1. Template

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

#define endl '\n'
#define ll long long int
#define dl double
#define ld long double
#define ff __float128
#define fore(i,a,b) for (int i = a; i < b; i++)
#define fi first
#define se second
#define pb push_back
#define all(v) v.begin(), v.end()
#define fast_io
↪  ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ll> vll;
typedef vector<vll> vvll;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef vector<pll> vpll;

const int inf = 1<<30;
const int mod = 1e9+7;

// clear && g++ -std=c++17 -O2 -Wall template.cpp -o template &&
↪  ./template
// ifstream cin("input.txt"); ofstream cout("output.txt");
```

2. Data Structure

2.1. Segment tree with lazy

```cpp
struct Node{
  int l, r;
  ll sum;
  int mark;
  ll lazy;
};//0 +, 1 *, 0 gcd, 1 mcm
const int Neutro = 0;
template<typename TT> struct SegmentTree{
  int n, h;
  vector<Node> st;

  SegmentTree  (int m,  vector<TT> &values) : n(m){
    h = 1 << ((int)( ceil(log2(n)) + 1));
    st.resize( h );
    build(1, 1, n, values);
  }
  TT merge(TT l, TT r){ return l + r; }//for query
  TT getValue(int curr){ return st[curr].sum; }//same^^

  int left(int n){ return (n << 1);}
  int right(int n){return (n << 1) | 1; }

  void initLeaf(int curr, TT value){
    st[curr].mark = 0; st[curr].lazy = Neutro;
    st[curr].sum = value;//
  }
  void updateFromChildren(int curr){//
    st[curr].sum = st[left(curr)].sum + st[right(curr)].sum;
  }
  void updateNodeLazy(int curr, TT value){//updates lazy
    int l = st[curr].l, r = st[curr].r;
    st[curr].sum += (r - l + 1) * value;//
    st[curr].mark = 1; st[curr].lazy += value;//
  }
  void propagateToChildren(int curr){//propagate lazy
    if(st[curr].mark != 0){
      updateNodeLazy(left(curr), st[curr].lazy );
      updateNodeLazy(right(curr), st[curr].lazy);
      st[curr].mark = 0; st[curr].lazy = Neutro;
    }
  }
  void build(int curr, int l, int r, vector<TT> &values){
    st[curr].l = l; st[curr].r = r;
    if(l == r) {
      initLeaf(curr, values[l]);//
    }else{
      int m = ((r - l) >> 1) + l;
```

```
      build(left(curr), l, m, values);
      build(right(curr), m + 1, r, values);
      updateFromChildren(curr);
    }
  }
  void rangeUpdate(int curr, int l, int r, int ql, int qr, TT
↪  value){
    if( r < ql || qr < l ) return;
    else if( ql <= l && r <= qr){
      updateNodeLazy(curr, value);
    }else{
      propagateToChildren(curr);
      int m = ((r - l) >> 1) +  l;
      rangeUpdate(left(curr), l, m, ql, qr, value);
      rangeUpdate(right(curr), m + 1, r, ql, qr, value);
      updateFromChildren(curr);
    }
  }// not lazy
  // void pointUpdate(int curr, int l, int r, int pos, TT value){
  //   if(l == r){
  //     st[curr].sum += value;
  //   }else{
  //     int m = ((r - l) >> 1) +  l;
  //     if(pos <= m) pointUpdate(left(curr), l, m, pos, value);
  //     else pointUpdate(right(curr), m + 1, r, pos, value);
  //     updateFromChildren(curr);
  //   }
  // }
  TT rangeQuery(int curr, int l, int r, int ql, int qr){
    if( r < ql || qr < l ) return Neutro;
    else if( ql <= l && r <= qr){
      return getValue(curr);
    }else{
      propagateToChildren(curr);
      int m = ((r - l) >> 1) + l;
      return merge( rangeQuery(left(curr), l, m, ql, qr),
↪  rangeQuery(right(curr), m+1, r, ql, qr)) ;
    }
  }
  void update( int ql, int qr, int value){
    rangeUpdate(1, 1, n, ql, qr, value);
  }
  TT query(int ql, int qr){
    return rangeQuery(1, 1, n, ql, qr);
  }
  // void printST(){
  //   cout << endl << "st = ";
  //   fore(i,0,h) cout << st[i].sum << ' '; cout << endl;
  // }
};
// vector<ll> nums(n + 1);
// SegmentTree<ll>* st = new SegmentTree<ll>(n, nums);
// st->update(l,r,x); st->query(l, r);
```

2.2. SQRT and MO's

