



Chulalongkorn University

# Vaporised

Teetat T., Borworntat D., Pakapim E.

template from KACTL

2024-08-02

1	Template
2	Mathematics
3	Numerical
4	Data Structures
5	Number Theory
6	Graph
7	Polynomials

## Template (1)

template.cpp	27 lines
#pragma once	
#include <bits/stdc++.h> #define sz(x) (int)(x).size() #define all(x) (x).begin(), (x).end()	
using namespace std;	
using ll = long long; using db = long double; using vi = vector<int>; using vl = vector<ll>; using vd = vector<db>; using pii = pair<int, int>; using pll = pair<ll, ll>; using pdd = pair<db, db>; const int INF = 0x3fffffff; // const int MOD=1000000007; const int MOD = 998244353; const ll LINF = 0x1fffffffffffffff; const db DINF = numeric_limits<db>::infinity(); const db EPS = 1e-9; const db PI = acos(db(-1));	
int main(){ cin.tie(nullptr)->sync_with_stdio(false); }	
c.sh	2 lines
g++ -std=gnu++2a -Wall \$1 -o a.out ./a.out	

## Mathematics (2)

### 2.1 Goldbatch’s Conjecture

- Even number can be written in sum of two primes (Up to 1e12)
- Range of  $N^{th}$  prime and  $N + 1^{th}$  prime will be less than or equal to 300 (Up to 1e12)

### 2.2 Divisibility

Number of divisors of  $N$  is given by  $\prod_{i=1}^k (a_i + 1)$  where  $N = \prod_{i=1}^k p_i^{a_i}$  and  $p_i$  are prime factors of  $N$ .

## Numerical (3)

### 3.1 Newton’s Method

if  $F(Q) = 0$ , then  $Q_{2n} \equiv Q_n - \frac{F(Q_n)}{F'(Q_n)} \pmod{x^{2n}}$

$$Q = P^{-1} : Q_{2n} \equiv Q_n \cdot (2 - P \cdot Q_n^2) \pmod{x^{2n}}$$

$$Q = \ln P = \int \frac{P'}{P} dx$$

$$Q = e^P : Q_{2n} \equiv Q_n(1 + P - \ln Q_n) \pmod{x^{2n}}$$

$$Q = \sqrt{P} : Q_{2n} \equiv \frac{1}{2}(Q_n + P \cdot Q_n^{-1}) \pmod{x^{2n}}$$

$$Q = P^k = \alpha^k x^{kt} e^{k \ln T}; P = \alpha \cdot x^t \cdot T, T(0) = 1$$

## Data Structures (4)

OrderedSet.hpp	
<b>Description:</b> Ordered Set	
\"../template/Header.hpp\", <bits/extc++.h>	1a7f5f, 14 lines
using namespace __gnu_pbds;	
template <class T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>; // can be change to less_equal	
void usage() { ordered_set<int> st, st_2; st.insert(2); st.insert(1); cout << st.order_of_key(2); cout << *st.find_by_order(1); st.join(st_2); // merge }	
FenwickTree.hpp	
<b>Description:</b> Fenwick / Binary Indexed Tree	
43767a, 41 lines	
template<class T> struct Fenwick{ int n, logn; vector<T> t; Fenwick(){} Fenwick(int _n){init(vector<T>(_n, T{}));} template<class U> Fenwick(const vector<U> &a){init(a);} template<class U> void init(const vector<U> &a){ n=(int)a.size(); logn=31-__builtin_clz(n); t.assign(n+1, T{}); for(int i=1; i<=n; i++){ t[i]=t[i]+a[i-1]; int j=i+(i&-i); if(j<=n)t[j]=t[j]+t[i];	

```

    }
}
void update(int x, const T &v) {
    for (int i = x + 1; i <= n; i += i & - i) t[i] = t[i] + v;
}
void update(int l, int r, const T &v) {
    update(l, v), update(r + 1, -v);
}
T query(int x) {
    T res{};
    for (int i = x + 1; i > 0; i -= i & - i) res = res + t[i];
    return res;
}
T query(int l, int r) {
    return query(r) - query(l - 1);
}
int find(const T &k) {
    int x = 0;
    T cur{};
    for (int i = 1 << logn; i > 0; i >= 1)
        if (x + i <= n && cur + t[x + i] <= k) x += i, cur = cur + t[x];
    return x;
}
};

```

## SegmentTree.hpp

**Description:** Segment Tree

c51dec, 85 lines

```

template<class Monoid>
struct SegmentTree {
    using T = typename Monoid::value_type;
    int n;
    vector<T> t;
    SegmentTree() {}
    SegmentTree(int n, function<T(int)> create) { init(n, create); }
    SegmentTree(int n, T v = Monoid::unit()) { init(n, [&](int) {
        return v; }); }
    template<class U>
    SegmentTree(const vector<U> &a) { init((int)a.size(), [&](int
        i) { return T(a[i]); }); }
    void init(int _n, function<T(int)> create) {
        n = _n;
        t.assign(4 << (31 - __builtin_clz(n)), Monoid::unit());
        function<void(int, int, int)> build = [&](int l, int r, int i
        ) {
            if (l == r) return void(t[i] = create(l));
            int m = (l + r) / 2;
            build(l, m, i * 2);
            build(m + 1, r, i * 2 + 1);
            pull(i);
        };
        build(0, n - 1, 1);
    }
    void pull(int i) {
        t[i] = Monoid::op(t[i * 2], t[i * 2 + 1]);
    }
    void modify(int l, int r, int i, int x, const T &v) {
        if (x < l || r < x) return;
        if (l == r) return void(t[i] = v);
        int m = (l + r) / 2;
        modify(l, m, i * 2, x, v);
        modify(m + 1, r, i * 2 + 1, x, v);
        pull(i);
    }
    void modify(int x, const T &v) {
        modify(0, n - 1, 1, x, v);
    }
    template<class U>
    void update(int l, int r, int i, int x, const U &v) {

```

```

        if(x<1||r<x) return;
        if(l==r) return void(t[i]=Monoid::op(t[i],v));
        int m=(l+r)/2;
        update(l,m,i*2,x,v);
        update(m+1,r,i*2+1,x,v);
        pull(i);
    }
    template<class U>
    void update(int x,const U &v){
        update(0,n-1,l,x,v);
    }
    T query(int l,int r,int i,int x,int y){
        if(y<1||r<x) return Monoid::unit();
        if(x<=l&&r<=y) return t[i];
        int m=(l+r)/2;
        return Monoid::op(query(l,m,i*2,x,y),query(m+1,r,i*2+1,x,y));
    }
    T query(int x,int y){
        return query(0,n-1,l,x,y);
    }
    template<class F>
    int findfirst(int l,int r,int i,int x,int y,const F &f){
        if(y<1||r<x||!f(t[i])) return n;
        if(l==r) return l;
        int m=(l+r)/2;
        int res=findfirst(l,m,i*2,x,y,f);
        if(res==n) res=findfirst(m+1,r,i*2+1,x,y,f);
        return res;
    }
    template<class F>
    int findfirst(int x,int y,const F &f){
        return findfirst(0,n-1,l,x,y,f);
    }
    template<class F>
    int findlast(int l,int r,int i,int x,int y,const F &f){
        if(y<1||r<x||!f(t[i])) return -1;
        if(l==r) return l;
        int m=(l+r)/2;
        int res=findlast(m+1,r,i*2+1,x,y,f);
        if(res==-1) res=findlast(l,m,i*2,x,y,f);
        return res;
    }
    template<class F>
    int findlast(int x,int y,const F &f){
        return findlast(0,n-1,l,x,y,f);
    }
};

