Chulalongkorn University

”” ”””,\0\n

Teetat T., Thawin T., Thitrin S.

2025-07-19

1 Contest 1

2 Data structures 1

3 Number Theory 2

4 Tree 2

5 Flows 4

6 String 6

7 Convolutions 7

Contest (1)

template.hpp

<bits/stdc++.h>, <ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> **1ac233**, 22 lines

```
using namespace std;
using namespace __gnu_pbds;

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 ll LINF=0x1fffffffffffffff;
const db DINF=numeric_limits<db>::infinity();
const db EPS=1e-9;
const db PI=acos(db(-1));

template<class T>
using ordered_set = tree<T,null_type,less<T>,rb_tree_tag,
    tree_order_statistics_node_update>;

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
mt19937_64 rng64(chrono::steady_clock::now().time_since_epoch()
    .count());
```

.bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Data structures (2)

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

line-container.hpp

Description: Line Container (Minimize).
Time: $\mathcal{O}(\log N)$

24a5c4, 37 lines

```
struct Line{
  static bool querymode;
    ll m,c;
    mutable ll p;
    Line(ll m,ll c):m(m),c(c),p(0){}
    Line(ll p):m(0),c(0),p(p){}
    bool operator<(const Line &o)const{
        return querymode?p<o.p:m>o.m;
    }
};

bool Line::querymode=false;

struct LineContainer:multiset<Line>{
    ll div(ll a,ll b){
        return a/b-((a^b)<0&&a%b);
    }
    bool isect(iterator x,iterator y){
        if(y==end())return x->p=LINF,false;
        if(x->m==y->m)x->p=x->c<=y->c?LINF:-LINF;
        else x->p=div(x->c-y->c,y->m-x->m);
        return x->p>=y->p;
    }
    void add(ll m,ll c){
        auto x=insert(Line(m,c)),y=next(x);
        while(isect(x,y))y=erase(y);
        if((y=x)!=begin()&&isect(--x,y))isect(x,erase(y));
        while((y=x)!=begin()&&(--x)->p>=y->p)isect(x,erase(y));
    }
    ll get(ll x){
        if(empty())return LINF;
        Line::querymode=true;
        auto l=lower_bound(Line(x));
        Line::querymode=false;
        return l->m*x+l->c;
    }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$

1754b4, 53 lines

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
    auto [L,R] = split(n->l, k);
```

```
    n->l = R;
    n->recalc();
    return {L, n};
  } else {
    auto [L,R] = split(n->r,k - cnt(n->l) - 1); // and just "k"
    n->r = L;
    n->recalc();
    return {n, R};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    return l->recalc(), l;
  } else {
    r->l = merge(l, r->l);
    return r->recalc(), r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto [l,r] = split(t, pos);
  return merge(merge(l, n), r);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

segment-tree-beats.hpp

Description: Segment Tree Beats

bc2a26, 134 lines

