



Chulalongkorn University

998244353

Teetat T., Borworntat D., Pakapim E.

template from KACTL

2024-08-15

1 Template

2 Mathematics

3 Numerical

4 Group

5 Data Structures

6 Number Theory

7 Graph

8 Tree

9 Polynomials

10 Strings

11 Geometry

12 Dynamic Programming

13 Convolutions

14 Various

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 = 0xffffffffffffffff;
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  $Q_{2n} \equiv Q_n - \frac{F(Q_n)}{F'(Q_n)} \pmod{x^{2n}}$ 


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



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



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



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



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


Group (4)

4.1 Monoid

monoid/MonoidBase.hpp
Description: Monoid Base class. e75b74, 6 lines

template<class T, T (*combine)(T, T), T (*identity)()>
struct MonoidBase{
    using value_type = T;
    static constexpr T op(const T &x, const T &y){return combine(x, y);}
    static constexpr T unit(){return identity();}
};

4.2 Action

action/MonoidActionBase.hpp
Description: Monoid Action Base class. 425d83, 11 lines

template<class MInfo, class MTag, typename MInfo::value_type
(*combine)(typename MInfo::value_type, typename MTag::value_type)>
struct MonoidActionBase{
```

```
using InfoMonoid = MInfo;
using TagMonoid = MTag;
using Info = typename InfoMonoid::value_type;
using Tag = typename TagMonoid::value_type;
static constexpr Info op(const Info &a, const Tag &b){
    return combine(a, b);
}
};

action/DefaultAction.hpp
Description: Default Action class. e45000, 10 lines

template<class Monoid>
struct DefaultAction{
    using InfoMonoid = Monoid;
    using TagMonoid = Monoid;
    using Info = typename Monoid::value_type;
    using Tag = typename Monoid::value_type;
    static constexpr Info op(const Info &a, const Tag &b){
        return Monoid::op(a, b);
    }
};

Data Structures (5)

OrderedSet.hpp
Description: Ordered Set 1a7f5f, 14 lines

template/Header.hpp, <bits/extc++.h>
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
Description: 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<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);
    }
};
```

```
        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 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-1,1,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-1,1,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<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-1,1,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-1,1,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-1,1,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 n;
        if(l==r)return l;
        int m=(l+r)/2;
        push(i);
        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 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-1,1,x,y,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);
}
};

```

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||!t||!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.

0b3eb8, 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> &x):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()-1,1,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?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,const 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 (6)

```
ExtendedEuclid.hpp
Description: Extended Euclid algorithm for solving diophantine equation
(ax + by = gcd(a, b)).
Time: 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};
}
```

6.1 Prime Numbers

```
LinearSieve.hpp
Description: Prime Number Generator in Linear Time
Time: 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: O(N log N)
"FastEratosthenes.hpp" 88fb23, 18 lines
pair<int, int> goldbatchConjecture(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);
}
```

```
euclid.h
Description: Finds two integers x and y, such that ax + by = gcd(a,b). If
you just need gcd, use the built in __gcd instead. If a and b are coprime,
then x is the inverse of a (mod b).
33ba8f, 5 lines
ll euclid(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d;
}
```

```
CRT.hpp
Description: Chinese Remainder Theorem.
crt(a, m, b, n) computes x such that x ≡ a (mod m), x ≡ b (mod n). If
|a| < m and |b| < n, x will obey 0 ≤ x < lcm(m,n). Assumes mn < 2^62.
Time: log(n)
"euclid.h" 04d93a, 7 lines
ll crt(ll a, ll m, ll b, ll n) {
    if (n > m) swap(a, b), swap(m, n);
    ll x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x;
}
```

Graph (7)

7.1 Matching

```
HopcroftKarp.hpp
Description: Fast bipartite matching algorithm.
Time: O(E√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;
}

```

7.2 Network Flow

Dinic.hpp

Description: Fast max-flow algorithm.

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

"/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=1;L<=(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};
}
};

```

Tree (8)

HLD.hpp

Description: HLD

"../template/Header.hpp" cf6882, 45 lines

```

vector<vi> adj;
vector<int> sz, lvl, hv, hd, p, disc;
int t;