```

### LazySegmentTree.hpp

**Description:** Segment Tree with Lazy Propagation

91ab0c, 103 lines

```

template<class MonoidAction>
struct LazySegmentTree{
    using InfoMonoid = typename MonoidAction::InfoMonoid;
    using TagMonoid = typename MonoidAction::TagMonoid;
    using Info = typename MonoidAction::Info;
    using Tag = typename MonoidAction::Tag;
    int n;
    vector<Info> t;
    vector<Tag> lz;
    LazySegmentTree(){}
    LazySegmentTree(int n,function<Info(int)> create){init(n,create);}
    LazySegmentTree(int n,Info v=InfoMonoid::unit()){init(n,[&](int){return v;});}
    template<class T>
    LazySegmentTree(const vector<T> &a){init((int)a.size(),[&](int i){return Info(a[i]);});}
};

```

```

void init(int _n,function<Info(int)> create){
    n=_n;
    int m=4<<(31-__builtin_clz(n));
    t.assign(m,InfoMonoid::unit());
    lz.assign(m,TagMonoid::unit());
    function<void(int,int,int)> build=[&](int l,int r,int i){
        if(l==r) return void(t[i]=create(l));
        int m=(l+r)/2;
        build(l,m,i*2);
        build(m+1,r,i*2+1);
        pull(i);
    };
    build(0,n-1,1);
}
void pull(int i){
    t[i]=InfoMonoid::op(t[i*2],t[i*2+1]);
}
void apply(int i,const Tag &v){
    t[i]=MonoidAction::op(t[i],v);
    lz[i]=TagMonoid::op(lz[i],v);
}
void push(int i){
    apply(i*2,lz[i]);
    apply(i*2+1,lz[i]);
    lz[i]=TagMonoid::unit();
}
void modify(int l,int r,int i,int x,const Info &v){
    if(x<1||r<x) return;
    if(l==r) return void(t[i]=v);
    int m=(l+r)/2;
    push(i);
    modify(l,m,i*2,x,v);
    modify(m+1,r,i*2+1,x,v);
    pull(i);
}
void modify(int x,const Info &v){
    modify(0,n-1,l,x,v);
}
void update(int l,int r,int i,int x,int y,const Tag &v){
    if(y<1||r<x) return;
    if(x<=l&&r<=y) return apply(i,v);
    int m=(l+r)/2;
    push(i);
    update(l,m,i*2,x,y,v);
    update(m+1,r,i*2+1,x,y,v);
    pull(i);
}
void update(int x,int y,const Tag &v){
    update(0,n-1,l,x,y,v);
}
Info query(int l,int r,int i,int x,int y){
    if(y<1||r<x) return InfoMonoid::unit();
    if(x<=l&&r<=y) return t[i];
    int m=(l+r)/2;
    push(i);
    return InfoMonoid::op(query(l,m,i*2,x,y),query(m+1,r,i*2+1,x,y));
}
Info query(int x,int y){
    return query(0,n-1,l,x,y);
}
template<class F>
int findfirst(int l,int r,int i,int x,int y,const F &f){
    if(y<1||r<x||!f(t[i])) return n;
    if(l==r) return l;
    int m=(l+r)/2;
    push(i);
    int res=findfirst(l,m,i*2,x,y,f);
    int res=findfirst(m+1,r,i*2+1,x,y,f);
    return res;
}

```

```

        if(res==n) res=findfirst(m+1,r,i*2+1,x,y,f);
        return res;
    }
    template<class F>
    int findfirst(int x,int y,const F &f){
        return findfirst(0,n-1,l,x,y,f);
    }
    template<class F>
    int findlast(int l,int r,int i,int x,int y,const F &f){
        if(y<1||r<x||!f(t[i])) return -1;
        if(l==r) return l;
        int m=(l+r)/2;
        push(i);
        int res=findlast(m+1,r,i*2+1,x,y,f);
        if(res==-1) res=findlast(l,m,i*2,x,y,f);
        return res;
    }
    template<class F>
    int findlast(int x,int y,const F &f){
        return findlast(0,n-1,l,x,y,f);
    }
};

```

### DynamicSegmentTree.hpp

**Description:** Dynamic Segment Tree

e84eeb, 106 lines

```

template<class MonoidAction>
struct DynamicSegmentTree{
    using InfoMonoid = typename MonoidAction::InfoMonoid;
    using TagMonoid = typename MonoidAction::TagMonoid;
    using Info = typename MonoidAction::Info;
    using Tag = typename MonoidAction::Tag;
    struct Node;
    using Ptr = Node*;
    struct Node{
        Info val;
        Tag lz;
        Ptr l,r;
        Node(Info v):val(v),lz(TagMonoid::unit()),l(nullptr),r(nullptr){}
        Node(Info v,Tag t):val(v),lz(t),l(nullptr),r(nullptr){}
    };
    ll lb,ub;
    Ptr rt;
    function<Info(ll,ll)> create;
    DynamicSegmentTree() {init(0,0);}
    DynamicSegmentTree(ll n) {init(0,n-1);}
    DynamicSegmentTree(ll lb,ll ub) {init(lb,ub);}
    DynamicSegmentTree(ll lb,ll ub,function<Info(ll,ll)> create) {init(lb,ub,create);}
    void init(ll _lb,ll _ub,function<Info(ll,ll)> _create=[&](ll l,ll r){return InfoMonoid::unit();}){
        lb=_lb,ub=_ub;
        create=_create;
        rt=new Node(create(lb,ub));
    }
    Info val(Ptr t){
        return t?t->val:InfoMonoid::unit();
    }
    void pull(Ptr &t){
        t->val=InfoMonoid::op(val(t->l),val(t->r));
    }
    void apply(Ptr &t,const Tag &v,ll l,ll r){
        if(!t)t=new Node(create(l,r));
        t->val=MonoidAction::op(t->val,v);
        t->lz=TagMonoid::op(t->lz,v);
    }
    void push(Ptr &t,ll l,ll m,ll r){
        apply(t->l,t->lz,l,m);
    }
};

```

```

        apply(t->r,t->lz,m+1,r);
        t->lz=TagMonoid::unit();
    }
    void modify(ll l,ll r,Ptr &t,ll x,const Info &v){
        if(x<l||r<x)return;
        if(l==r)return void(t->val=v);
        ll m=l+(r-l)/2;
        push(t,l,m,r);
        modify(l,m,t->l,x,v);
        modify(m+1,r,t->r,x,v);
        pull(t);
    }
    void modify(ll x,const Info &v){
        modify(lb,ub,rt,x,v);
    }
    void update(ll l,ll r,Ptr &t,ll x,ll y,const Tag &v){
        if(y<l||r<x)return;
        if(x<=l&&r<=y)return apply(t,v,l,r);
        ll m=l+(r-l)/2;
        push(t,l,m,r);
        update(l,m,t->l,x,y,v);
        update(m+1,r,t->r,x,y,v);
        pull(t);
    }
    void update(ll x,ll y,const Tag &v){
        update(lb,ub,rt,x,y,v);
    }
    Info query(ll l,ll r,Ptr &t,ll x,ll y){
        if(y<l||r<x)return InfoMonoid::unit();
        if(x<=l&&r<=y)return t->val;
        ll m=l+(r-l)/2;
        push(t,l,m,r);
        return InfoMonoid::op(query(l,m,t->l,x,y),query(m+1,r,t->r,x,y));
    }
    Info query(ll x,ll y){
        return query(lb,ub,rt,x,y);
    }
    template<class F>
    ll findfirst(ll l,ll r,Ptr t,ll x,ll y,const F &f){
        if(y<l||r<x||!f(t->val))return -1;
        if(l==r)return l;
        ll m=l+(r-l)/2;
        push(t,l,m,r);
        ll res=findfirst(l,m,t->l,x,y,f);
        if(res==-1)res=findfirst(m+1,r,t->r,x,y,f);
        return res;
    }
    template<class F>
    ll findfirst(ll x,ll y,const F &f){
        return findfirst(lb,ub,rt,x,y,f);
    }
    template<class F>
    ll findlast(ll l,ll r,Ptr t,ll x,ll y,const F &f){
        if(y<l||r<x||!f(t->val))return -1;
        if(l==r)return l;
        ll m=l+(r-l)/2;
        push(t,l,m,r);
        ll res=findlast(m+1,r,t->r,x,y,f);
        if(res==-1)res=findlast(l,m,t->l,x,y,f);
        return res;
    }
    template<class F>
    ll findlast(ll x,ll y,const F &f){
        return findlast(lb,ub,rt,x,y,f);
    }
};

```

## DSU.hpp

Description: Disjoint Set Union.