```
struct SegmentTreeBeats{
    struct Node{
        ll sum,add;
        ll mn,mn2,fn;
        ll mx,mx2,fx;
        Node(){
            sum=add=fn=fx=0,mn=mn2=LINF,mx=mx2=-LINF;
        }
        Node(ll v){
            sum=mn=mx=v,add=0,mn2=LINF,mx2=-LINF,fn=fx=1;
        }
        friend Node operator+(const Node &l,const Node &r){
            Node res;
            res.sum=l.sum+r.sum;
            res.add=0;
            if(l.mx>r.mx){
                res.mx=l.mx,res.fx=l.fx;
                res.mx2=max(l.mx2,r.mx);
            }else if(r.mx>l.mx){
                res.mx=r.mx,res.fx=r.fx;
                res.mx2=max(r.mx2,l.mx);
            }else{
                res.mx=l.mx,res.fx=l.fx+r.fx;
                res.mx2=max(l.mx2,r.mx2);
            }
            if(l.mn<r.mn){
                res.mn=l.mn,res.fn=l.fn;
                res.mn2=min(l.mn2,r.mn);
            }else if(r.mn<l.mn){
                res.mn=r.mn,res.fn=r.fn;
                res.mn2=min(r.mn2,l.mn);
            }else{
                res.mn=l.mn,res.fn=l.fn+r.fn;
                res.mn2=min(l.mn2,r.mn2);
            }
            return res;
        }
        void apply(int l,int r,ll v){
            sum+=(r-l+1)*v;
            mx+=v,mx2+=v;
            mn+=v,mn2+=v;
            add+=v;
        }
        void chmin(ll v){
            if(v>=mx)return;
            sum+=(v-mx)*fx;
            if(mn==mx)mn=v;
            if(mn2==mx)mn2=v;
            mx=v;
        }
        void chmax(ll v){
            if(v<=mn)return;
            sum+=(v-mn)*fn;
            if(mx==mn)mx=v;
            if(mx2==mn)mx2=v;
            mn=v;
        }
    };
    int n;
    vector<Node> t;
    SegmentTreeBeats(){}
    SegmentTreeBeats(int n){init(n,[&](int){return 0;});}
    template<class F>
    SegmentTreeBeats(int n,const F &f){init(n,f);}
    template<class F>
    void init(int _n,const F &f){
        n=_n;
        int s=1;
        while(s<n*2)s<<=1;
        t.assign(s,Node());
        build(f);
    }
    template<class F>
    void build(int l,int r,int i,const F &f){
        if(l==r)return void(t[i]=f(l));
        int m=(l+r)/2;
        build(l,m,i*2,f);
        build(m+1,r,i*2+1,f);
        pull(i);
    }
    void pull(int i){
        t[i]=t[i*2]+t[i*2+1];
    }
    void push(int l,int r,int i){
        int m=(l+r)/2;
        t[i*2].apply(l,m,t[i].add);
        t[i*2+1].apply(m+1,r,t[i].add);
        t[i*2].chmin(t[i].mx);
        t[i*2+1].chmin(t[i].mx);
        t[i*2].chmax(t[i].mn);
        t[i*2+1].chmax(t[i].mn);
        t[i].add=0;
    }
    void range_add(int l,int r,int i,int x,int y,ll v){
        if(y<l||r<x)return;
        if(x<=l&&r<=y)return t[i].apply(l,r,v);
        int m=(l+r)/2;
        push(l,r,i);
        range_add(l,m,i*2,x,y,v);
        range_add(m+1,r,i*2+1,x,y,v);
        pull(i);
    }
    void range_chmin(int l,int r,int i,int x,int y,ll v){
        if(y<l||r<x||t[i].mx<=v)return;
        if(x<=l&&r<=y&&t[i].mx2<v)return t[i].chmin(v);
        int m=(l+r)/2;
        push(l,r,i);
        range_chmin(l,m,i*2,x,y,v);
        range_chmin(m+1,r,i*2+1,x,y,v);
        pull(i);
    }
    void range_chmax(int l,int r,int i,int x,int y,ll v){
        if(y<l||r<x||t[i].mn>=v)return;
        if(x<=l&&r<=y&&t[i].mn2>v)return t[i].chmax(v);
        int m=(l+r)/2;
        push(l,r,i);
        range_chmax(l,m,i*2,x,y,v);
        range_chmax(m+1,r,i*2+1,x,y,v);
        pull(i);
    }
    ll query(int l,int r,int i,int x,int y){
        if(y<l||r<x)return 0;
        if(x<=l&&r<=y)return t[i].sum;
        int m=(l+r)/2;
        push(l,r,i);
        return query(l,m,i*2,x,y)+query(m+1,r,i*2+1,x,y);
    }
    template<class F>
    void build(const F &f){build(0,n-1,1,f);}
    void range_add(int x,int y,ll v){range_add(0,n-1,1,x,y,v);}
    void range_chmin(int x,int y,ll v){range_chmin(0,n-1,1,x,y,
        v);}
    void range_chmax(int x,int y,ll v){range_chmax(0,n-1,1,x,y,
        v);}
    ll query(int x,int y){return query(0,n-1,1,x,y);}
};
```

Number Theory (3)

euclid.hpp

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 $a \pmod b$. $x = x_0 + k * (b/g)$ $y = y_0 - k * (a/g)$

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 $x \equiv a \pmod m$, $x \equiv b \pmod n$). If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$. If x0 and y0 is one of the solutions of ax + by = g, then the general solution is x = x0 + k * (b / g) and y = y0 - k * (a / g).

Time: $\log(n)$

"src/number-theory/euclid.hpp" cfd447, 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);
  if((a-b)%g!=0)return -1LL; // no solution
  x=(b-a)%n*x%n/g*m+a;
  return x<0?x+m*n/g:x;
}
```

floor-sum.hpp

Description: Floor sum function. $f(a,b,c,n) = \sum_{x=0}^{n} \lfloor \frac{ax+b}{c} \rfloor$ becareful when a,b,c are negetive (use custom floor division and mod instead)
Time: $\mathcal{O}(\log a)$

d088d2, 7 lines

```
ll floor_sum(ll a,ll b,ll c,ll n){
    ll res=n*(n+1)/2*(a/c)+(n+1)*(b/c);
    a%=c,b%=c;
    if(a==0)return res;
    ll m=(a*n+b)/c;
    return res+n*m-floor_sum(c,c-b-1,a,m-1);
}
```

Tree (4)

link-cut-tree.hpp

Description: Link Cut Tree (1-indexed)