```

```

void dfs(int u, int parent) {
    sz[u] = 1;
    lvl[u] = lvl[parent] + 1;
    p[u] = parent;
    int c_hv=0, c_max=0;
    for(auto v: adj[u]) {
        if(v == parent) continue;
        dfs(v, u);
        sz[u] += sz[v];
        if(c_max < sz[v]) {
            c_hv = v;
            c_max = sz[v];
        }
    }
    hv[u] = c_hv;
}

```

```

void hld(int u, int parent) {
    if(hd[u] == 0) {
        hd[u] = u;
    }
    disc[u] = ++t;
    if(hv[u] != 0) {

```

```

        hd[hv[u]] = hd[u];
        hld(hv[u], u);
    }
    for(auto v: adj[u]) {
        if(v == parent || v == hv[u]) {
            continue;
        }
        hld(v, u);
    }
}

```

```

int lca(int u, int v) {
    while(hd[u] != hd[v]) {
        if(lvl[hd[u]] > lvl[hd[v]]) swap(u, v);
        v=p[hd[v]];
    }
    return lvl[u] < lvl[v] ? u: v;
}

```

CentroidDecom.hpp

Description: Centroid

"../template/Header.hpp" e46d44, 32 lines

```

vector<vi> adj;
vi sz;
vector<bool> used;

```

```

int find_size(int u, int p) {
    sz[u] = 1;
    for(auto v: adj[u]) {
        if(v == p || used[v]) continue;
        sz[u] += find_size(v, u);
    }
    return sz[u];
}

```

```

int find_cen(int u, int p, int t) {
    for(auto v: adj[u]) {
        if(v == p || used[v]) continue;
        if(sz[v] * 2 > t) find_cen(v, u, t);
    }
    return u;
}

```

```

void decom(int u) {
    u = find_cen(u, 0, find_size(u, 0));
    used[u] = true;
    for(auto v: adj[u]) {
        // dfs do something
    }
    for(auto v: adj[u]) {
        if(used[v]) continue;
        decom(v);
    }
}

```

Polynomials (9)

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);
    }
};

```

Strings (10)

Manacher.hpp

Description: Manacher's Algorithm. $pal[i]$:= the length of the longest palindrome centered at $i/2$.

```

"../template/Header.hpp" 53856e, 15 lines
template<class STR>
vector<int> manacher(const STR &s){
    int n=(int)s.size();
    if(n==0) return {};
    vector<int> pal(2*n-1);
    for(int p=0,l=-1,r=-1;p<2*n-1;p++){
        int i=(p+1)>>1,j=p>>1;
        int k=(i>=r?0:min(r-i,pal[2*(l+r)-p]));
        while(j+k+1<n&&i-k-1>=0&&s[j+k+1]==s[i-k-1])k++;
        pal[p]=k;
        if(j+k>r)l=i-k,r=j+k;
    }
    for(int i=0;i<2*n-1;i++)pal[i]=pal[i]<<1|(i&1^1);
    return pal;
}

```

SuffixArray.hpp

Description: Suffix Automaton.

```

"../data-structure/SparseTable.hpp", "../group/monoid/Min.hpp" b9cfb1, 39 lines
template<class STR>
struct SuffixArray{
    int n;
    vector<int> sa,isa,lcp;
    SparseTable<MinMonoid<int>> st;
    SuffixArray(){}
    SuffixArray(const STR &s){init(s);}
    void init(const STR &s){
        n=(int)s.size();
        sa=isa=lcp=vector<int>(n+1);
        sa[0]=n;
        iota(sa.begin()+1,sa.end(),0);
        sort(sa.begin()+1,sa.end(),[&](int i,int j){return s[i]<s[j];});
        for(int i=1;i<=n;i++){
            int x=sa[i-1],y=sa[i];
            isa[y]=i>1&&s[x]==s[y]?isa[x]:i;
        }
        for(int len=1;len<=n;len<=1){
            vector<int> ps(sa),pi(isa),pos(n+1);
            iota(pos.begin(),pos.end(),0);
            for(auto i:ps)if((i=len)>=0)sa[pos[isa[i]]++]=i;
            for(int i=1;i<=n;i++){
                int x=sa[i-1],y=sa[i];
                isa[y]-pi[x]==pi[y]&&pi[x+len]==pi[y+len]?isa[x]:i;
            }
        }
        for(int i=0,k=0;i<=n;i++){
            for(int j=sa[isa[i]-1];j+k<=n&&s[j+k]==s[i+k];k++);
            lcp[isa[i]]=k;
            if(k)--;
        }
        st.init(lcp);
    }
    int get_lcp(int i,int j){
        if(i==j) return n-i;
        auto [l,r]=minmax(isa[i],isa[j]);
        return st.query(l+1,r);
    }
};

```

ZAlgo.hpp

Description: Z Algorithm. $z[i]$:= the length of the longest common prefix between s and $s[i]$.