```
//SQRT decomposition
//if RTE, change limits to min(br, n)
template <typename TT>
struct SQRT{
  int n, s;
  TT neutro = 0;
  vector<TT> A, B;
  vector<TT> lazy, marks;

  SQRT(int m, vector<TT> &arr): n(m){
    s = sqrt(n) + 1;//puede variar
    A.assign(n, neutro);
    B.assign(n / s + 1, neutro);
    lazy.assign(s, neutro); marks.assign(s, neutro);
    fore(i,0,n){ A[i] = arr[i]; B[i/s] += arr[i]; }
  }
  void pushLazy(int block){
    if(marks[block]){
      fore(i,block,(block+1) * s && i < n) A[i] += lazy[block];
      lazy[block] = neutro; marks[block] = 0;
    }
  }
  void rangeUpdate(int l, int r, TT value){
    int bl = l/s, br = r/s;
    if(bl == br){
      pushLazy(bl);
      fore(i,l,r+1) A[i] += value;
      TT res = neutro;
      fore(i, bl*s, (bl+1) * s && i < n) res += A[i];
      B[bl] = res;
    }else{
```

```
      pushLazy(bl);pushLazy(br);
      fore(i,l,(bl+1) * s){ A[i] += value; B[bl] += value;}
      fore(i,bl+1, br)    { B[i] += s * value; lazy[i] += value;
↪  marks[i] = 1;}
      fore(i,br * s, r+1) { A[i] += value; B[br] += value;}
    }
  }
  void pointUpdate(int idx, TT value){//not lazy
    int block = idx / s;
    A[idx] = value;
    TT res = neutro;
    fore(i, block * s, (block + 1) * s && i < n) res += A[i];
    B[block] = res;
  }
  TT rangeQuery(int l, int r){
    int bl = l/s, br = r/s;
    TT res = 0;
    if(bl == br){
      pushLazy(bl);
      fore(i,l,r+1) res += value;
    }else{
      pushLazy(bl); pushLazy(br);
      fore(i,l,(bl+1) * s) res += A[i];
      fore(i,bl+1, br)     res += B[i];
      fore(i,br * s, r+1)  res += A[i];
    }
  }
};
//MO's algorithm
ll answer, neutro = 0; vll arr;
struct MOquery{
  int l, r, index, S;
  MOquery(int l, int r, int idx, int S): l(l), r(r), index(idx),
↪  S(S){}
  bool operator<(const MOquery & q) const{
    int bl = l / S, bq = q.l / S;
    if(bl == bq) return r < q.r;
    return bl < bq;
  }
};
void add(int idx){
  answer += arr[idx];
}
void remove(int idx){
  answer -= arr[idx];
}
vector<ll> MO(vector<MOquery> & queries){
  vector<ll> ans(queries.size());
  sort(queries.begin(), queries.end());
  ll current = 0;
  int prevL = 0, prevR = -1;
  int i, j;
  answer = neutro;
  for(const MOquery & q : queries){
    while (prevL > q.l) { prevL--; add(prevL); }
    while (prevR < q.r) { prevR++; add(prevR); }
    while (prevL < q.l) { remove(prevL); prevL++; }
    while (prevR > q.r) { remove(prevR); prevR--;}
    ans[q.index] = answer;
  }
  return ans;
}
```

2.3. Convex Hull Trick

```
const int MX = 200005;
const ll inf = 1e18;

bool Q = 0;
struct Line {
  mutable ll m, b, x;
  // Maximo: m < ot.m
  // Minimo: m > ot.m
  bool operator < (const Line ot) const {
    return Q ? x < ot.x : m < ot.m;
  }
};

ll ceil (ll a, ll b) {
  if (a < 0 != b < 0) return a / b;
  return (abs(a) + abs(b) - 1) / abs(b);
}

ll intersection (const Line &p, const Line &q) {
  return ceil(q.b - p.b, p.m - q.m);
}
```

