Contents

cardinal.hpp

```cpp
#include <bits/stdc++.h>
using namespace std;
using INT=int;
//#define int long long
#define pb push_back
#define eb emplace_back
#define all(a) (a).begin(),(a).end()
template<class T>
using refT=reference_wrapper<T>;
template<class T>
using crefT=reference_wrapper<const T>;
auto &_=std::ignore;
using ll=long long;
template<class T>
using vec=vector<T>;
template<bool B,class T=void>
using enableif_t=typename enable_if<B,T>::type;
```

```
18
19  #define DEF_COULD(name,exp) \
20  template<class U> \
21  struct name{\
22      template<class T>\
23      constexpr static auto is(int i)->decltype(exp,true){return true;}\
24      template<class T>\
25      constexpr static bool is(...){return false;}\
26      static const bool value=is<U>(1);\
27  };
28  #define DEF_CAN(name,exp) DEF_COULD(can##name,exp)
29  #define ENABLE(T,name) enableif_t<can##name<T>::value>(1)
30  #define ENABLEN(T,name) enableif_t<!can##name<T>::value>(1)
31  #define FOR_TUPLE enableif_t<i!=tuple_size<T>::value>(1)
32  #define END_TUPLE enableif_t<i==tuple_size<T>::value>(1)
33
34  #define DEF_INF(name,exp)\
35  constexpr struct{\
36      template<class T>\
37      constexpr operator T()const {return numeric_limits<T>::exp();}\
38  } name;
39
40  DEF_CAN(Out,(cout<<*(T*)(0))) DEF_CAN(For,begin(*(T*)(0)))
41  DEF_INF(INF,max) DEF_INF(MINF,min)
42
43  template<size_t i,class T>
44  auto operator>>(istream& is,T &r)->decltype(END_TUPLE,is){
45      return is;
46  }
47  template<size_t i=0,class T>
48  auto operator>>(istream& is,T &r)->decltype(FOR_TUPLE,is){
49      is>>get<i>(r);
50      return operator>> <i+1>(is,r);
51  }
52  template<class T>
53  auto __format(ostream &os,const char *c,const T& cv)->decltype(ENABLE(T,Out),c+1){
54      os << cv;
55      while (*c!='}') c++;
56      return c+1;
57  }
58  template<size_t i,class T>
59  auto __format(ostream &os,const char *c,const T&
    ↪  cv)->decltype(ENABLEN(T,For),END_TUPLE,c+1){
60      return c;
61  }
62  template<size_t i=0,class T>
63  auto __format(ostream &os,const char *c,const T&
    ↪  cv)->decltype(ENABLEN(T,For),FOR_TUPLE,c+1){
64      while (*c!='{') os << *c++;
65      c=__format(os,c,get<i>(cv));
66      return __format<i+1>(os,c,cv);
67  }
68  template<class T>
69  auto __format(ostream &os,const char *c,const T&
    ↪  cv)->decltype(ENABLEN(T,Out),ENABLE(T,For),c+1){
70      const char *ct=c+1;
```

```
    if (cv.size()==0){
        while (*ct!='}') ct++;
        ct++;
        while (*ct!='}') ct++;
    }else{
        for (auto &i:cv){
            const char *cc=c+1;
            while (*cc!='{') os << *cc++;
            cc=__format(os,cc,i);
            while (*cc!='}') os << *cc++;
            ct=cc;
        }
    }
    return ct+1;
}
void _format(ostream &os,const char *c){
    while (*c!='{'&&*c!='\0') os<< *c++;
}
template<class T,class ...Args>
void _format(ostream &os,const char *c,const T &a,Args&& ...rest){
    while (*c!='{'&&*c!='\0') os<< *c++;
    if (*c=='{') c=__format(os,c,a);
    _format(os,c,forward<Args>(rest)...);
}
template<class ...Args>
string format(const char *c,Args&& ...rest){
    ostringstream os;
    _format(os,c,forward<Args>(rest)...);
    return os.str();
}
template<class ...Args>
ostream& print(const char *c,Args&& ...rest){return
 ↪  _format(cout,c,forward<Args>(rest)...),cout;}

#ifdef LOCAL
#define debug(...) cerr<<format(__VA_ARGS__)
#else
#define debug(...) cerr
#endif
template<class T,class ...Args>
struct Rtar{
    T& a;tuple<Args...> n;
    Rtar(T& a,tuple<Args...> n):a(a),n(n){}
};
template<class T,class ...Args>
Rtar<T,Args&...> rtar(T &a,Args&... rest){
    return Rtar<T,Args&...>(a,tie(rest...));
}
template<size_t i,class U,class ...Args,class T=tuple<Args&...>>
auto operator>>(istream& is,Rtar<U,Args&...> r)->decltype(END_TUPLE,is){
    return is>>r.a;
}
template<size_t i=0,class U,class ...Args,class T=tuple<Args&...>>
auto operator>>(istream& is,Rtar<U,Args&...> r)->decltype(FOR_TUPLE,is){
    r.a=typename decay<U>::type(get<i>(r.n));
    for (auto &w:r.a)
```

```
        operator>> <i+1>(is,Rtar<decltype(w),Args&...>(w,r.n));
    return is;
}
template<class T1,class T2>
bool cmin(T1 &a,const T2 b){return a>b?a=b,1:0;}
template<class T1,class T2>
bool cmax(T1 &a,const T2 b){return a<b?a=b,1:0;}
template<class T1,class T2,class ...T3>
bool cmin(T1 &a,const T2 b,const T3 ...rest){return cmin(a,b)|cmin(a,rest...);}
template<class T1,class T2,class ...T3>
bool cmax(T1 &a,const T2 b,const T3 ...rest){return cmax(a,b)|cmax(a,rest...);}
bool MULTIDATA=true;
```