```

"../template/Header.hpp" b93726, 12 lines
template<class STR>
vector<int> z_algorithm(const STR &s){
    int n=(int)s.size();
    vector<int> z(n);
    z[0]=n;
    for(int i=1,l=0,r=1;i<=n;i++){
        if(i<r)z[i]=min(r-i,z[i-1]);
        while(i+z[i]<=n&&s[i+z[i]]==s[i+z[i]])z[i]++;
        if(i+z[i]>r)l=i,r=i+z[i];
    }
    return z;
}

```

Geometry (11)

11.1 Geometric primitives

Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

```

47ec0a, 28 lines
template<class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist()==1
    P perp() const { return P(-y, x); } // rotates +90 degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the origin
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));
    }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

```

lineDistance.h

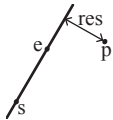
Description:

Returns the signed distance between point p and the line containing points a and b . Positive value on left side and negative on right as seen from a towards b . $a=b$ gives nan. P is supposed to be $\text{Point}<T>$ or $\text{Point3D}<T>$ where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D , call .dist on the result of the cross product.

```

"Point.h" f6bf6b, 4 lines
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (double)(b-a).cross(p-a)/(b-a).dist();
}

```



SegmentDistance.h

Description:

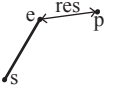
Returns the shortest distance between point p and the line segment from point s to e .

Usage: $\text{Point}<\text{double}> a, b(2,2), p(1,1);$
 $\text{bool onSegment} = \text{segDist}(a,b,p) < 1e-10;$

```

"Point.h" 5c88f4, 6 lines
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
}

```



SegmentIntersection.h

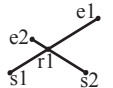
Description:

If a unique intersection point between the line segments going from s_1 to e_1 and from s_2 to e_2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is $\text{Point}<\text{ll}>$ and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

```

"Point.h", "OnSegment.h" 9d57f2, 13 lines
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)};
    set<P> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d);
    return {all(s)};
}

```



lineIntersection.h

Description:

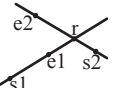
If a unique intersection point of the lines going through s_1, e_1 and s_2, e_2 exists $\{1, \text{point}\}$ is returned. If no intersection point exists $\{0, (0,0)\}$ is returned and if infinitely many exist $\{-1, (0,0)\}$ is returned. The wrong position will be returned if P is $\text{Point}<\text{ll}>$ and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.

Usage: $\text{auto res} = \text{lineInter}(s_1, e_1, s_2, e_2);$

```

"Point.h" a01f81, 8 lines
if (res.first == 1)
    cout << "intersection point at " << res.second << endl;
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {-(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d};
}

```



sideOf.h

Description: Returns where p is as seen from s towards e . $1/0/-1 \Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage: bool left = sideOf(p1,p2,q)==1;

```
"Point.h" 3af81c, 9 lines

template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
    auto a = (e-s).cross(p-s);
    double l = (e-s).dist()*eps;
    return (a > l) - (a < -l);
}
```

OnSegment.h

Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

```
"Point.h" c597e8, 3 lines

template<class P> bool onSegment(P s, P e, P p) {
    return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h

Description:

Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

```
"Point.h" 03a306, 6 lines

typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
    P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
    return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted

int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }

// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i

```
struct Angle {
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
    int half() const {
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
    Angle t180() const { return {-x, -y, t + half()}; }
    Angle t360() const { return {x, y, t + 1}; }
};

bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (1l)b.x) <
        make_tuple(b.t, b.half(), a.x * (1l)b.y);
}
```

// Given two points, this calculates the smallest angle between

```
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}

Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}

Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

11.2 Circles

CircleIntersection.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

```
"Point.h" 84d6d3, 11 lines

typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b - a;
    double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false;
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per};
    return true;
}
```

CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

```
"Point.h" b0153d, 13 lines

template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
    P d = c2 - c1;
    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) return {};
    vector<pair<P, P>> out;
    for (double sign : {-1, 1}) {
        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    if (h2 == 0) out.pop_back();
    return out;
}
```

CirclePolygonIntersection.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

Time: $\mathcal{O}(n)$

```
"../content/geometry/Point.h" a1ee63, 19 lines

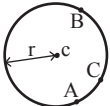
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
```

```
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s, v = p + d * t;
        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
    };
    auto sum = 0.0;
    rep(i,0,sz(ps))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

circumcircle.h

Description:

The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



```
"Point.h" 1caa3a, 9 lines

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
        abs((B-A).cross(C-A))/2;
}

P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

Time: expected $\mathcal{O}(n)$

```
"circumcircle.h" 09dd0a, 17 lines

pair<P, double> mec(vector<P> ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;
        rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
            r = (o - ps[i]).dist();
            rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
                o = ccCenter(ps[i], ps[j], ps[k]);
                r = (o - ps[i]).dist();
            }
        }
    }
    return {o, r};
}
```

11.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};

bool in = inPolygon(v, P{3, 3}, false);

Time: $\mathcal{O}(n)$

```
"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i,0,n) {
        P q = p[(i + 1) % n];
```

```
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
}
return cnt;
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

```
"Point.h" f12300, 6 lines

template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

Time: $\mathcal{O}(n)$

```
"Point.h" 9706dc, 9 lines

typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    }
    return res / A / 3;
}
```

PolygonCut.h

Description:

Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

```
"Point.h", "lineIntersection.h" f2b7d4, 13 lines

typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;
    rep(i,0,sz(poly)) {
        P cur = poly[i], prev = i ? poly[i-1] : poly.back();
        bool side = s.cross(e, cur) < 0;
        if (side != (s.cross(e, prev) < 0))
            res.push_back(lineInter(s, e, cur, prev).second);
        if (side)
            res.push_back(cur);
    }
    return res;
}
```

ConvexHull.h

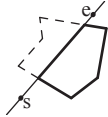
Description:

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

Time: $\mathcal{O}(n \log n)$

```
"Point.h" 310954, 13 lines

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
    if (sz(pts) <= 1) return pts;
    sort(all(pts));
    vector<P> h(sz(pts)+1);
    int s = 0, t = 0;
    for (int it = 2; it--; s = --t, reverse(all(pts)))
```



```
    for (P p : pts) {
        while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
        h[t++] = p;
    }
    return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

Time: $\mathcal{O}(n)$

```
"Point.h" c571b8, 12 lines

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i,0,j)
        for (; j = (j + 1) % n) {
            res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
                break;
        }
    return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

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

```
"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
    int a = 1, b = sz(l) - 1, r = !strict;
    if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
    if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
    if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <= -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
    }
    return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: $\bullet(-1, -1)$ if no collision, $\bullet(i, -1)$ if touching the corner i , $\bullet(i, i)$ if along side $(i, i + 1)$, $\bullet(i, j)$ if crossing sides $(i, i + 1)$ and $(j, j + 1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i, i + 1)$. The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.

Time: $\mathcal{O}(\log n)$

```
"Point.h" 7cf45b, 39 lines

#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
```

```
    }
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i,0,2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

11.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.

Time: $\mathcal{O}(n \log n)$

```
"Point.h" ac41a6, 17 lines

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
    assert(sz(v) > 1);
    set<P> S;
    sort(all(v), [](P a, P b) { return a.y < b.y; });
    pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
    int j = 0;
    for (P p : v) {
        P d(1 + (ll)sqrt(ret.first), 0);
        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
        for (; lo != hi; ++lo)
            ret = min(ret, {(lo - p).dist2(), {lo, p}});
        S.insert(p);
    }
    return ret.second;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

```
"Point.h" bac5b0, 63 lines

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = -INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0;
```



```
T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
}

Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
        x0 = min(x0, p.x); x1 = max(x1, p.x);
        y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
        // divide by taking half the array for each child (not
        // best performance with many duplicates in the middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()});
    }
};

struct KDTree {
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

    pair<T, P> search(Node *node, const P& p) {
        if (!node->first) {
            // uncomment if we should not find the point itself:
            // if (p == node->pt) return {INF, P()};
            return make_pair((p - node->pt).dist2(), node->pt);
        }