```
struct Hull : multiset<Line> {
  bool valid (auto it) {
    if (it == begin()) {
      auto sig = it; sig++;
      if (sig != end()) sig->x = intersection(*it, *sig);
      return it->x = -inf;
    }
    auto ant = it, sig = it;
    ant--, sig++;
    if (sig == end()) {
      it->x = intersection(*it, *ant);
      return 1;
    }
    ll x = intersection(*it, *ant);
    ll y = intersection(*it, *sig);
    if (x > y) return 0;
    it->x = x, sig->x = y;
    return 1;
  }
  void add (ll m, ll b) {
    auto it = lower_bound({m, b, -inf});
    if (it != end() && it->m == m) {
      //Maximo: it->b > b
      //Minimo: it->b < b
      if (it->b > b) return;
      it->b = b;
    } else  { it = insert({m, b, -inf}); }

    if (!valid(it)) { erase(it); return; }
    auto ant = it;
    while (ant != begin()) {
      if (valid(--ant)) break;
      erase(ant);
      if (it == begin()) { it->x = -inf; break; }
      ant = it;
    }
    auto sig = it; sig++;
    while (sig != end() && !valid(sig)) erase(sig++);
  }

  ll query (ll x) {
    if (empty()) return 0;
    Q = 1;auto it = upper_bound({0, 0, x});
    it--;
    Q = 0;return x * it->m + it->b;
  }
};
```

2.4. Field extension

```
int sq = 5;
// const lli sqrt5 = 383008016;//mod1e9+9
const ll mod = 1000000007;//is important to be CONST
struct EX {
  ll re, im;
  EX (ll re = 0, ll im = 0) : re(re), im(im){}

  EX& operator = (EX oth) {
    return re = oth.re, im = oth.im, *this;
  }
  int norm () const {
    return trim((1ll * re * re - 1ll * sq * im % mod * im) % mod);
  }
  EX conj () const {
    return {re, trim(-im)};
  }
  EX operator * (EX ot) const {
    return {
      int((1ll * re * ot.re + 1ll * sq * im % mod * ot.im) % mod),
      int((1ll * re * ot.im + 1ll * im * ot.re) % mod)
    };
  };
  EX& operator *= (const EX& ot) {
    *this = *this * ot; return *this;
  }
  EX operator * (ll k) const {
    k = ((k % mod) + mod ) % mod;
    return { (re * k) % mod, (im * k) % mod };
  };
  EX operator / (ll n) const {
    return { re * inv(n) % mod, im * inv(n) % mod };
  }
  EX operator / (EX ot) const {
    return *this * ot.conj() / ot.norm();
  }
  EX& operator /= (const EX& ot) {
    *this = *this / ot; return *this;
```

```
  }
  EX operator + (EX ot) const {
    return { trim(re + ot.re),  trim(im + ot.im) };
  }
  EX& operator += (const EX& ot) {
    *this = *this + ot; return *this;
  }
  EX operator - (EX ot) const {
    return { trim(re - ot.re),  trim(im - ot.im) };
  }
  EX& operator -= (const EX& ot) {
    *this = *this - ot; return *this;
  }
  EX pow (ll p) const {
    EX res(1), b = *this;
    while (p) {
      if (p & 1) res *= b; b *= b; p /= 2;
    }
    return res;
  }

  bool operator == (EX ot) const {
    return re == ot.re && im == ot.im;
  }
  bool operator != (EX ot) const {
    return !(*this == ot);
  }

  static ll trim(ll a) {
    if (a >= mod) a -= mod;
    if (a < 0) a += mod;
    return a;
  }
  static ll inv (ll b) {
    ll res = 1, p = mod - 2;
    while (p) {
      if (p & 1) res = 1ll * res * b % mod;
      b = 1ll * b * b % mod;
      p /= 2;
    }
    return res;
  };
};
```

2.5. BIT

```
struct Fenwick {
    int n;
    vector<long long> tree;

    Fenwick(int _n) : n(_n), tree(n + 1, 0) {}

    void update(int idx, long long val) {
        for (; idx <= n; idx += idx & -idx) {
            tree[idx] += val;
        }
    }

    long long query(int idx) {
        long long ret = 0;
        for (; idx > 0; idx -= idx & -idx) {
            ret += tree[idx];
        }
        return ret;
    }

    long long query(int x, int y) { return query(y) - query(x -
↪  1); }
};
```

2.6. merge sort tree

```
vi tree[400000];
vi vv;
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        tree[v] = vi(1, a[tl]);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        merge(tree[v*2].begin(), tree[v*2].end(),
↪  tree[v*2+1].begin(), tree[v*2+1].end(),
↪  back_inserter(tree[v]));
    }
}
void query(int v, int tl, int tr, int l, int r, int x){
```