dijkstra.hpp

```
template<class vT=int,class GT>
vector<vT> dijkstra(const GT &G,int i){
    using P=pair<vT,int>;
    const auto n=G.size();
    vector<vT> dis(n,(vT)INF);
    vector<bool> book(n,true);
    dis[i]=0;
    set<P> p;
    p.insert(make_pair(0,i));
    int now=i;
    while (p.size()){
        now=p.begin()->second;p.erase(p.begin());
        if (!book[now]) continue;
        book[now]=false;
        for (auto &e:G[now])
            if (book[e.v]&&dis[e.v]>dis[now]+e.w){
                p.erase(make_pair(dis[e.v],e.v));
                dis[e.v]=dis[now]+e.w;
                p.insert(make_pair(dis[e.v],e.v));
            }
    }
    return dis;
}
```

dinic.hpp

```
template<class vT,class T,class lT>
vT dinic_dfs(T &G,int s,int t,lT &level,vT maxf=INF){
    if (s==t||maxf==0) return maxf;
    vT flow=0;
    for (auto &e:G[s]){
        int v=e.v;
        if (level[v]==level[s]+1&&e.flow<e.cap){
            vT f=dinic_dfs(G,v,t,level,min(e.cap-e.flow,maxf));
            if (f>0){
                e.flow+=f;
                e.iedge->flow-=f;
                flow+=f;
                maxf-=f;
                if (maxf==0) break;
```

```
            }
        }
    }
    if (flow==0) level[s]=0;
    return flow;
}

template<class vT,class T>
vT dinic(T &G,int s,int t){
    const size_t n=G.size();
    vT ans=0;
    vector<int> level;
    while([&](){
        level=vector<int>(n,0);
        queue<int> q;
        level[s]=1;
        q.push(s);
        while(!q.empty()){
            int x=q.front();q.pop();
            for(auto &e:G[x]){
                int v=e.v;
                if(!level[v])
                    if(e.flow<e.cap){
                        level[v]=level[x]+1;
                        q.push(v);
                    }
            }
        }
        return level[t];
    }()){
        while(vT tmp=dinic_dfs<vT>(G,s,t,level)) ans+=tmp;
    }
    return ans;
}
```

EK.hpp

```
template<class vT,class GT>
vT EK(GT &G,int source,int sink){
    vector<typename GT::value_type::value_type*> path(G.size());
    vT totflow=0;
    while (true){
        vector<vT> flow(G.size());
        queue<int> q;
        flow[source]=INF;
        q.push(source);
        while (q.size()){
            int x=q.front();q.pop();
            for (auto &e:G[x])
                if (!flow[e.v]&&e.cap>e.flow){
                    path[e.v]=&e;
                    flow[e.v]=min(flow[e.u],e.cap-e.flow);
                    q.push(e.v);
                }
            if (flow[sink]) break;
```

```
        }
        if (!flow[sink]) break;
        for (auto i=path[sink];i!=path[source];i=path[i->u]){
            i->flow+=flow[sink];
            i->iedge->flow-=flow[sink];
        }
        totflow+=flow[sink];
    }
    return totflow;
}
```