0b3cb8, 26 lines

```

struct DSU{
    vector<int> p,sz;
    DSU(){}
    DSU(int n){init(n);}
    void init(int n){
        p.resize(n);
        iota(p.begin(),p.end(),0);
        sz.assign(n,1);
    }
    int find(int u){
        return p[u]==u?p[u]=find(p[u]);
    }
    bool same(int u,int v){
        return find(u)==find(v);
    }
    bool merge(int u,int v){
        u=find(u),v=find(v);
        if(u==v)return false;
        sz[u]+=sz[v];
        p[v]=u;
        return true;
    }
    int size(int u){
        return sz[find(u)];
    }
};

```

## BinaryTrie.hpp

Description: Binary Trie

ae5b7a, 66 lines

```

template<int BIT,class T = uint32_t,class S = int>
struct BinaryTrie{
    struct Node{
        array<int,2> ch;
        S cnt;
        Node():ch{-1,-1},cnt(0){}
    };
    vector<Node> t;
    BinaryTrie():t{Node()}{ }
    int new_node(){
        t.emplace_back(Node());
        return t.size()-1;
    }
    S size(){
        return t[0].cnt;
    }
    bool empty(){
        return size()==0;
    }
    S get_cnt(int i){
        return i!=-1?t[i].cnt:S(0);
    }
    void insert(T x,S k=1){
        int u=0;
        t[u].cnt+=k;
        for(int i=BIT-1;i>=0;i--){
            int v=x>>i&1;
            if(t[u].ch[v]==-1)t[u].ch[v]=new_node();
            u=t[u].ch[v];
            t[u].cnt+=k;
        }
    }
    void erase(T x,S k=1){
        int u=0;
        assert(t[u].cnt>=k);
        t[u].cnt-=k;
        for(int i=BIT-1;i>=0;i--){

```

```

            int v=x>>i&1;
            u=t[u].ch[v];
            assert(u!=-1&&t[u].cnt>=k);
            t[u].cnt-=k;
        }
    }
    T kth(S k,T x=0){
        assert(k<size());
        int u=0;
        T res=0;
        for(int i=BIT-1;i>=0;i--){
            int v=x>>i&1;
            if(k<get_cnt(t[u].ch[v])){
                u=t[u].ch[v];
            }else{
                res|=T(1)<<i;
                k-=get_cnt(t[u].ch[v]);
                u=t[u].ch[v^1];
            }
        }
        return res;
    }
    T min(T x){
        return kth(0,x);
    }
    T max(T x){
        return kth(size()-1,x);
    }
};

```

## LiChaoTree.hpp

Description: Li-Chao Tree (minimize).

4ab713, 52 lines

```

template<class T>
struct LiChaoTree{
    static const T INF=numeric_limits<T>::max()/2;
    struct Line{
        T m,c;
        Line(T _m,T _c):m(_m),c(_c){}
        inline T eval(T x)const{return m*x+c;}
    };
    vector<T> xs;
    vector<Line> t;
    LiChaoTree(){}
    LiChaoTree(const vector<T> &xs):xs(x){init(x);}
    LiChaoTree(int n):xs(n){
        vector<T> x(n);
        iota(x.begin(),x.end(),0);
        init(x);
    }
    void init(const vector<T> &x){
        xs=x;
        sort(xs.begin(),xs.end());
        xs.erase(unique(xs.begin(),xs.end()),xs.end());
        t.assign(4<<(31-__builtin_clz(xs.size())),Line(0,INF));
    }
    void insert(int l,int r,int i,Line v){
        int m=(l+r)/2;
        if(v.eval(xs[m])<t[i].eval(xs[m]))swap(t[i],v);
        if(v.eval(xs[l])<t[i].eval(xs[l]))insert(l,m,i*2,v);
        if(v.eval(xs[r])<t[i].eval(xs[r]))insert(m+1,r,i*2+1,v);
    }
    inline void insert(T m,T c){
        insert(0,(int)xs.size()-1,l,Line(m,c));
    }
    void insert_range(int l,int r,int i,T x,T y,Line v){
        if(y<xs[l]||xs[r]<x)return;
        if(x<=xs[l]&&xs[r]<=y)return insert(l,r,i,v);
    }
};

```

```

        int m=(l+r)/2;
        insert_range(l,m,i*2,x,y,v);
        insert_range(m+1,r,i*2+1,x,y,v);
    }
    inline void insert_range(T m,T c,T x,T y){
        insert_range(0,(int)xs.size()-1,l,x,y,Line(m,c));
    }
    T query(int l,int r,int i,T x){
        if(l==r)return t[i].eval(x);
        int m=(l+r)/2;
        if(x<=xs[m])return min(t[i].eval(x),query(l,m,i*2,x));
        return min(t[i].eval(x),query(m+1,r,i*2+1,x));
    }
    inline T query(T x){
        return query(0,(int)xs.size()-1,l,x);
    }
};

```

## DynamicLiChaoTree.hpp

**Description:** Dynamic Li-Chao Tree (minimize).

b8af36, 50 lines

```

template<class T>
struct DynamicLiChaoTree{
    static const T INF=numeric_limits<T>::max()/2;
    struct Line{
        T m,c;
        Line(T _m,T _c):m(_m),c(_c){}
        inline T eval(T x)const{return m*x+c;}
    };
    struct Node;
    using Ptr = Node*;
    struct Node{
        Line v;
        Ptr l,r;
        Node():v(0,INF),l(nullptr),r(nullptr){}
        Node(Line _v):v(_v),l(nullptr),r(nullptr){}
    };
    ll lb,ub;
    Ptr root;
    DynamicLiChaoTree(ll _lb,ll _ub):lb(_lb),ub(_ub),root(nullptr){}
    void insert(T l,T r,Ptr &t,Line v){
        if(!t)return void(t=new Node(v));
        T m=l+(r-l)/2;
        if(v.eval(m)<t->v.eval(m))swap(t->v,v);
        if(v.eval(l)<t->v.eval(l))insert(l,m,t->l,v);
        if(v.eval(r)<t->v.eval(r))insert(m+1,r,t->r,v);
    }
    inline void insert(T m,T c){
        insert(lb,ub,root,Line(m,c));
    }
    void insert_range(T l,T r,Ptr &t,T x,T y,Line v){
        if(y<l||r<x)return;
        if(!t)t=new Node();
        if(x<=l&&r<=y)return insert(l,r,t,v);
        T m=l+(r-l)/2;
        insert_range(l,m,t->l,x,y,v);
        insert_range(m+1,r,t->r,x,y,v);
    }
    inline void insert_range(T m,T c,T x,T y){
        insert_range(lb,ub,root,x,y,Line(m,c));
    }
    T query(T l,T r,Ptr t,T x){
        if(!t)return INF;
        T m=l+(r-l)/2;
        if(x<=m)return min(t->v.eval(x),query(l,m,t->l,x));
        return min(t->v.eval(x),query(m+1,r,t->r,x));
    }
    inline T query(T x){

