = nod;
        dfs(next, nod);
    }
}

void prepare_lca() {
    const int root = 0;
    dfs(root, -1);
    par[0][root] = root;
    for (int i = 1; i < MAXLN; ++i)
        for (int j = 0; j < n; ++j)
            par[i][j] = par[i - 1][par[i - 1][j]];
}

// find lowest common ancestor in tree between u & v
// assumption : must call 'prepare_lca' once before call this
// O(logV)
int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXLN - 1; i >= 0; --i)
            if (depth[u] - (1 << i) >= depth[v])
                u = par[i][u];
    }
    if (u == v) return u;
    for (int i = MAXLN - 1; i >= 0; --i) {
        if (par[i][u] != par[i][v]) {
            u = par[i][u];
            v = par[i][v];
        }
    }
    return par[0][u];
}
```

4.3 Centroid Decomposition

```
// O(n lg n) for centroid decomposition
auto cent_decom = [](const auto& adj) {
  const int n = adj.size() - 1;
  vector sz(n + 1, 1), dep(n + 1, 0), par(n + 1, 0);
  auto dfs = [&](const auto& self, int cur, int prv) -> void {
    for (auto [nxt, cost] : adj[cur]) {
      if (nxt == prv) continue;
      self(self, nxt, cur);
      sz[cur] += sz[nxt];
    }
  };
  auto adjust = [&](int cur) {
    while (1) {
      int f = 0;
      for (auto [nxt, cost] : adj[cur]) {
        if (dep[nxt] || sz[cur] >= 2 * sz[nxt]) continue;
        sz[cur] -= sz[nxt], sz[nxt] += sz[cur];
        cur = nxt, f = 1;
        break;
      }
      if (!f) return cur;
    }
  };
  auto rec = [&](const auto& self, int cur, int prv) -> void {
    cur = adjust(cur);
    par[cur] = prv;
    dep[cur] = dep[prv] + 1;
    for (auto [nxt, cost] : adj[cur]) {
      if (dep[nxt]) continue;
      self(self, nxt, cur);
    }
  };
  dfs(dfs, 1, 0);
  rec(rec, 1, 0);
  return pair(dep, par);
};
```

4.4 Minimum Spanning Tree

```
// O(ElogV)
ll prim() {
  priority_queue<pll, vector<pll>, greater<pll> > q;
  ll count = 0; ll ret = 0;
  q.push(make_pair(0, 0)); // (cost, vertex)
  while (!q.empty()){
    ll x = q.top().second; // also able to get edges
    visit[x] = 1; ret += q.top().first; q.pop(); count++;
    for (ll i = 0; i < adj[x].size(); i++)
      q.push({adj[x][i].second, adj[x][i].first});
    while (!q.empty() && visit[q.top().second]) q.pop();
  }
  if (count != n) return -1;
  else return ret;
}

ll Kruskal(){
  ll ret = 0;vector<ll>par;
  iota(par.beging(),par.end(),0);
  vector<pair<ll, pll>> e;
  for(ll i= 0; i < n; i++)
    for(ll j=0; j < adj[i].size(); j++)
      e.push_back({adj[i][j].second, {i, adj[i][j].first}});
  sort(e.begin(), e.end());
  for(ll i=0; i < e.size(); i++){
    ll x = e[i].second.first,y = e[i].second.second;
    if(find(x) != find(y)){
      union(x, y);
      ret += e[i].first;
    }
  }
  ll p=find(0);
  for(ll i=1;i<n;i++){
```

```
    if(find(i)!=p)return -1;
  }
  else return ret;
}
```

4.5 Offline Dynamic Connectivity

```
struct OFDC {
    vector<tlll> query;
    vector<ll> grp, sz;
    vector<vector<pll>> tree;
    map<pll, ll> conn;
    ll n, q;
    OFDC(ll n, ll q): n(n), q(q), query(q+1), grp(n+1), sz(n+1, 1), tree(4*(q+1)) {
        iota(grp.begin(), grp.end(), 0);
    }
    void update(ll node, ll s, ll e, ll l, ll r, pll edge) {
        if (r < s || e < l) return;
        if (l <= s && e <= r) {
            tree[node].push_back(edge);
            return;
        }
        ll mid = (s + e) >> 1;
        update(node << 1, s, mid, l, r, edge);
        update(node << 1 | 1, mid + 1, e, l, r, edge);
    }
    ll _find(ll x) {
        if (grp[x] == x) return x;
        return _find(grp[x]);
    }
    pll _union(ll x, ll y) {
        x = _find(x), y = _find(y);
        if (x == y) return {-1, -1};
        if (sz[x] < sz[y]) swap(x, y);
        grp[y] = x;
        sz[x] += sz[y];
        return {x, y};
    }
    void _delete(ll u, ll v) {
        sz[u] -= sz[v];
        grp[v] = v;
    }
    void dfs(ll node, ll s, ll e) {
        vector<pll> rconn;
        for (auto& [u, v]: tree[node]) {
            auto [x, y] = _union(u, v);
            if (x != -1) rconn.push_back({x, y});
        }
        if (s == e) {
            if (get<0>(query[s]) == 3) {
                cout << (_find(get<1>(query[s])) ==
                    _find(get<2>(query[s]))) << '\n';
            }
        } else {
            ll mid = (s + e) >> 1;
            dfs(node << 1, s, mid);
            dfs(node << 1 | 1, mid + 1, e);
        }
        for (auto& [u, v]: rconn) {
            _delete(u, v);
        }
    }
    void run() {
        for (ll i = 0; i < q; i++) {
            auto& [type, u, v] = query[i];
            cin >> type >> u >> v;
            if (u > v) swap(u, v);
            if (type == 1) {
                conn[{u, v}] = i;
            } else if (type == 2) {
                update(1, 0, q, conn[{u, v}], i, {u, v});
                conn.erase({u, v});
            }
        }
        for (auto&[edge, time] : conn) {
            auto&[u, v] = edge;
            update(1, 0, q, time, q, {u, v});
        }
        dfs(1, 0, q);
    }
};
```

5 String

5.1 KMP

```
void calculate_pi(vector<int>& pi, const string& str) {
  pi[0] = -1;
  for (int i = 1, j = -1; i < str.size(); i++) {
    while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
    if (str[i] == str[j + 1]) pi[i] = ++j;
    else pi[i] = -1;
  }
}
// returns all positions matched
// O(|text|+|pattern|)
vector<int> kmp(const string& text, const string& pattern) {
  vector<int> pi(pattern.size()), ans;
  if (pattern.size() == 0) return ans;
  calculate_pi(pi, pattern);
  for (int i = 0, j = -1; i < text.size(); i++) {
    while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
    if (text[i] == pattern[j + 1]) {
      j++;
      if (j + 1 == pattern.size()) ans.push_back(i - j), j = pi[j];
    }
  }
  return ans;
}
```

5.2 Z Algorithm

```
// Z[i] : maximum common prefix length of &s[0] and &s[i] with O(|s|)
auto get_z = [](const string& s) {
  const int n = s.size(); vector z(n, 0); z[0] = n;
  for (int i = 1, l = -1, r = -1; i < n; i++) {
  if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
    if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
  }
  return z;
};
```