fft.hpp

```
template<class T>
struct Complex{
    T x,y;
    Complex(T x=0,T y=0):x(x),y(y){}
    operator T(){return x;}
};
template<class T>
Complex<T> operator+(const Complex<T> a,const Complex<T> b){return Complex<T>(a.x+b.x,a.y+b.y);}
template<class T>
Complex<T> operator-(const Complex<T> a,const Complex<T> b){return Complex<T>(a.x-b.x,a.y-b.y);}
template<class T>
Complex<T> operator*(const Complex<T> a,const Complex<T> b){return Complex<T>(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);}
template<class T>
Complex<T> operator/(const Complex<T> a,const Complex<T> b){return Complex<T>(a.x*b.x+a.y*b.y,a.x*b.y-a.y*b.x)/(b.x*b.x+b.y*b.y);}
template<class T>
Complex<T>& operator+=(Complex<T> &a,const Complex<T> b){a.x+=b.x;a.y+=b.y;return a;}
template<class T>
Complex<T>& operator-=(Complex<T> &a,const Complex<T> b){a.x-=b.x;a.y-=b.y;return a;}
template<class T>
Complex<T>& operator*=(Complex<T> &a,const Complex<T> b){tie(a.x,a.y)=make_tuple(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);return a;}
template<class T>
Complex<T>& operator/=(Complex<T> &a,const Complex<T> b){tie(a.x,a.y)=make_tuple((a.x*b.x+a.y*b.y,a.x*b.y-a.y*b.x)/(b.x*b.x+b.y*b.y));return a;}
template<class T,template <class U> class complexT=Complex>
struct fft{
    static constexpr T Pi=M_PI;
    int size,l,n;
    vector<int> r;
    vector<complexT<T>> W[2];
    int floorintlog2(int i){
        int k=0;
        while (i) i>>=1,k++;
        return k;
    }
    fft(int size):size(size){
        l=floorintlog2(size);
```

```
        n=1<<l;
        r.resize(n,0);
        W[0].resize(n,{0,0});
        W[1].resize(n,{0,0});
        for (int i=0;i<n;i++)
            r[i]=(r[i>>1]>>1)|((i&1)<<(l-1));
        for (int type:{0,1})
            for (int i=0;i<n;i++)
                W[type][i]={cos(Pi/i),(type?1:-1)*sin(Pi/i)};
    }
    template<int type,class U>
    valarray<complexT<T>> _FFT(const U& B)const{
        using cT=complexT<T>;
        valarray<cT> A(n);
        copy(std::begin(B),std::end(B),begin(A));
        for (int i=0;i<n;i++)
            if(i<r[i]) swap(A[i],A[r[i]]);
        for (int mid=1;mid<n;mid<<=1){
            const cT Wn=W[type][mid];
            for (int R=mid<<1,j=0;j<n;j+=R){
                cT w(1,0);
                for (int k=0;k<mid;k++,w=w*Wn){
                    const cT x=A[j+k],y=w*A[j+mid+k];
                    A[j+k]=x+y;
                    A[j+mid+k]=x-y;
                }
            }
        }
        return A;
    }
    template<class U>
    valarray<complexT<T>> FFT(const U& A)const{return _FFT<0>(A);}
    template<class vT>
    valarray<vT> DFT(const valarray<complexT<T>>& A)const{
        auto b=_FFT<1>(A);
        valarray<vT> a(size);
        if (is_integral<vT>::value)
            for (int i=0;i<size;i++)
                a[i]=llround(b[i]/n);
        else
            for (int i=0;i<n;i++)
                a[i]=b[i]/n;
        return a;
    }
};
```

floyd.hpp

```
template<class vT=int,class GT>
vector<vector<vT>> floyd(const GT &G){
    const auto n=G.size();
    vector<vector<vT>> dp(n,vector<vT>(n,INF));
    for (size_t i=0;i<n;i++) dp[i][i]=0;
    for (size_t k=0;k<n;k++)
        for (auto &e:G[k])
```

```
            cmin(dp[k][e.v],e.w);
    for (size_t k=0;k<n;k++)
        for (size_t i=0;i<n;i++)
            for (size_t j=0;j<n;j++)
                cmin(dp[i][j],dp[i][k]+dp[k][j]);
    return dp;
}
```

graph.hpp

```
template <class vT>
struct wedge{int u,v;vT w;};
template <class vT>
struct sedge{int u,v;const constexpr static vT w=1;};
template <template <class vT> class etT,class vT>
struct graph:public vector<vector<etT<vT>>>{
    using eT=etT<vT>;
    using esT=vector<eT>;
    using GT=vector<esT>;
    using vector<vector<etT<vT>>>::vector;
    void addEdge(const eT &a){GT::operator[](a.u).push_back(a);}
    void add2Edge(eT a){addEdge(a);swap(a.u,a.v);addEdge(a);}
};
```