```

```

        return query(lb,ub,root,x);
    }
};

SplayTreeBase.hpp
Description: Splay Tree. splay(u) will make node u be the root of the tree
in amortized O(log n) time.
cc90a9, 113 lines

template<class Node>
struct SplayTreeBase{
    using Ptr = Node*;
    bool is_root(Ptr t){
        return !(t->p)|| (t->p->l!=t&&t->p->r!=t);
    } // The parent of the root stores the path parant in link
        cut tree.
    int size(Ptr t){
        return t?t->size:0;
    }
    virtual void push(Ptr t){};
    virtual void pull(Ptr t){};
    int pos(Ptr t){
        if(t->p){
            if(t->p->l==t)return -1;
            if(t->p->r==t)return 1;
        }
        return 0;
    }
    void rotate(Ptr t){
        Ptr x=t->p,y=x->p;
        if(pos(t)==-1){
            if((x->l==t->r))t->r->p=x;
            t->r=x,x->p=t;
        }else{
            if((x->r==t->l))t->l->p=x;
            t->l=x,x->p=t;
        }
        pull(x),pull(t);
        if((t->p==y)){
            if(y->l==x)y->l=t;
            if(y->r==x)y->r=t;
        }
    }
    void splay(Ptr t){
        if(!t)return;
        push(t);
        while(!is_root(t)){
            Ptr x=t->p;
            if(is_root(x)){
                push(x),push(t);
                rotate(t);
            }else{
                Ptr y=x->p;
                push(y),push(x),push(t);
                if(pos(x)==pos(t))rotate(x),rotate(t);
                else rotate(t),rotate(t);
            }
        }
    }
    Ptr get_first(Ptr t){
        while(t->l)push(t),t=t->l;
        splay(t);
        return t;
    }
    Ptr get_last(Ptr t){
        while(t->r)push(t),t=t->r;
        splay(t);
        return t;
    }
    Ptr merge(Ptr l,Ptr r){

```

```

        splay(l),splay(r);
        if(!l)return r;
        if(!r)return l;
        l=get_last(l);
        l->r=r;
        r->p=l;
        pull(l);
        return l;
    }
    pair<Ptr,Ptr> split(Ptr t,int k){
        if(!t)return {nullptr,nullptr};
        if(k==0)return {nullptr,t};
        if(k==size(t))return {t,nullptr};
        push(t);
        if(k<=size(t->l)){
            auto x=split(t->l,k);
            t->l=x.second;
            t->p=nullptr;
            if(x.second)x.second->p=t;
            pull(t);
            return {x.first,t};
        }else{
            auto x=split(t->r,k-size(t->l)-1);
            t->r=x.first;
            t->p=nullptr;
            if(x.first)x.first->p=t;
            pull(t);
            return {t,x.second};
        }
    }
    void insert(Ptr &t,int k,Ptr v){
        splay(t);
        auto x=split(t,k);
        t=merge(merge(x.first,v),x.second);
    }
    void erase(Ptr &t,int k){
        splay(t);
        auto x=split(t,k);
        auto y=split(x.second,1);
        // delete y.first;
        t=merge(x.first,y.second);
    }
    template<class T>
    Ptr build(const vector<T> &v){
        if(v.empty())return nullptr;
        function<Ptr(int,int)> build=[&](int l,int r){
            if(l==r)return new Node(v[l]);
            int m=(l+r)/2;
            return merge(build(l,m),build(m+1,r));
        };
        return build(0,v.size()-1);
    }
};

```

## LazyReversibleBBST.hpp

**Description:** Lazy Reversible BBST Base.

904708, 81 lines

```

template<class Tree,class Node,class MonoidAction>
struct LazyReversibleBBST:Tree{
    using Tree::merge;
    using Tree::split;
    using typename Tree::Ptr;
    using InfoMonoid = typename MonoidAction::InfoMonoid;
    using TagMonoid = typename MonoidAction::TagMonoid;
    using Info = typename MonoidAction::Info;
    using Tag = typename MonoidAction::Tag;

    LazyReversibleBBST()=default;

```

```

Info sum(Ptr t){
    return t?>->sum:InfoMonoid::unit();
}
void pull(Ptr t){
    if(!t) return;
    push(t);
    t->size=1;
    t->sum=t->val;
    t->revsum=t->val;
    if(t->l){
        t->size+=t->l->size;
        t->sum=InfoMonoid::op(t->l->sum,t->sum);
        t->revsum=InfoMonoid::op(t->revsum,t->l->revsum);
    }
    if(t->r){
        t->size+=t->r->size;
        t->sum=InfoMonoid::op(t->sum,t->r->sum);
        t->revsum=InfoMonoid::op(t->r->revsum,t->revsum);
    }
}
void push(Ptr t){
    if(!t) return;
    if(t->rev){
        toggle(t->l);
        toggle(t->r);
        t->rev=false;
    }
    if(t->lz!=TagMonoid::unit()){
        propagate(t->l,t->lz);
        propagate(t->r,t->lz);
        t->lz=TagMonoid::unit();
    }
}
void toggle(Ptr t){
    if(!t) return;
    swap(t->l,t->r);
    swap(t->sum,t->revsum);
    t->rev^=true;
}
void propagate(Ptr t,const Tag &v){
    if(!t) return;
    t->val=MonoidAction::op(t->val,v);
    t->sum=MonoidAction::op(t->sum,v);
    t->revsum=MonoidAction::op(t->revsum,v);
    t->lz=TagMonoid::op(t->lz,v);
}
void apply(Ptr &t,int l,int r,const Tag &v){
    if(l>r) return;
    auto x=split(t,l);
    auto y=split(x.second,r-l+1);
    propagate(y.first,v);
    t=merge(x.first,merge(y.first,y.second));
}
Info query(Ptr &t,int l,int r){
    if(l>r) return InfoMonoid::unit();
    auto x=split(t,l);
    auto y=split(x.second,r-l+1);
    Info res=sum(y.first);
    t=merge(x.first,merge(y.first,y.second));
    return res;
}
void reverse(Ptr &t,int l,int r){
    if(l>r) return;
    auto x=split(t,l);
    auto y=split(x.second,r-l+1);
    toggle(y.first);
    t=merge(x.first,merge(y.first,y.second));
}
};

```

### LazyReversibleSplayTree.hpp

**Description:** Lazy Reversible Splay Tree.

"SplayTreeBase.hpp", "LazyReversibleBBST.hpp" b8455b, 23 lines

```

template<class MonoidAction>
struct LazyReversibleSplayTreeNode{
    using Ptr = LazyReversibleSplayTreeNode*;
    using InfoMonoid = typename MonoidAction::InfoMonoid;
    using TagMonoid = typename MonoidAction::TagMonoid;
    using Info = typename MonoidAction::Info;
    using Tag = typename MonoidAction::Tag;
    using value_type = Info;
    Ptr l,r,p;
    Info val,sum,revsum;
    Tag lz;
    int size;
    bool rev;
    LazyReversibleSplayTreeNode(const Info &_val=InfoMonoid::
        unit(),const Tag &_lz=TagMonoid::unit())
        :l(),r(),p(),val(_val),sum(_val),revsum(_val),lz(_lz),
        size(1),rev(false){}
};

template<class MonoidAction>
struct LazyReversibleSplayTree
: LazyReversibleBBST<SplayTreeBase<
    LazyReversibleSplayTreeNode<MonoidAction>>,
    LazyReversibleSplayTreeNode<MonoidAction>,MonoidAction>{
    using Node = LazyReversibleSplayTreeNode<MonoidAction>;
};

```

### LinkCutTreeBase.hpp

**Description:** Link Cut Tree Base.

**Usage:** evert(u): make u be the root of the tree.

link(u,v): attach u to v.

cut(u,v): remove edge between u and v.

get\_root(u): get the root of the tree containing u.

lca(u,v): get the lowest common ancestor of u and v.

fold(u,v): get the value of the path from u to v.