```
    if(l > r) return;
    if(tr <= r){
    for(auto i = tree[v].begin(); i < tree[v].end(); i++){
      if(*i <= x) vv.pb(*i);
      else break;
    }
    return;
  }
  if(tl > r) return;
  int tm = (tl + tr) / 2;
  query(v*2, tl, tm, l, r, x),
  query(v*2+1, tm+1, tr, l, r, x);
  return;
}
```

3. FFT

3.1. FFT

```
using cd = complex<double>;
const double PI = acos(-1);


void fft(vector<cd> & a, int inv) {
    int n = a.size();
   for(int i = 1, j = 0; i < n - 1; ++i){
    for(int k = n >> 1; (j ^= k) < k; k >>= 1);
    if(i < j) swap(a[i], a[j]);
  }

  vector<cd> w(n >> 1);

    for (int k = 2; k <= n; k <<= 1) {
        //   cd w1 = polar(1.0,  2 * PI / k * inv) ;
        w[0] = 1;
        for (int j = 1; j < k >> 1; j++) // best precision but
↪  slower
            w[j] = polar(1.0, 2 * j * PI / k * inv);
          // w[j] = w[j-1]*w1;
        for (int i = 0; i < n; i += k) {
            for (int j = 0; j < k >> 1; j++) {
                cd u = a[i + j], v = a[i + j + (k >> 1)] * w[j];
                a[i + j] = u + v;
                a[i + j + (k >> 1)] = u - v;
            }
        }
    }
    if (inv == -1) for (cd & x : a) x /= n;
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size() - 1)  n <<= 1;
  fa.resize(n); fb.resize(n);

  fft(fa, 1); fft(fb, 1);
  fore(i,0,n) fa[i] *= fb[i];
    fft(fa, -1);

    vector<int> result(n);
    for (int i = 0; i < n; i++) result[i] = round(fa[i].real());
    return result;
}
```

4. Binary Search

```
int lowerBound(vi &nums, int a) {
    int l = 0, r = nums.size() - 1;
    while(l <= r) {
        int m = ((r - l) >> 1) + l;
  //  if(nums[m] == a) return m;//binary
    nums[m] < a ? l = m + 1 : r = m - 1;  //lower & binary
  //  nums[m] <= a ? l = m + 1 : r = m - 1; //upper
    }
    return l;//return -1; //binary
}


for(int j = 0;j < 300; j++){
  ld mid1 = l + (r - l) / 3, mid2 = r - (r - l) / 3;
  ld f1 = f(p0, Friend1[i + 1], p1, Friend2[i + 1], mid1);
  ld f2 = f(p0, Friend1[i + 1], p1, Friend2[i + 1], mid2);
  if(f1 >= f2) l = mid1;
```

```
  else r = mid2;
}
```

5. Flujos

5.0.1. Dinic

```
typedef tuple<int, ll, ll> edge;
class max_flow{
  private:
    int V;
    vector<edge> EL;
    vvi AL;
    vi d, last;
    vpii p;
    bool BFS(int s, int t){
      d.assign(V, -1);d[s] = 0;
      queue<int> q({s});
      p.assign(V, {-1, -1});
      while( !q.empty()){
        int u = q.front(); q.pop();
        if( u== t) break;
        for(auto &idx: AL[u]){
          auto &[v, cap, flow] = EL[idx];
          if( (cap - flow > 0) && d[v] == -1){
            d[v] = d[u] + 1, q.push(v), p[v] = {u, idx};
          }
        }
      }
      return d[t] != -1;
    }
    ll send_one_flow(int s, int t, ll f = inf){
      if  ( s == t) return f;
      auto &[u,idx] = p[t];
      auto &cap = get<1>(EL[idx]), &flow = get<2>(EL[idx]);
      ll pushed = send_one_flow(s, u, min(f, cap-flow));
      flow += pushed;
      return pushed;
    }
    ll DFS(int u, int t, ll f = inf){
      if( (u == t) || (f == 0)) return f;
      for(int &i = last[u]; i < (int) AL[u].size(); ++i){
        auto &[v, cap, flow] = EL[AL[u][i]];
        if(d[v] != d[u] + 1) continue;
        if(ll pushed = DFS(v, t, min(f, cap - flow))){
          flow += pushed;
          auto &rflow = get<2> (EL[AL[u][i] ^ 1]);
          rflow -= pushed;
          return pushed;
        }
      }
      return 0;
    }
  public:
    max_flow(int initialV) : V(initialV){
      EL.clear();
      AL.assign(V, vi());
    }
    void add_edge(int u, int v, ll w, bool directed = true){
      if( u == v) return;
      EL.emplace_back(v, w, 0);
      AL[u].pb(EL.size() - 1);
      EL.emplace_back(u, directed ? 0 : w, 0);
      AL[v].pb(EL.size() - 1);
    }
    ll edmonds_karp(int s, int t){
      ll mf = 0;
      while( BFS(s,t)){
        ll f = send_one_flow(s, t);
        if ( f == 0)break;
        mf += f;
      }
      return mf;
    }
    ll dinic(int s, int t ){
      ll mf = 0;
      while( BFS(s,t)){
        last.assign(V, 0);
        while( ll f = DFS(s,t)){
          mf += f;
        }
      }
      return mf;
    }
};
```

5.0.2. Ford-Fulkerson