kdbitree.hpp

```
template<class T>
T lowbit(T a){return a&(-a);}
template<class T,size_t n,size_t d,size_t i=0>
struct kdbitree{
    kdbitree<T,n,d,i+1> a[n];
    void add(const array<int,d>& ax,T v){
        auto x=ax[i];
        while (x<n) a[x].add(ax,v),x+=lowbit(x);
    }
    T sum(const array<int,d>& ax){
        T ans=0;
        auto x=ax[i];
        while (x>0) ans+=a[x].sum(ax),x-=lowbit(x);
        return ans;
    }
};
template<class T,size_t n,size_t d>
struct kdbitree<T,n,d,d>{
    T a;
    kdbitree():a(0){}
    void add(const array<int,d>& ax,T v){a+=v;}
    T sum(const array<int,d>& ax){return a;}
};
```

kmp.hpp

```
template<class T>
vector<int> to_next(T s){
```

```
    vector<int> next(s.size(),-1);
    next[0]=-1;
    for (int i=1;i<s.size();i++){
        int w=next[i-1];
        while (w!=-1&&s[w+1]!=s[i]) w=next[w];
        next[i]=(s[w+1]==s[i])+w;
    }
    return next;
}
template<class T>
vector<int> kmp(T a,T b){
    vector<int> pos;
    auto bn=to_next(b);
    int ai=0,bi=0;
    while (ai<=a.size()){
        if (bi!=b.size()&&a[ai]==b[bi]){
            ai++;bi++;
        }else{
            if (bi==b.size()){
                pos.push_back(ai-bi);
            }
            if (bi!=0){
                bi=bn[bi-1]+1;
            }else{
                ai++;
            }
        }
    }
    return pos;
}
```

math.hpp

```
#define ATL_MATH
constexpr ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
constexpr ll lcm(ll a,ll b){return a*b/gcd(a,b);}
template<class T>
T power(T a,size_t b,const T &unit=1){
    if (b==0) return unit;
    if (b&1) return a*power(a*a,b>>1,unit);
    return power(a*a,b>>1,unit);
}
constexpr ll ceildiv(const ll a,const ll b){return a/b+(a%b?1:0);}
tuple<ll,ll,ll> exgcd(ll a,ll b){//a1+b2=gcd(a,b)
    if (b==0) return {a,1,0};
    ll g,x,y;
    tie(g,x,y)=exgcd(b,a%b);
    return make_tuple(g,y,x-a/b*y);
}
tuple<ll,ll,ll> Fexgcd(ll a,ll b){//a1+b2=gcd(a,b),ensure 1>0
    auto k=exgcd(a,b);
    if (get<1>(k)<0) {
        get<1>(k)+=b;
        get<2>(k)-=a;
    }
```

```
    return k;
}
```

matrix.hpp

```
template<class T,size_t n,size_t m>
struct matrix:public valarray<valarray<T>>{
    using base1=valarray<T>;
    using base2=valarray<base1>;
    using base2::base2;
    matrix(const T &a):base2(base1(a,m),n){}
    matrix():base2(base1(m),n){}
};

template<class T,size_t n,size_t m>
matrix<T,n,n> operator*(const matrix<T,n,m> &a,const matrix<T,m,n> &b){
    matrix<T,n,n> x;
    for (size_t i=0;i<n;i++)
        for (size_t j=0;j<n;j++)
            for (size_t k=0;k<m;k++)
                x[i][j]+=a[i][k]*b[k][j];
    return x;
}

template<class T,class U>
auto operator*(const valarray<T> &a,const U &b)->decltype(b+=1,a){
    valarray<T> x(a);
    for (auto &i:x) i*=b;
    return x;
}

template<class T,class U>
auto operator*=(valarray<T> &a,const U &b)->decltype(b+=1,a){
    for (auto &i:a) i*=b;
    return a;
}

template<class T,size_t n>
matrix<T,n,n> unitmatrix(){
    matrix<T,n,n> m;
    for (int i=0;i<n;i++) m[i][i]=1;
    return m;
}

template<class T,size_t n>
pair<bool,matrix<T,n,n>> inverse(matrix<T,n,n> a){
    auto b=unitmatrix<T,n>();
    for (int i=0;i<n-1;i++){
        int k=-1;
        for (int j=i;j<n;j++) if (a[j][i]!=0) {k=j;break;}
        if (k!=-1){
            swap(b[i],b[k]);swap(a[i],a[k]);
            for (int j=i+1;j<n;j++)
                if (a[k][i]!=0){
                    T d=a[j][i]/a[k][i];
```

```
                    a[j]-=a[k]*d;
                    b[j]-=b[k]*d;
                }
        }else return {false,b};
    }
    for (int i=n-1;i>=0;i--){
        b[i]/=a[i][i];a[i]/=a[i][i];
        for (int j=i-1;j>=0;j--){
            b[j]-=b[i]*a[j][i];
            a[j]-=a[i]*a[j][i];
        }
    }
    return {true,b};
}
```