b432c3, 59 lines

```

template<class Splay>
struct LinkCutTreeBase:Splay{
    using Node = typename Splay::Node;
    using Ptr = Node*;
    using T = typename Node::value_type;
    Ptr expose(Ptr t){
        Ptr pc=nullptr; // preferred child
        for(Ptr cur=t;cur;cur=cur->p){
            this->splay(cur);
            cur->r=pc;
            this->pull(cur);
            pc=cur;
        }
        this->splay(t);
        return pc;
    }
    void evert(Ptr t){ // make t be the root of the tree
        expose(t);
        this->toggle(t);
        this->push(t);
    }
    void link(Ptr u,Ptr v){ // attach u to v
        evert(u);
        expose(v);
        u->p=v;
    }
    void cut(Ptr u,Ptr v){ // cut edge between u and v
        evert(u);
        expose(v);
        assert(u->p==v);
    }
};

```

```

v->l=u->p=nullptr;
this->pull(v);
}
Ptr get_root(Ptr t){
    expose(t);
    while(t->l)this->push(t),t=t->l;
    this->splay(t);
    return t;
}
Ptr lca(Ptr u,Ptr v){
    if(get_root(u)!=get_root(v)) return nullptr;
    expose(u);
    return expose(v);
}
void set_val(Ptr t,const T &val){
    this->evert(t);
    t->val=val;
    this->pull(t);
}
T get_val(Ptr t){
    this->evert(t);
    return t->val;
}
T fold(Ptr u,Ptr v){
    evert(u);
    expose(v);
    return v->sum;
}
};

```

### LazyLinkCutTree.hpp

**Description:** Lazy Link Cut Tree.

**Usage:** using Lct = LazyLinkCutTree<Action>;

using Ptr = Lct::Ptr;

using Node = Lct::Node;

vector<Ptr> ptr(n);

for(int i=0;i<n;i++)ptr[i]=new Node(val[i]);

auto link=[](int u,int v){

Lct::link(ptr[u],ptr[v]);

};

auto cut=[](int u,int v){

Lct::cut(ptr[u],ptr[v]);

};

auto update=[](int u,int v,Action::Tag val){

Lct::apply(ptr[u],ptr[v],val);

};

auto query=[](int u,int v){

return Lct::fold(ptr[u],ptr[v]);

};

"LazyReversibleSplayTree.hpp", "LinkCutTreeBase.hpp" ead3da, 12 lines

```

template<class MonoidAction>
struct LazyLinkCutTree:LinkCutTreeBase<LazyReversibleSplayTree<
    MonoidAction>>{
    using base = LinkCutTreeBase<LazyReversibleSplayTree<
        MonoidAction>>;
    using Ptr = typename base::Ptr;
    using Tag = typename MonoidAction::Tag;

    void apply(Ptr u,Ptr v,const Tag &val){
        this->evert(u);
        this->expose(v);
        this->propagate(v,val);
    }
};

```

## Number Theory (5)

ExtendedEuclid.hpp

**Description:** Extended Euclid algorithm for solving diophantine equation (ax + by = gcd(a, b)).  
**Time:**  $\mathcal{O}(\log \max\{a, b\})$

```

"../template/Header.hpp"
229e7c, 13 lines

pair<ll, ll> euclid(ll a, ll b) {
    ll x=1, y=0, x1=0, y1=1;
    while (b!=0) {
        ll q=a/b;
        x-=q*x1;
        y-=q*y1;
        a-=q*b;
        swap(x, x1);
        swap(y, y1);
        swap(a, b);
    }
    return {x, y};
}
```

5.1 Prime Numbers

LinearSieve.hpp

**Description:** Prime Number Generator in Linear Time  
**Time:**  $\mathcal{O}(N)$

```

"../template/Header.hpp"
194fb1, 15 lines

vi linear_sieve(int n) {
    vi prime, composite(n + 1);
    for(int i=2; i<=n; ++i) {
        if(!composite[i]) {
            prime.emplace_back(i);
        }
        for(int j=0; j<(int) prime.size() && i*prime[j]<=n; ++j) {
            composite[i * prime[j]] = true;
            if(i % prime[j] == 0) {
                break;
            }
        }
    }
    return prime;
}
```

FastEratosthenes.hpp

**Description:** Prime sieve for generating all primes smaller than LIM.  
**Time:** LIM=1e9 ≈ 1.5s

```

"../template/Header.hpp"
295b58, 33 lines

const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
    const int S = (int) round(sqrt(LIM)), R = LIM / 2;
    vi pr = {2}, sieve(S + 1);
    pr.reserve((int)(LIM/log(LIM) * 1.1));
    vector<pii> cp;
    for(int i=3; i<=S; i+=2) {
        if(!sieve[i]) {
            cp.emplace_back(i, i * i / 2);
            for(int j=i*i; j<=S; j+=2*i) {
                sieve[j] = 1;
            }
        }
    }
    for(int L=1; L<=R; L+=S) {
        array<bool, S> block{};
        for(auto &[p, idx]: cp) {
            for(int i=idx; i<S+L; idx=(i+p)) {
                block[i - L] = 1;
            }
        }
        for(int i=0; i<min(S, R-L); ++i) {
```

```

        if(!block[i]) {
            pr.emplace_back((L + i) * 2 + 1);
        }
    }
}

for(int i: pr) {
    isPrime[i] = 1;
}
return pr;
}
```

GolbatchConjecture.hpp

**Description:** Find two prime numbers which sum equals s  
**Time:**  $\mathcal{O}(N \log N)$

```

"FastEratosthenes.hpp"
88fb23, 18 lines

pair<int, int> golbatchConjecture(int s, vi pr = {}){
    if (s <= 2 || s % 2 != 0) {
        return make_pair(-1, -1);
    }
    if (pr.size() == 0) {
        pr = eratosthenes();
    }
    for (auto x : pr) {
        if (x > s / 2) {
            break;
        }
        int d = s - x;
        if (binary_search(pr.begin(), pr.end(), d)) {
            return make_pair(min(x, d), max(x, d));
        }
    }
    return make_pair(-1, -1);
}
```

Graph (6)

6.1 Matching

HopcroftKarp.hpp

**Description:** Fast bipartite matching algorithm.

**Time:**  $\mathcal{O}(E\sqrt{V})$

```

"../template/Header.hpp"
0bd56f, 52 lines

struct HopcroftKarp{
    int n,m;
    vi l,r,lv,ptr;
    vector<vi> adj;
    HopcroftKarp(){}
    HopcroftKarp(int _n,int _m){init(_n,_m);}
    void init(int _n,int _m){
        n=_n,m=_m;
        adj.assign(n+m,vi{});
    }
    void addEdge(int u,int v){
        adj[u].emplace_back(v+n);
    }
    void bfs(){
        lv=vi(n,-1);
        queue<int> q;
        for(int i=0;i<n;i++)if(l[i]==-1){
            lv[i]=0;
            q.emplace(i);
        }
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(int v:adj[u])if(r[v]!=-1&&lv[r[v]]==-1){
                lv[r[v]]=lv[u]+1;
```

```

                q.emplace(r[v]);
            }
        }
    }
    bool dfs(int u){
        for(int &i=ptr[u];i<sz(adj[u]);i++){
            int v=adj[u][i];
            if(r[v]==-1||(lv[r[v]]==lv[u]+1&&dfs(r[v]))){
                l[u]=v,r[v]=u;
                return true;
            }
        }
        return false;
    }
    int maxMatching(){
        int match=0,cnt=0;
        l=r=vi(n+m,-1);
        do{
            ptr=vi(n);
            bfs();
            cnt=0;
            for(int i=0;i<n;i++)if(l[i]==-1&&dfs(i))cnt++;
            match+=cnt;
        }while(cnt);
        return match;
    }
};
```