        Node *f = node->first, *s = node->second;
        T bfirst = f->distance(p), bsec = s->distance(p);
        if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

        // search closest side first, other side if needed
        auto best = search(f, p);
        if (bsec < best.first)
            best = min(best, search(s, p));
        return best;
    }

    // find nearest point to a point, and its squared distance
    // (requires an arbitrary operator< for Point)
    pair<T, P> nearest(const P& p) {
        return search(root, p);
    }
};
```

```
pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
        // uncomment if we should not find the point itself:
        // if (p == node->pt) return {INF, P()};
        return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
        best = min(best, search(s, p));
    return best;
}

// find nearest point to a point, and its squared distance
// (requires an arbitrary operator< for Point)
pair<T, P> nearest(const P& p) {
    return search(root, p);
}
};
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ...}, all counter-clockwise.

Time: $O(n \log n)$

```
"Point.h" eefdf5, 88 lines

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t l1l; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point

struct Quad {
    Q rot, o; P p = arb; bool mark;
    P& F() { return r()->p; }
    Q& r() { return rot->rot; }
```

```
Q prev() { return rot->o->rot; }
Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    l1l p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2;
    return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}

Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad(new Quad{new Quad{new Quad{0}}});
    H = r->o; r->r()->r() = r;
    rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
    r->p = orig; r->F() = dest;
    return r;
}

void splice(Q a, Q b) {
    swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}

Q connect(Q a, Q b) {
    Q q = makeEdge(a->F(), b->p);
    splice(q, a->next());
    splice(q->r(), b);
    return q;
}

pair<Q,Q> rec(const vector<P>& s) {
    if (sz(s) <= 3) {
        Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
        if (sz(s) == 2) return { a, a->r() };
        splice(a->r(), b);
        auto side = s[0].cross(s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
    }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
    Q A, B, ra, rb;
    int half = sz(s) / 2;
    tie(ra, A) = rec({all(s) - half});
    tie(B, rb) = rec({sz(s) - half + all(s)});
    while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
    Q base = connect(B->r(), A);
    if (A->p == ra->p) ra = base->r();
    if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
        Q t = e->dir; \
        splice(e, e->prev()); \
        splice(e->r(), e->r()->prev()); \
        e->o = H; H = e; e = t; \
    }
    for (;;) {
        DEL(LC, base->r(), o); DEL(RC, base, prev());
        if (!valid(LC) && !valid(RC)) break;
        if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
            base = connect(RC, base->r());
        else
            base = connect(base->r(), LC->r());
    }
    return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
    if (sz(pts) < 2) return {};
```

```
Q e = rec(pts).first;
vector<Q> q = {e};
int qi = 0;
while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
q.push_back(c->r()); c = c->next(); } while (c != e); }
ADD; pts.clear();
while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
return pts;
}
```

11.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

```
3058c3, 6 lines

template<class V, class L>
double signedPolyVolume(const V& p, const L& trilst) {
    double v = 0;
    for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
    return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

```
8058ae, 32 lines

template<class T> struct Point3D {
    typedef Point3D P;
    typedef const P& R;
    T x, y, z;
    explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
    bool operator<(R p) const {
        return tie(x, y, z) < tie(p.x, p.y, p.z); }
    bool operator==(R p) const {
        return tie(x, y, z) == tie(p.x, p.y, p.z); }
    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
    P operator*(T d) const { return P(x*d, y*d, z*d); }
    P operator/(T d) const { return P(x/d, y/d, z/d); }
    T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
    P cross(R p) const {
        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    }
    T dist2() const { return x*x + y*y + z*z; }
    double dist() const { return sqrt((double)dist2()); }
    //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
    double phi() const { return atan2(y, x); }
    //Zenith angle (latitude) to the z-axis in interval [0, pi]
    double theta() const { return atan2(sqrt(x*x+y*y),z); }
    P unit() const { return *this/(T)dist(); } //makes dist()==1
    //returns unit vector normal to *this and p
    P normal(P p) const { return cross(p).unit(); }
    //returns point rotated 'angle' radians ccw around axis
    P rotate(double angle, P axis) const {
        double s = sin(angle), c = cos(angle); P u = axis.unit();
        return u.dot(u)*(1-c) + (*this)*c - cross(u)*s;
    }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.