mfset.hpp

```
struct mfset:protected vector<int>{
    mfset(){}
    mfset(int size){resize(size);}
    void resize(int size){
        vector<int>::resize(size);
        iota(this->begin(),this->end(),0);
    }
    int find(int a){
        int &b=this->operator[](a);
        return a==b?a:b=find(b);
    }
    void merge(int a,int b){
        int aa=find(a),bb=find(b);
        if (aa!=bb)
            this->operator[](bb)=aa;
    }
};
```

mint.hpp

```
#define op_mint(op)\
_mint operator op (const _mint a)const{_mint k(*this);k op##=a;return k;}
#define cmp_mint(op)\
bool operator op (const _mint a)const{return v op a.v;}
template<class T>
struct _mint{
    T v;
    static T mod;
    _mint()=default;
    _mint(const T &a){(v=a%mod)<0&&(v+=mod);}
    _mint& operator+=(const _mint a){return (v+=a.v)>=mod&&(v-=mod),*this;}
    _mint& operator-=(const _mint a){return (v-=a.v)<0&&(v+=mod),*this;}
    _mint& operator*=(const _mint a){return (v*=a.v)%=mod,*this;}
    op_mint(+) op_mint(-) op_mint(*)
    cmp_mint(<) cmp_mint(>) cmp_mint(<=) cmp_mint(>=) cmp_mint(!=) cmp_mint(==)
    #ifdef ATL_MATH
    _mint inverse(){_mint a;a.v=get<1>(Fexgcd(v,mod));return a;}
    _mint& operator/=(const _mint a){return (*this)*=a.inverse()%=mod,*this;}
```

```
    op_mint(/)
    #endif
};
template<class T>
T _mint<T>::mod;
template<class T>
ostream& operator<<(ostream& os,const _mint<T>& a){return os<<a.v;}
template<class T>
istream& operator>>(istream& os,_mint<T>& a){T k;os>>k;a=_mint<T>(k);return os;}
using mint=_mint<int>;
using mll=_mint<ll>;
```

network.hpp

```
template <class vT>
struct fedge{int u,v;vT cap,flow;fedge* iedge;};
template <class vT>
struct cfedge{int u,v;vT cap,cost,flow;cfedge* iedge;};
template <class edgeT>
struct network:public vector<list<edgeT>>{
    using vector<list<edgeT>>::vector;
    using eT=edgeT;
    using esT=list<eT>;
    using GT=vector<esT>;
    eT* addFlow(const eT &a){
        auto &l=GT::operator[](a.u);
        return &*l.insert(l.end(),a);
    }
    void add2Flow(eT a){
        a.flow=0;
        auto b=addEdge(a);
        swap(a.u,a.v);a.cap=0;
        auto c=addEdge(a);
        tie(b->iedge,c->iedge)=make_tuple(c,b);
    }
};
```

prime.hpp

```
template <size_t n>
struct Primes{
    bitset<n> book;
    array<size_t,n> phi;
    vector<size_t> pri;
    Primes(){
        phi[1]=1;
        for (size_t i=2;i<n;i++){
            if (!book[i]) {
                phi[i]=i-1;
                pri.pb(i);
            }
            for (size_t j=0;j<pri.size();++j) {
                size_t w=i*pri[j];
                if (w>=n) break;
                book[w]=1;
```

```
                if (i%pri[j]) {
                    phi[w]=phi[i]*(pri[j]-1);
                } else {
                    phi[w]=phi[i]*pri[j];
                    break;
                }
            }
        }
    }
    template <class T>
    bool is(const T &a)const{return book[a];}
    ll operator[](const size_t &i)const{return pri[i];}
};
Primes<100000> primes;
```