Kuhn.hpp

**Description:** Kuhn Algorithm to find maximum bipartite matching or find augmenting path in bipartite graph.

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

```

"../template/Header.hpp"
fc7d17, 15 lines

vi adj[1010], match(1010, -1);
bitset<1010> visited;
bool kuhn(int u) {
    if(visited[u]) {
        return false;
    }
    visited[u] = true;
    for(auto x: adj[u]) {
        if(match[x] == -1 || kuhn(match[x])) {
            match[x] = u;
            return true;
        }
    }
    return false;
}
```

6.2 Network Flow

Dinic.hpp

**Description:** Fast max-flow algorithm.

**Time:**  $\mathcal{O}(VE \log U)$  where  $U = \max |cap|$

```

"../template/Header.hpp"
7409c7, 68 lines

template<class T>
struct Dinic{
    struct Edge{
        int to;
        ll flow,cap;
        Edge(int _to,ll _cap):to(_to),flow(0),cap(_cap){}
        ll getcap(){
            return cap-flow;
        }
    };
    int n;
    ll U;
    vector<Edge> e;
    vector<vi> adj;
```



```

vi ptr, lvl;
Dinic() {}
Dinic(int _n) {
    init(_n);
}
void init(int _n) {
    n=_n, U=0;
    e.clear();
    adj.assign(n, {});
}
void addEdge(int u, int v, ll cap) {
    U=max(U, cap);
    adj[u].emplace_back(sz(e));
    e.emplace_back(v, cap);
    adj[v].emplace_back(sz(e));
    e.emplace_back(u, 0); // change 0 to cap for undirected
    flow
}
bool bfs(int s, int t, ll scale) {
    lvl.assign(n, 0);
    vi q(s);
    lvl[s]=1;
    for(int i=0; i<sz(q); i++) {
        int u=q[i];
        for(auto j:adj[u]) if(!lvl[e[j].to]&&e[j].getcap()>=
            scale) {
            q.emplace_back(e[j].to);
            lvl[e[j].to]=lvl[u]+1;
        }
    }
    return lvl[t];
}
ll dfs(int u, int t, ll f) {
    if(u==t||!f) return f;
    for(int &i=ptr[u]; i<sz(adj[u]); i++) {
        int j=adj[u][i];
        if(lvl[e[j].to]==lvl[u]+1) {
            if(ll p=dfs(e[j].to, t, min(f, e[j].getcap()))) {
                e[j].flow+=p;
                e[j^1].flow-=p;
                return p;
            }
        }
    }
    return 0;
}
ll flow(int s, int t) {
    ll flow=0;
    for(ll L=1ll<<(63-__builtin_clzll(U)); L>0; L>>=1) // L =
        1 may be faster but it's O(V^2 E)
        while(bfs(s, t, L)) {
            ptr.assign(n, 0);
            while(ll p=dfs(s, t, LINF)) flow+=p;
        };
    return flow;
}
};

```

### MinCostFlow.hpp

**Description:** minimum-cost flow algorithm.

**Time:**  $\mathcal{O}(FE \log V)$  where  $F$  is max flow.

../template/Header.hpp

Sea1d2, 83 lines

```

template<class F, class C>
struct MinCostFlow {
    struct Edge {
        int to;
        F flow, cap;
        C cost;
    };
};

```

```

Edge(int _to, F _cap, C _cost):to(_to), flow(0), cap(_cap),
    cost(_cost) {}
F getcap() {
    return cap-flow;
}
};
int n;
vector<Edge> e;
vector<vi> adj;
vector<C> pot, dist;
vi pre;
bool neg;
const F FINF=numeric_limits<F>::max()/2;
const C CINF=numeric_limits<C>::max()/2;
MinCostFlow() {}
MinCostFlow(int _n) {
    init(_n);
}
void init(int _n) {
    n=_n;
    e.clear();
    adj.assign(n, {});
    neg=false;
}
void addEdge(int u, int v, F cap, C cost) {
    adj[u].emplace_back(sz(e));
    e.emplace_back(v, cap, cost);
    adj[v].emplace_back(sz(e));
    e.emplace_back(u, 0, -cost);
    if(cost<0) neg=true;
}
bool dijkstra(int s, int t) {
    using P = pair<C, int>;
    dist.assign(n, CINF);
    pre.assign(n, -1);
    priority_queue<P, vector<P>, greater<P>> pq;
    dist[s]=0;
    pq.emplace(0, s);
    while(!pq.empty()) {
        auto [d, u]=pq.top();
        pq.pop();
        if(dist[u]<d) continue;
        for(int i:adj[u]) {
            int v=e[i].to;
            C ndist=d+pot[u]-pot[v]+e[i].cost;
            if(e[i].getcap()>0&&dist[v]>ndist) {
                pre[v]=i;
                dist[v]=ndist;
                pq.emplace(ndist, v);
            }
        }
    }
    return dist[t]<CINF;
}
pair<F, C> flow(int s, int t) {
    F flow=0;
    C cost=0;
    pot.assign(n, 0);
    if(neg) for(int t=0; t<n; t++) for(int i=0; i<sz(e); i++) if(e
        [i].getcap()>0) {
        int u=e[i^1].to, v=e[i].to;
        pot[v]=min(pot[v], pot[u]+e[i].cost);
    } // Bellman-Ford
    while(dijkstra(s, t)) {
        for(int i=0; i<n; i++) pot[i]+=dist[i];
        F aug=FINF;
        for(int u=t; u!=s; u=e[pre[u]^1].to) {
            aug=min(aug, e[pre[u]].getcap());
        } // find bottleneck
    }
}

```

```

for(int u=t; u!=s; u=e[pre[u]^1].to) {
    e[pre[u]].flow+=aug;
    e[pre[u]^1].flow-=aug;
} // push flow
flow+=aug;
cost+=aug*pot[t];
}
return {flow, cost};
}
};

```

## Polynomials (7)

### FormalPowerSeries.hpp

**Description:** basic operations of formal power series

"NTT.hpp"

416433, 136 lines

```

template<class mint>
struct FormalPowerSeries:vector<mint> {
    using vector<mint>::vector;
    using FPS = FormalPowerSeries;

    FPS &operator+=(const FPS &rhs) {
        if(rhs.size()>this->size()) this->resize(rhs.size());
        for(int i=0; i<rhs.size(); i++) (*this)[i]+=rhs[i];
        return *this;
    }
    FPS &operator+=(const mint &rhs) {
        if(this->empty()) this->resize(1);
        (*this)[0]+=rhs;
        return *this;
    }
    FPS &operator-=(const FPS &rhs) {
        if(rhs.size()>this->size()) this->resize(rhs.size());
        for(int i=0; i<rhs.size(); i++) (*this)[i]-=rhs[i];
        return *this;
    }
    FPS &operator-=(const mint &rhs) {
        if(this->empty()) this->resize(1);
        (*this)[0]-=rhs;
        return *this;
    }
    FPS &operator*=(const FPS &rhs) {
        auto res=NTT<mint>() (*this, rhs);
        return *this=FPS(res.begin(), res.end());
    }
    FPS &operator*=(const mint &rhs) {
        for(auto &a: *this) a*=rhs;
        return *this;
    }
    friend FPS operator+(FPS lhs, const FPS &rhs) {return lhs+=
        rhs;}
    friend FPS operator+(FPS lhs, const mint &rhs) {return lhs+=
        rhs;}
    friend FPS operator+(const mint &lhs, FPS &rhs) {return rhs+=
        lhs;}
    friend FPS operator-(FPS lhs, const FPS &rhs) {return lhs-=
        rhs;}
    friend FPS operator-(FPS lhs, const mint &rhs) {return lhs-=
        rhs;}
    friend FPS operator-(const mint &lhs, FPS rhs) {return -(rhs-
        lhs);}
    friend FPS operator*(FPS lhs, const FPS &rhs) {return lhs*=
        rhs;}
    friend FPS operator*(FPS lhs, const mint &rhs) {return lhs*=
        rhs;}
    friend FPS operator*(const mint &lhs, FPS rhs) {return rhs*=
        lhs;}
}

