



Chulalongkorn University

# Vaporised

Teetat T., Borworntat D., Pakapim E.

template from KACTL

2024-08-01

1

Template

2

Mathematics

3

Numerical

4

Data Structures

5

Number Theory

6

Graph

7

Polynomials

# Template (1)

template.cpp27 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.sh2 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)

FenwickTree.hppDescription: Fenwick / Binary Indexed Tree5d9372, 31 lines

```
template<class T>
struct Fenwick{
    int n;
    vector<T> t;
    Fenwick(int n=0){init(n);}
    void init(int _n){
        n=_n;
        t.assign(n+1,T{});
    }
    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;<31-__builtin_clz(n);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 Treed12984, 86 lines

```
template<class Monoid>
struct SegmentTree{
    using T = typename Monoid::value_type;
    int n;
    vector<T> t;
    SegmentTree(){}
    SegmentTree(int n,T v=Monoid::unit()){init(n,v);}
    template<class U>
    SegmentTree(const vector<U> &a){init(a);}
    void init(int n,T v=Monoid::unit()){init(vector<T>(n,v));}
    template<class U>
    void init(const vector<U> &a){
        n=sz(a);
        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]=a[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<l||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-1,1,x,v);
    }
    T query(int l,int r,int i,int x,int y){
        if(y<l||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-1,1,x,y);
    }
    template<class F>
    int findfirst(int l,int r,int i,int x,int y,const F &f){
        if(y<l||r<x||!f(t[i]))return -1;
        if(l==r)return l;
        int m=(l+r)/2;
        int res=findfirst(l,m,i*2,x,y,f);
    }
};
```

```

        if(res==-1)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<l||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

"/template/Header.hpp" 901d10, 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,Info v=InfoMonoid::unit()){init(n,v);
    }
    template<class T>
    LazySegmentTree(const vector<T> &a){init(a);}
    void init(int n,Info v=InfoMonoid::unit()){init(vector<Info>
        >(n,v));}
    template<class T>
    void init(const vector<T> &a){
        n=sz(a);
        t.assign(4<<31-__builtin_clz(n),InfoMonoid::unit());
        lz.assign(4<<31-__builtin_clz(n),TagMonoid::unit());
        function<void(int,int,int)> build=[&](int l,int r,int i
            ){
                if(l==r)return void(t[i]=a[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<l||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<l||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<l||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<l||r<x||!f(t[i]))return -1;
        if(l==r)return l;
        int m=(l+r)/2;
        push(i);
        int res=findfirst(l,m,i*2,x,y,f);
        if(res==-1)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<l||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);
    }
};

```

## 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  $\approx$  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,lv1;
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){
    lv1.assign(n,0);
    vi q{s};
    lv1[s]=1;
    for(int i=0;i<sz(q);i++){
        int u=q[i];
        for(auto j:adj[u]) if(!lv1[e[j].to]&&e[j].getcap()>=scale){
            q.emplace_back(e[j].to);
            lv1[e[j].to]=lv1[u]+1;
        }
    }
    return lv1[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(lv1[e[j].to]==lv1[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" 8eald2, 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", 2b2392, 39 lines

```

"../modular-arithmetic/MontgomeryModInt.hpp"
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);
        }
};

```