segtree.hpp

```
template<class T>
struct segtree{
    T fsum;
    virtual ~segtree(){}
    virtual void dadd(const T &nadd)=0;
    virtual void add(const ll &l,const ll &r,const T &nadd)=0;
    virtual void dmul(const T &nadd)=0;
    virtual void mul(const ll &l,const ll &r,const T &nadd)=0;
    virtual T sum(const ll &l,const ll &r)=0;
};
template<class T,class U>
unique_ptr<segtree<T>> make_seg(const U& op,const U& ed);
template<class T>
struct segtreec:public segtree<T>{
    using segtree<T>::fsum;
    unique_ptr<segtree<T>> lson,rson;
    ll count,mid;
    T fadd,fmul;
    template<class U>
    segtreec(const U& op,const U& ed):fadd(0),fmul(1){
        mid=(count=ed-op)>>1;
        lson=make_seg<T>(op,op+mid);
        rson=make_seg<T>(op+mid,ed);
        pushup();
    }
    void pushdown(){
        lson->dmul(fmul);
        rson->dmul(fmul);
        fmul=1;
        lson->dadd(fadd);
        rson->dadd(fadd);
        fadd=0;
    }
    void pushup(){
        fsum=lson->fsum+rson->fsum;
    }
    void dadd(const T &nadd){
        fadd+=nadd;
```

```
        fsum+=nadd*count;
    }
    void add(const ll &l,const ll &r,const T &nadd){
        if (l<=1&&count<=r)
            dadd(nadd);
        else{
            pushdown();
            if (l<=mid) lson->add(l,r,nadd);
            if (r>mid) rson->add(l-mid,r-mid,nadd);
            pushup();
        }
    }
    void dmul(const T &nmul){
        fmul*=nmul;
        fadd*=nmul;
        fsum*=nmul;
    }
    void mul(const ll &l,const ll &r,const T &nmul){
        if (l<=1&&count<=r)
            dmul(nmul);
        else{
            pushdown();
            if (l<=mid) lson->mul(l,r,nmul);
            if (r>mid) rson->mul(l-mid,r-mid,nmul);
            pushup();
        }
    }
    T sum(const ll &l,const ll &r){
        if (l<=1&&count<=r) return fsum;
        pushdown();
        T ans=0;
        if (l<=mid) ans+=lson->sum(l,r);
        if (r>mid) ans+=rson->sum(l-mid,r-mid);
        return ans;
    }
};
template<class T>
struct segtreen:public segtree<T>{
    using segtree<T>::fsum;
    template<class U>
    segtreen(const U& op,const U& ed){fsum=*op;}
    void dadd(const T &nadd){fsum+=nadd;}
    void add(const ll &l,const ll &r,const T &nadd){fsum+=nadd;}
    void dmul(const T &nmul){fsum*=nmul;}
    void mul(const ll &l,const ll &r,const T &nmul){fsum*=nmul;}
    T sum(const ll &l,const ll &r){return fsum;}
};
template<class T,class U>
unique_ptr<segtree<T>> make_seg(const U& op,const U& ed){
    if (ed-op==1) return unique_ptr<segtree<T>>(new segtreen<T>(op,ed));
    else return unique_ptr<segtree<T>>(new segtreec<T>(op,ed));
}
```