```



```

FPS operator-() {return (*this)*-1;}

FPS rev() {
    FPS res(*this);
    reverse(res.begin(), res.end());
    return res;
}

FPS pre(int sz) {
    FPS res(this->begin(), this->begin() + min((int) this->size(), sz));
    if (res.size() < sz) res.resize(sz);
    return res;
}

FPS shrink() {
    FPS res(*this);
    while (!res.empty() && res.back() == mint{}) res.pop_back();
    return res;
}

FPS operator>>(int sz) {
    if (this->size() <= sz) return {};
    FPS res(*this);
    res.erase(res.begin(), res.begin() + sz);
    return res;
}

FPS operator<<(int sz) {
    FPS res(*this);
    res.insert(res.begin(), sz, mint{});
    return res;
}

FPS diff() {
    const int n = this->size();
    FPS res(max(0, n-1));
    for (int i=1; i<n; i++) res[i-1] = (*this)[i] * mint(i);
    return res;
}

FPS integral() {
    const int n = this->size();
    FPS res(n+1);
    res[0] = 0;
    if (n > 0) res[1] = 1;
    ll mod = mint::get_mod();
    for (int i=2; i<=n; i++) res[i] = (-res[mod%i]) * (mod/i);
    for (int i=0; i<n; i++) res[i+1] = (*this)[i];
    return res;
}

mint eval(const mint &x) {
    mint res=0, w=1;
    for (auto &a: *this) res += a*w, w*=x;
    return res;
}

FPS inv(int deg=-1) {
    assert(!this->empty() && (*this)[0] != mint(0));
    if (deg == -1) deg = this->size();
    FPS res(mint(1) / (*this)[0]);
    for (int i=2; i>>1 < deg; i<=1) {
        res = (res * (mint(2) - res * pre(i))).pre(i);
    }
    return res.pre(deg);
}

FPS log(int deg=-1) {
    assert(!this->empty() && (*this)[0] == mint(1));
    if (deg == -1) deg = this->size();
    return (pre(deg).diff() * inv(deg)).pre(deg-1).integral();
}

FPS exp(int deg=-1) {
    assert(this->empty() || (*this)[0] == mint(0));

```

```

        if (deg == -1) deg = this->size();
        FPS res(mint(1));
        for (int i=2; i>>1 < deg; i<=1) {
            res = (res * (pre(i) - res.log(i) + mint(1))).pre(i);
        }
        return res.pre(deg);
    }

    FPS pow(ll k, int deg=-1) {
        const int n = this->size();
        if (deg == -1) deg = n;
        if (k == 0) {
            FPS res(deg);
            if (deg) res[0] = mint(1);
            return res;
        }
        for (int i=0; i<n; i++) {
            if (__int128_t(i) * k >= deg) return FPS(deg, mint(0));
            if ((*this)[i] == mint(0)) continue;
            mint rev = mint(1) / (*this)[i];
            FPS res = (((*this * rev) >> i).log(deg) * k).exp(deg);
            res = ((res * binpow((*this)[i], k)) << (i * k)).pre(deg);
            return res;
        }
        return FPS(deg, mint(0));
    }
};
using FPS = FormalPowerSeries<mint>;

```

### FFT.hpp

**Description:** Fast Fourier transform

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

"/template/Header.hpp" 5d476b, 73 lines

```

template<class T=ll, int mod=0>
struct FFT {
    using vt = vector<T>;
    using cd = complex<db>;
    using vc = vector<cd>;

    static const bool INT=true;

    static void fft(vc &a) {
        int n=a.size(), L=31-__builtin_clz(n);
        vc rt(n);
        rt[1]=1;
        for (int k=2; k<n; k*=2) {
            cd z=polar(db(1), PI/k);
            for (int i=k; i<2*k; i++) rt[i] = i&1?rt[i/2]*z:rt[i/2];
        }
        vi rev(n);
        for (int i=1; i<n; i++) rev[i] = (rev[i/2] | (i&1)<<L)/2;
        for (int i=1; 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++) {
            cd z=rt[j+k]*a[i+j+k];
            a[i+j+k] = a[i+j]-z;
            a[i+j] += z;
        }
    }

    template<class U>
    static db norm(const U &x) {
        return INT?round(x):x;
    }

    static vt conv(const vt &a, const vt &b) {
        if (a.empty() || b.empty()) return {};
        vt res(a.size()+b.size()-1);
        int L=32-__builtin_clz(res.size()), n=1<<L;
        vc in(n), out(n);
        copy(a.begin(), a.end(), in.begin());
        for (int i=0; i<b.size(); i++) in[i].imag(b[i]);
    }
};

```

```

fft(in);
for (auto &x: in) x*=x;
for (int i=0; i<n; i++) out[i] = in[-i&(n-1)] - conj(in[i]);
fft(out);
for (int i=0; i<res.size(); i++) res[i] = norm(imag(out[i]) / (4*n));
return res;
}

static vl convMod(const vl &a, const vl &b) {
    assert(mod>0);
    if (a.empty() || b.empty()) return {};
    vl res(a.size()+b.size()-1);
    int L=32-__builtin_clz(res.size()), n=1<<L;
    ll cut=int(sqrt(mod));
    vc in1(n), in2(n), out1(n), out2(n);
    for (int i=0; i<a.size(); i++) in1[i] = cd(ll(a[i])/cut, ll(a[i])%cut); // a1 + i * a2
    for (int i=0; i<b.size(); i++) in2[i] = cd(ll(b[i])/cut, ll(b[i])%cut); // b1 + i * b2
    fft(in1), fft(in2);
    for (int i=0; i<n; i++) {
        int j=-i&(n-1);
        out1[j] = (in1[i] + conj(in1[j])) * in2[i] / (2.1*n); // f1 * (g1 + i * g2) = f1 * g1 + i f1 * g2
        out2[j] = (in1[i] - conj(in1[j])) * in2[i] / cd(0.1, 2.1*n); // f2 * (g1 + i * g2) = f2 * g1 + i f2 * g2
    }
    fft(out1), fft(out2);
    for (int i=0; i<res.size(); i++) {
        ll x=round(real(out1[i])), y=round(imag(out1[i])) + round(real(out2[i])), z=round(imag(out2[i]));
        res[i] = ((x%mod*cut+y)%mod*cut+z)%mod; // a1 * b1 * cut^2 + (a1 * b2 + a2 * b1) * cut + a2 * b2
    }
    return res;
}

vt operator()(const vt &a, const vt &b) {
    return mod>0?conv(a, b):convMod(a, b);
}

};
template<>
struct FFT<db> {
    static const bool INT=false;
};

```

### NTT.hpp

**Description:** Number theoretic transform

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

"/template/Header.hpp", "/modular-arithmetic/BinPow.hpp",

"/modular-arithmetic/MontgomeryModInt.hpp" 2b2392, 39 lines

```

template<class mint=mint>
struct NTT {
    using vm = vector<mint>;

    static constexpr mint root=mint::get_root();
    static_assert(root!=0);

    static void ntt(vm &a) {
        int n=a.size(), L=31-__builtin_clz(n);
        vm rt(n);
        rt[1]=1;
        for (int k=2, s=2; k<n; k*=2, s++) {
            mint z[]={1, binpow(root, MOD>>s)};
            for (int i=k; i<2*k; i++) rt[i] = rt[i/2]*z[i&1];
        }
        vi rev(n);
        for (int i=1; i<n; i++) rev[i] = (rev[i/2] | (i&1)<<L)/2;
        for (int i=1; 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++){
        mint z=rt[j+k]*a[i+j+k];
        a[i+j+k]=a[i+j]-z;
        a[i+j]+=z;
    }
}
static vm conv(const vm &a,const vm &b){
    if(a.empty()||b.empty())return {};
    int s=a.size()+b.size()-1,n=1<<(32-__builtin_clz(s));
    mint inv=mint(n).inv();
    vm in1(a),in2(b),out(n);
    in1.resize(n),in2.resize(n);
    ntt(in1),ntt(in2);
    for(int i=0;i<n;i++)out[-i&(n-1)]=in1[i]*in2[i]*inv;
    ntt(out);
    return vm(out.begin(),out.begin()+s);
}
vm operator()(const vm &a,const vm &b){
    return conv(a,b);
}
};