Time: $O(n^2)$

```
"Point3D.h" 5b45fc, 49 lines

typedef Point3D<double> P3;
```

```
struct PR {
    void ins(int x) { (a == -1 ? a : b) = x; }
    void rem(int x) { (a == x ? a : b) = -1; }
    int cnt() { return (a != -1) + (b != -1); }
    int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
    vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i]))
            q = q * -1;
        F f{q, i, j, k};
        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
        FS.push_back(f);
    };
    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
        mf(i, j, k, 6 - i - j - k);

    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
                E(b,c).rem(f.a);
                swap(FS[j--], FS.back());
                FS.pop_back();
            }
        }
        int nw = sz(FS);
        rep(j,0,nw) {
            F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
            C(a, b, c); C(a, c, b); C(b, c, a);
        }
        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
            A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
        return FS;
    };
};

double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz);
    return radius*2*asin(d/2);
}
```

Dynamic Programming (12)

```
DVC.hpp
Description: Optimize  $O(N^2K)$  to  $O(NK \log N)$ 
"../template/Header.hpp" aa5ddf, 19 lines

vector<vl> cst, dp;

ll cost(int l, int r) {
    return cst[l][r];
}

void divide(int l, int r, int opt_l, int opt_r, int c) {
    if(l > r) return;
    int mid = (l + r) / 2;
    pair<ll, int> best = make_pair(INF, -1);
    for(int k=opt_l; k<=min(mid, opt_r); ++k) {
        best = min(best, make_pair(dp[c - 1][k] + cost(k + 1, mid),
            k));
    }
    dp[c][mid] = best.first;
    divide(l, mid - 1, opt_l, best.second, c);
    divide(mid + 1, r, best.second, opt_r, c);
}

// for(int c=1; c<=K; ++c) divide(1, N, 1, N, c);

SlopeTrick.hpp
Description: Absolute Smth
"../template/Header.hpp" f62f9a, 36 lines

ll extending_value;

struct slope_trick {
    multiset<ll> ms_l, ms_r;
    ll min_y = 0ll, lz_l = 0ll, lz_r = 0ll;
    bool extending = false;
    void add_line(ll v) {
        if(extending) {
            lz_l -= extending_value;
            lz_r -= extending_value;
        }
        extending = true;
        if(ms_l.empty() && ms_r.empty()) {
            ms_l.emplace(v);
            ms_r.emplace(v);
        }
        else if(v <= *ms_l.rbegin() + lz_l) {
            min_y += (*ms_l.rbegin() + lz_l) - v;
            ms_r.emplace(*ms_l.rbegin() + lz_l - lz_r);
            ms_l.erase(--ms_l.end());
            ms_l.emplace(v - lz_l);
            ms_l.emplace(v - lz_l);
        }
        else if(v >= *ms_r.begin() + lz_r) {
            min_y += v - (*ms_r.begin() + lz_r);
            ms_l.emplace(*ms_r.begin() + lz_r - lz_l);
            ms_r.erase(ms_r.begin());
            ms_r.emplace(v - lz_r);
            ms_r.emplace(v - lz_r);
        }
        else {
            ms_l.emplace(v - lz_l);
            ms_r.emplace(v - lz_r);
        }
    }
};
```

Convolutions (13)

```
AndConvolution.hpp
Description: Bitwise AND Convolution. Superset Zeta Transform:  $A'[S] = \sum_{T \supseteq S} A[T]$ . Superset Mobius Transform:  $A[T] = \sum_{S \supseteq T} (-1)^{|S-T|} A'[S]$ .
Time:  $\mathcal{O}(N \log N)$ .
"../template/Header.hpp" 7916f8, 34 lines

template<class T>
void superset_zeta(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1;i<n;i<=<=1){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j^i]+=a[j];
            }
        }
    }
}

template<class T>
void superset_mobius(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=n;i>=1;i++){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j^i]-=a[j];
            }
        }
    }
}

template<class T>
vector<T> and_convolution(vector<T> a,vector<T> b){
    superset_zeta(a);
    superset_zeta(b);
    for(int i=0;i<(int)a.size();i++)a[i]*=b[i];
    superset_mobius(a);
    return a;
}

GCDConvolution.hpp
Description: GCD Convolution. Multiple Zeta Transform:  $A'[n] = \sum_{n|m} A[m]$ . Multiple Mobius Transform:  $A[n] = \sum_{n|m} \mu(m/n)A'[m]$ .
Time:  $\mathcal{O}(N \log \log N)$ .
"../template/Header.hpp" 7f6c2d, 34 lines

template<class T>
void multiple_zeta(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=(n-1)/p;i>=1;i--){
            is_prime[i*p]=false;
            a[i]+=a[i*p];
        }
    }
}

template<class T>
void multiple_mobius(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=1;i*p<n;i++){
            is_prime[i*p]=false;
            a[i]-=a[i*p];
        }
    }
}
```



```
    }
}

template<class T>
vector<T> gcd_convolution(vector<T> a,vector<T> b){
    multiple_zeta(a);
    multiple_zeta(b);
    for(int i=0;i<(int)a.size();i++) a[i]*=b[i];
    multiple_mobius(a);
    return a;
}
```