sgttree.hpp

```
template<class T,class iT>
struct sgttree{
    constexpr static double s_alpha=0.724;
    constexpr static double s_beta=0.35;
    struct Tnode{
        static Tnode nilnode;
        static Tnode* nil;
        T val;
        iT cnt,size,cover,rsize;
        Tnode *ch[2];
        Tnode (T v,iT c){val=v;cnt=size=c;rsize=(c!=0);cover=1;}
        ~Tnode(){
            if (ch[0]!=Tnode::nil) delete ch[0];
            if (ch[1]!=Tnode::nil) delete ch[1];
        }
        void maintain(){
                        size=cnt+ch[0]->size+ch[1]->size;
                        rsize=(cnt!=0?1:0)+ch[0]->rsize+ch[1]->rsize;
                        cover=1+ch[0]->cover+ch[1]->cover;}
        void rmaintain(){maintain();cover=rsize;}
        bool isBad(){return max(ch[0]->cover,ch[1]->cover)>=
                            (s_alpha*cover);}
        char cmp(T v){
            if (v==val) return -1;
            return v>=val;
        }
        char cmpkth(iT &k){
            iT p=k-ch[0]->size;
            if (p<=0) return 0;
            k=p;p-=cnt;
            if (p<=0) return -1;
            k=p;return 1;
        }
    };
    Tnode *head;

    sgttree():head(Tnode::nil){}
    ~sgttree(){if (head!=Tnode::nil) delete head;}
    int ci;
    void toArr(Tnode *e,vector<Tnode*> &g){
        if (e==Tnode::nil) return ;
        if (e->ch[0]!=Tnode::nil) toArr(e->ch[0],g);
        if (e->cnt) g.pb(e);
        if (e->ch[1]!=Tnode::nil) toArr(e->ch[1],g);
        if (!e->cnt) delete e;
    }

    Tnode* toTree(int l,int r,vector<Tnode*> &g){
        if (l>=r) return Tnode::nil;
        int mid=(l+r)>>1;
        Tnode &e=*(g[mid]);
        e.ch[0]=toTree(l,mid,g);
        e.ch[1]=toTree(mid+1,r,g);
        e.maintain();
```

```
55          return &e;
56      }
57
58      void reBuild(Tnode *&e){
59          if (e!=Tnode::nil){
60              vector<Tnode*> g;
61              g.reserve(e->cover);
62              toArr(e,g);
63              e=toTree(0,g.size(),g);
64          }
65      }
66
67      Tnode **to;
68      T v;iT s;
69      void insert(Tnode *&e){
70          if (e==Tnode::nil){
71              e=new Tnode(v,s);
72              e->ch[0]=e->ch[1]=Tnode::nil;
73              return ;
74          }
75          char d=e->cmp(v);
76          if (d==-1)
77              e->cnt+=s;
78          else insert(e->ch[d]);
79          e->maintain();
80          if (e->isBad()) to=&e,e->rmaintain();
81      }
82
83      void erase(Tnode *&e){
84          if (e==Tnode::nil) return ;
85          char d=e->cmp(v);
86          if (d==-1)
87              e->cnt=max(0,e->cnt-s);
88          else erase(e->ch[d]);
89          e->maintain();
90      }
91
92      void insert(T vs,iT ss=1){
93          v=vs;s=ss;
94          to=&Tnode::nil;
95          insert(head);
96          reBuild(*to);
97      }
98
99      void erase(T vs,iT ss=1){
100         v=vs;s=ss;
101         erase(head);
102         if ((head->cover-head->rsize)>head->cover*s_beta)
103             reBuild(head);
104     }
105
106     iT rank(T v){
107         Tnode *e=head;iT k=1;char d;
108         while (e!=Tnode::nil&&(d=e->cmp(v))!=-1){
109             if (d==1) k+=e->ch[0]->size+e->cnt;
110             e=e->ch[d];
```

```
        }
        return k+e->ch[0]->size;
    }

    Tnode* kth(iT k){
        Tnode *e=head;char d;
        while ((d=e->cmpkth(k))!=-1) e=e->ch[d];
        return e;
    }
};
template<class T,class IT>
typename sgttree<T,IT>::Tnode sgttree<T,IT>::Tnode::nilnode=[](){
    Tnode nil(0,0);
    nil.rsize=nil.cover=0;
    nil.ch[0]=nil.ch[1]=&sgttree<T,IT>::Tnode::nilnode;;
    return nil;
}();
template<class T,class IT>
typename sgttree<T,IT>::Tnode* sgttree<T,IT>::Tnode::nil=&sgttree<T,IT>::Tnode::nilnode;
```