```

## 7.1 Various

### GaussianElimination.hpp

**Description:** Gaussian Elimination

"../template/Header.hpp" e89ecb, 34 lines

```

struct Gauss {
    int n, sz;
    vector<ll> basis;
    Gauss(int n = 0) {
        init(n);
    }
    void init(int _n) {
        n = _n, sz = 0;
        basis.assign(n, 0);
    }
    void insert(ll x) {
        for (int i = n - 1; i >= 0; i--)
            if (x >> i & 1) {
                if (!basis[i]) {
                    basis[i] = x;
                    sz++;
                    return;
                }
                x ^= basis[i];
            }
    }
    ll getmax(ll k = 0) {
        ll tot = 1ll << sz, res = 0;
        for (int i = n - 1; i >= 0; i--)
            if (basis[i]) {
                tot >>= 1;
                if ((k >= tot && res >> i & 1) || (k < tot && res >> i
                    & 1 ^ 1))
                    res ^= basis[i];
                if (k >= tot)
                    k -= tot;
            }
        return res;
    }
};

```

### BinaryTrie.hpp

**Description:** Binary Trie

"../template/Header.hpp" 525bf4, 59 lines

```

using node_t = array<int, 2>;
template<size_t S>

```

```

struct binary_trie {
    vector<node_t> t = {node_t{}};
    vector<int> cnt = {0};
    int cnt_nodes = 0;
    void insert(int v) {
        int cur = 0;
        cnt[0]++;
        for(int i=S-1; i>=0; --i) {
            int b = (v & (1 << i)) ? 1 : 0;
            if(!t[cur][b]) {
                t[cur][b] = ++cnt_nodes;
                t.emplace_back(node_t{});
                cnt.emplace_back(0);
            }
            cnt[t[cur][b]]++;
            cur = t[cur][b];
        }
    }
    void remove(int v) {
        int cur = 0;
        cnt[0]--;
        for(int i=S-1; i>=0; --i) {
            int b = (v & (1 << i)) ? 1 : 0;
            cnt[t[cur][b]]--;
            cur = t[cur][b];
        }
    }
    int get_min(int v) {
        int cur = 0, res = 0;
        for(int i=(int) S-1; i>=0; --i) {
            int b = (v & (1 << i)) ? 1 : 0;
            if(t[cur][b] && cnt[t[cur][b]]) {
                cur = t[cur][b];
            }
            else {
                res |= (1 << i);
                cur = t[cur][!b];
            }
        }
        return res;
    }
    int get_max(int v) {
        int cur = 0, res = 0;
        for(int i=(int) S-1; i>=0; --i) {
            int b = (v & (1 << i)) ? 1 : 0;
            if(t[cur][!b] && cnt[t[cur][!b]]) {
                res |= (1 << i);
                cur = t[cur][!b];
            }
            else {
                cur = t[cur][b];
            }
        }
        return res;
    }
};

```

# Competitive Programming Topics

## (A)

topics.txt

159 lines

Recursion  
Divide and conquer  
    Finding interesting points in  $N \log N$   
Algorithm analysis  
    Master theorem  
    Amortized time complexity  
Greedy algorithm  
    Scheduling  
    Max contiguous subvector sum  
    Invariants  
    Huffman encoding  
Graph theory  
    Dynamic graphs (extra book-keeping)  
    Breadth first search  
    Depth first search  
    \* Normal trees / DFS trees  
    Dijkstra's algorithm  
    MST: Prim's algorithm  
    Bellman-Ford  
    Konig's theorem and vertex cover  
    Min-cost max flow  
    Lovasz toggle  
    Matrix tree theorem  
    Maximal matching, general graphs  
    Hopcroft-Karp  
    Hall's marriage theorem  
    Graphical sequences  
    Floyd-Warshall  
    Euler cycles  
    Flow networks  
    \* Augmenting paths  
    \* Edmonds-Karp  
    Bipartite matching  
    Min. path cover  
    Topological sorting  
    Strongly connected components  
    2-SAT  
    Cut vertices, cut-edges and biconnected components  
    Edge coloring  
    \* Trees  
    Vertex coloring  
    \* Bipartite graphs ( $\Rightarrow$  trees)  
    \*  $3^n$  (special case of set cover)  
    Diameter and centroid  
    K'th shortest path  
    Shortest cycle  
Dynamic programming  
    Knapsack  
    Coin change  
    Longest common subsequence  
    Longest increasing subsequence  
    Number of paths in a dag  
    Shortest path in a dag  
    Dynprog over intervals  
    Dynprog over subsets  
    Dynprog over probabilities  
    Dynprog over trees  
     $3^n$  set cover  
    Divide and conquer  
    Knuth optimization  
    Convex hull optimizations  
    RMQ (sparse table a.k.a  $2^k$ -jumps)  
    Bitonic cycle

    Log partitioning (loop over most restricted)  
Combinatorics  
    Computation of binomial coefficients  
    Pigeon-hole principle  
    Inclusion/exclusion  
    Catalan number  
    Pick's theorem  
Number theory  
    Integer parts  
    Divisibility  
    Euclidean algorithm  
    Modular arithmetic  
    \* Modular multiplication  
    \* Modular inverses  
    \* Modular exponentiation by squaring  
    Chinese remainder theorem  
    Fermat's little theorem  
    Euler's theorem  
    Phi function  
    Frobenius number  
    Quadratic reciprocity  
    Pollard-Rho  
    Miller-Rabin  
    Hensel lifting  
    Vieta root jumping  
Game theory  
    Combinatorial games  
    Game trees  
    Mini-max  
    Nim  
    Games on graphs  
    Games on graphs with loops  
    Grundy numbers  
    Bipartite games without repetition  
    General games without repetition  
    Alpha-beta pruning  
Probability theory  
Optimization  
    Binary search  
    Ternary search  
    Unimodality and convex functions  
    Binary search on derivative  
Numerical methods  
    Numeric integration  
    Newton's method  
    Root-finding with binary/ternary search  
    Golden section search  
Matrices  
    Gaussian elimination  
    Exponentiation by squaring  
Sorting  
    Radix sort  
Geometry  
    Coordinates and vectors  
    \* Cross product  
    \* Scalar product  
    Convex hull  
    Polygon cut  
    Closest pair  
    Coordinate-compression  
    Quadtrees  
    KD-trees  
    All segment-segment intersection  
Sweeping  
    Discretization (convert to events and sweep)  
    Angle sweeping  
    Line sweeping  
    Discrete second derivatives  
Strings

    Longest common substring  
    Palindrome subsequences  
    Knuth-Morris-Pratt  
    Tries  
    Rolling polynomial hashes  
    Suffix array  
    Suffix tree  
    Aho-Corasick  
    Manacher's algorithm  
    Letter position lists  
Combinatorial search  
    Meet in the middle  
    Brute-force with pruning  
    Best-first (A\*)  
    Bidirectional search  
    Iterative deepening DFS / A\*  
Data structures  
    LCA ( $2^k$ -jumps in trees in general)  
    Pull/push-technique on trees  
    Heavy-light decomposition  
    Centroid decomposition  
    Lazy propagation  
    Self-balancing trees  
    Convex hull trick (wcipeg.com/wiki/Convex\_hull\_trick)  
    Monotone queues / monotone stacks / sliding queues  
    Sliding queue using 2 stacks  
    Persistent segment tree