LCMConvolution.hpp
Description: LCM Convolution. Divisor Zeta Transform: $A'[n] = \sum_{d|n} A[d]$. Divisor Mobius Transform: $A[n] = \sum_{d|n} \mu(n/d)A'[d]$.
Time: $\mathcal{O}(N \log \log N)$.
"../template/Header.hpp" 41fe9d, 34 lines

```
template<class T>
void divisor_zeta(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=1;i*p<n;i++){
            is_prime[i*p]=false;
            a[i*p]+=a[i];
        }
    }

template<class T>
void divisor_mobius(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=(n-1)/p;i>=1;i--){
            is_prime[i*p]=false;
            a[i*p]-=a[i];
        }
    }

template<class T>
vector<T> lcm_convolution(vector<T> a,vector<T> b){
    divisor_zeta(a);
    divisor_zeta(b);
    for(int i=0;i<(int)a.size();i++) a[i]*=b[i];
    divisor_mobius(a);
    return a;
}
```

ORConvolution.hpp
Description: Bitwise OR Convolution. Subset Zeta Transform: $A'[S] = \sum_{T \subseteq S} A[T]$. Subset Mobius Transform: $A[T] = \sum_{S \subseteq T} (-1)^{|T-S|} A'[S]$.
Time: $\mathcal{O}(N \log N)$.
"../template/Header.hpp" c58b77, 34 lines

```
template<class T>
void subset_zeta(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&n));
    for(int i=1;i<n;i<=1){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j]+=a[j^i];
            }
        }
    }
}
```

```
    }
}

template<class T>
void subset_mobius(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&n));
    for(int i=n;i>=1;){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j]-=a[j^i];
            }
        }
    }
}

template<class T>
vector<T> or_convolution(vector<T> a,vector<T> b){
    subset_zeta(a);
    subset_zeta(b);
    for(int i=0;i<(int)a.size();i++) a[i]*=b[i];
    subset_mobius(a);
    return a;
}
```

XORConvolution.hpp
Description: Bitwise XOR Convolution. Fast Walsh-Hadamard Transform: $A'[S] = \sum_T (-1)^{|S \& T|} A[T]$.
Time: $\mathcal{O}(N \log N)$.
"../template/Header.hpp" 05848d, 29 lines

```
template<class T>
void fwht(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&n));
    for(int i=1;i<n;i<=1){
        for(int j=0;j<n;j++){
            if(j&i){
                T &u=a[j^i],&v=a[j];
                tie(u,v)=make_pair(u+v,u-v);
            }
        }
    }

template<class T>
vector<T> xor_convolution(vector<T> a,vector<T> b){
    int n=(int)a.size();
    fwht(a);
    fwht(b);
    for(int i=0;i<n;i++) a[i]*=b[i];
    fwht(a);
    T div=T(1)/T(n);
    if(div==T(0)){
        for(auto &x:a) x/=n;
    }else{
        for(auto &x:a) x*=div;
    }
    return a;
}
```

Various (14)

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;
}
};

```

14.1 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

14.1.1 Bit hacks

- $x \& -x$ is the least bit in x .
- `for (int x = m; x;) { --x &= m; ... }` loops over all subset masks of m (except m itself).
- $c = x \& -x$, $r = x + c$; $((r^x) >> 2) / c$ | r is the next number after x with the same number of bits set.
- `rep(b, 0, K) rep(i, 0, (1 << K))`
`if (i & 1 << b) D[i] += D[i^(1 << b)];`
 computes all sums of subsets.

14.1.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

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