splay.hpp

```
template<class T,class iT>
struct splay{
    struct Tnode{
        T val;
        iT cnt,size;
        Tnode *ch[2];
        Tnode (T v,iT c){val=v;size=cnt=c;}
        void maintain(){
            size=ch[0]->size+ch[1]->size+cnt;
        }
        char cmp(const T& v){
            if (v==val) return -1;
            return val<v;
        }
        char cmpkth(iT &k){
            iT p=k-ch[0]->size;
            if (p<=0) return 0;
            k=p;p-=cnt;
            if (p<=0) return -1;
            k=p;return 1;
        }
    };
    static Tnode nilnode;
    static Tnode* nil;
    Tnode *head;

    splay(){
        head=nil;
    }

    void rotate(Tnode *&e,const char d){
        if (e==nil) return ;
        Tnode *&a=e,*&b=e->ch[d],*&c=e->ch[d]->ch[d^1];
```

```
34          swap(a,b);swap(b,c);
35          e->ch[d^1]->maintain();
36          e->maintain();
37      }
38
39      void Splay(Tnode *&e,const T& v){
40          if (e==nil) return ;
41          char d1=e->cmp(v);
42          if (d1==-1 || e->ch[d1]==nil){
43              return ;
44          }else{
45              Tnode *&n1=e->ch[d1];
46              char d2=n1->cmp(v);
47              if (d2==-1 || n1->ch[d2]==nil) {rotate(e,d1);return ;}
48              else{
49                  Splay(n1->ch[d2],v);
50                  if (d1==d2) {rotate(e,d1);rotate(e,d1);}
51                  else {rotate(n1,d2);rotate(e,d1);}
52              }
53          }
54      }
55
56      void splaykth(Tnode *&e,iT& k){
57          if (e==nil) return ;
58          char d1=e->cmpkth(k);
59          if (d1==-1 || e->ch[d1]==nil){
60              return ;
61          }else{
62              Tnode *&n1=e->ch[d1];
63              char d2=n1->cmpkth(k);
64              if (d2==-1 || n1->ch[d2]==nil) {rotate(e,d1);return ;}
65              else{
66                  splaykth(n1->ch[d2],k);
67                  if (d1==d2) {rotate(e,d1);rotate(e,d1);}
68                  else {rotate(n1,d2);rotate(e,d1);}
69              }
70          }
71      }
72
73      void _insert(const T& vs,const iT& ss,Tnode *&e){
74          if (e==nil){
75              e=new Tnode(vs,ss);
76              e->ch[0]=e->ch[1]=nil;
77              return ;
78          }
79          char d=e->cmp(vs);
80          if (d==-1){
81              e->cnt+=ss;
82          }else _insert(vs,ss,e->ch[d]);
83      }
84
85      void insert(const T& vs,const iT& ss=1){
86          _insert(vs,ss,head);
87          Splay(head,vs);
88      }
89
```

```
    void erase(const T& vs,const iT& ss=1){
        Splay(head,vs);
        if (ss<head->cnt){
            head->cnt-=ss;
            head->size-=ss;
            return ;
        }
        Tnode *l=head->ch[0],*r=head->ch[1];
        delete head;
        if (r!=nil){
            Splay(r,vs+1);
            r->ch[0]=l;
            r->maintain();
            head=r;
        }else head=l;
    }

    iT rank(T v){
        Splay(head,v);
        return head->ch[0]->size+(head->val<v?head->cnt:0)+1;
    }

    Tnode* kth(iT k){
        splaykth(head,k);
        return head;
    }

};
template<class T,class IT>
typename splay<T,IT>::Tnode splay<T,IT>::nilnode(0,0);
template<class T,class IT>
typename splay<T,IT>::Tnode* splay<T,IT>::nil=&splay<T,IT>::nilnode;
```

vector.hpp

```
#define op_array(x) \
template<class T,size_t d> \
array<T,d>& operator x##=(array<T,d>& a,const array<T,d>& b){for (size_t i=0;i<d;i++) a[i]
 ↪  x##=b[i];return a;}\
template<class T,class U,size_t d>\
auto operator x##=(array<T,d>& a,const U& b)->decltype(T(declval<U>()),a){for (size_t
 ↪  i=0;i<d;i++) a[i] x##=b;return a;}\
template<class T,size_t d,class U>\
auto operator x (const array<T,d>& a,const U&
 ↪  b)->decltype(T(declval<U>()),array<T,d>()){array<T,d> k(a);k x##=b;return k;}
op_array(+) op_array(-) op_array(*) op_array(/)
template<class T>
using vec2=array<T,2>;
template<class T>
using vec3=array<T,3>;
template<class T,size_t d>
T dot(const array<T,d>& a,const array<T,d>& b){
    T ans=0;
    for (size_t i=0;i<d;i++) ans+=a[i]*b[i];
    return ans;
```

```
}
template<class T,size_t d>
T abs(const array<T,d>& a){return sqrt(dot(a,a));}
template<class T>
T crs(const vec2<T> &a,const vec2<T> &b){return a[0]*b[1]-a[1]*b[0];}
template<class T>
vec3<T> crs(const vec3<T> &a,const vec3<T> &b){return
 ↪ {a[1]*b[2]-a[2]*b[1],a[2]*b[0]-a[0]*b[2],a[0]*b[1]-a[1]*b[0]};}
template<class T,size_t d>
bool operator<(const array<T,d>& a,const array<T,d>& b){for (size_t i=0;i<d;i++) if
 ↪ (a[i]<b[i]) return true; else if (a[i]<b[i]) return false;return false;}
template<class T,size_t d>
istream& operator>>(istream& is,array<T,d> &p){
    for (size_t i=0;i<d;i++) is>>p[i];
    return is;
}
```