```
const int sink = 37;
int C[50][50], F[50][50], visited[50];
int sendFlow(int node, int bottleneck){
  if(node == sink){
    return bottleneck;
  }
  visited[node] = true;
  fore(i,0,sink+1){
    int f = C[node][i] - F[node][i];
    if(f>0 && !visited[i]){
      f = sendFlow(i, min(f, bottleneck));
      if(!f) continue;
      F[node][i] += f;
      F[i][node] -= f;
      return f;
    }
  }
  return 0;
}
```

6. Strings

6.1. aho-corasik

```
//#define feach(f, g) for(auto &f: g)
const int N=1e5+10, MOD=1e9+7, SIG=26;
int id=1, dp[N];
string s;
vector<int> adj[2*N];

struct node{
  int fail,ch[SIG]={};
  vector<int> lens;
}t[2*N];
void insert(string s){
  int u=1;
  for(auto &c: s){
    c-='a';
    if(!t[u].ch[c]) t[u].ch[c]=++id;
    u=t[u].ch[c];
  }
  t[u].lens.pb(s.size());
}
void dfs(int u){
  t[u].lens.insert(t[u].lens.end(), t[t[u].fail].lens.begin(),
↪  t[t[u].fail].lens.end());
  for(auto &v: adj) dfs(v);
}
void build(){
  queue<int> q;
  int u=1;
  t[1].fail=1;
  fore(i,0,SIG) {
    if(t[u].ch[i]) t[t[u].ch[i]].fail=u, q.push(t[u].ch[i]);
    else t[u].ch[i]=1;
  }
  while(!q.empty()){
    u=q.front(); q.pop();
    fore(i,0,SIG){
      if(t[u].ch[i]) t[t[u].ch[i]].fail=t[t[u].fail].ch[i],
↪  q.push(t[u].ch[i]);
      else t[u].ch[i]=t[t[u].fail].ch[i];
    }
  }
  fore(i,2,id+1) adj[t[i].fail].pb(i);
  dfs(1);
}
```

7. Math

7.1. nCr

```
vll fact(100007,0), inv(100007,0), invfact(100007,0);
void factorial(){
  fact[0] = 1; inv[0] = inv[1] = 1; invfact[0] = 1;
  fore(i,1,100005) fact[i] = (fact[i-1] * i) % MOD;
  fore(i,2,100005) inv[i] = inv[MOD % i] * (MOD - MOD / i) %
↪  MOD;
  fore(i,1,100005) invfact[i] = invfact[i-1] * inv[i] % MOD;
}
ll ncr(ll n, ll r){
  return fact[n] * invfact[n - r] % MOD * invfact[r] % MOD ;
```

```
}
```

7.2. Discrete Root

```
int generator(int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i); while (n % i == 0) n /= i;
        }
    }
    if (n > 1) fact.push_back(n);
    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (int factor : fact) {
            if (powmod(res, phi / factor, p) == 1) {
                ok = false; break;
            }
        }
        if (ok) return res;
    }
    return -1;
}
vi rootK() {// finds all numbers x such that x^k = a (mod n)
    int n, k, a;
    scanf("%d %d %d", &n, &k, &a);
    if (a == 0) return vi(1,1);
    int g = generator(n);
    // Baby-step giant-step discrete logarithm algorithm
    int sq = (int) sqrt (n + .0) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
        dec[i-1] = {powmod(g, i * sq * k % (n - 1), n), i};
    sort(dec.begin(), dec.end());
    int any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        int my = powmod(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(dec.begin(), dec.end(),
↪   make_pair(my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;break;
        }
    }
    if (any_ans == -1) return vi(1,-1);
    // Print all possible answers
    int delta = (n-1) / gcd(k, n-1);
    vi ans;
    for (int cur = any_ans % delta; cur < n-1; cur += delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```