38324f, 78 lines

```
template<int N,class T>
struct LinkCutTree{
    int ch[N][2],par[N],lz[N],rev[N];
    T val[N],sum[N],rsum[N];
    void toggle(int v){
        if(!v)return;
        swap(ch[v][0],ch[v][1]);
        swap(sum[v],rsum[v]);
        rev[v]^=1;
    }
    void push(int v){
        if(!v||!rev[v])return;
        toggle(ch[v][0]);
        toggle(ch[v][1]);
        rev[v]=0;
    }
    void pull(int v){
        if(!v)return;
        sum[v]=sum[ch[v][0]]+val[v]+sum[ch[v][1]];
        rsum[v]=rsum[ch[v][0]]+val[v]+rsum[ch[v][1]];
    }
    bool is_root(int v){
        return ch[par[v]][0]!=v&&ch[par[v]][1]!=v;
    }
    bool pos(int v){
        return ch[par[v]][1]==v;
    }
    void rotate(int v){
        int u=par[v],g=par[u];
        bool x=pos(v);
        if(!is_root(u))ch[g][pos(u)]=v;
        ch[u][x]=ch[v][!x];
        if(ch[u][x])par[ch[u][x]]=u;
        ch[v][!x]=u,par[u]=v,par[v]=g;
        pull(u),pull(v);
    }
    void splay(int v){
        if(!v)return;
        for(push(v);!is_root(v);rotate(v)){
            int u=par[v];
            if(is_root(u))push(u),push(v);
            else push(par[u]),push(u),push(v),rotate(pos(u)==
                 pos(v)?u:v);
        }
    }
    void access(int v){
        for(int u=v,c=0;u;u=par[u]){
            splay(u);
            ch[u][1]=c;
            pull(c=u);
        }
        splay(v);
    }
    void evert(int v){
        access(v),toggle(v);
    }
    void link(int u,int v){
        evert(u);
        access(v);
        par[u]=v;
    }
    void cut(int u,int v){
        evert(u);
        access(v);
        assert(par[u]==v);
        ch[v][0]=par[u]=0;
        pull(v);
    }
    T aggregate(int u,int v){
        evert(u);
        access(v);
        return sum[v];
    }
    void set(int u,T v){
        evert(u);
        val[u]=v;
        pull(u);
    }
};
```

static-top-tree.hpp

Description: Static Top Tree.

"src/contest/template.hpp" d0731a, 198 lines

```
template<class G>
struct StaticTopTree{
    using P = pair<int,int>;
    enum Type{Compress,Rake,AddEdge,AddVertex,Vertex};
    int n,root;
    G &adj;
    vector<int> hv,fa,lch,rch,par;
    vector<Type> type;
    StaticTopTree(G &adj):adj(adj){build();}
    int dfs(int u){
        int s=1,mx=0;
        for(auto v:adj[u]){
            if(v==fa[u])continue;
            fa[v]=u;
            int t=dfs(v);
            if(t>mx)mx=t,hv[u]=v;
            s+=t;
        }
        return s;
    }
    void build(){
        n=adj.size();
        hv=fa=lch=rch=par=vector<int>(n,-1);
        type.assign(n,Compress);
        dfs(0,-1);
        root=compress(0).second;
    }
    int add(int i,int l,int r,Type t){
        if(i==-1){
            i=n++;
            lch.emplace_back(l);
            rch.emplace_back(r);
            par.emplace_back(-1);
            type.emplace_back(t);
        }else{
            lch[i]=l,rch[i]=r,type[i]=t;
        }
        if(l!=-1)par[l]=i;
        if(r!=-1)par[r]=i;
        return i;
    }
    /*
    pair<int,int> merge(vector<pair<int,int>> a,Type t){
        if(a.size()==1)return a[0];
        int tot=0;
        vector<pair<int,int>> l,r;
        for(auto [i,s]:a)tot+=s;
        for(auto [i,s]:a){
            (tot>s?l:r).emplace_back(i,s);
            tot-=s*2;
        }
        auto [i,si]=merge(l,t);
        auto [j,sj]=merge(r,t);
        return {add(-1,i,j,t),si+sj};
    }
    */
    P compress(int i){
        vector<P> a{add_vertex(i)};
        auto work=[&](){
            auto [sj,j]=a.back();
            a.pop_back();
            auto [si,i]=a.back();
            a.back()={max(si,sj)+1,add(-1,i,j,Compress)};
        };
        while(hv[i]!=-1){
            a.emplace_back(add_vertex(i=hv[i]));
            while(true){
                if(a.size()>=3&&(a.end()[-3].first==a.end()
                    [-2].first||a.end()[-3].first<=a.back().
                    first)){
                    P tmp=a.back();
                    a.pop_back();
                    work();
                    a.emplace_back(tmp);
                }else if(a.size()>=2&&a.end()[-2].first<=a.back
                    ().first){
                    work();
                }else break;
            }
        }
        while(a.size()>=2)work();
        return a[0];
    }
    P rake(int i){
        priority_queue<P,vector<P>,greater<P>> pq;
        for(int j:adj[i])if(j!=fa[i]&&j!=hv[i])pq.emplace(
            add_edge(j));
        while(pq.size()>=2){
            auto [si,i]=pq.top();pq.pop();
            auto [sj,j]=pq.top();pq.pop();
            pq.emplace(max(si,sj)+1,add(-1,i,j,Rake));
        }
        return pq.empty()?make_pair(0,-1):pq.top();
    }
    P add_edge(int i){
        auto [sj,j]=compress(i);
        return {sj+1,add(-1,j,-1,AddEdge)};
    }
    P add_vertex(int i){
        auto [sj,j]=rake(i);
        return {sj+1,add(i,j,-1,j==-1?Vertex:AddVertex)};
    }
};

/*
```

```
struct TreeDP{
    struct Path{
        static Path unit();
    };
    struct Point{
        static Point unit();
    };
    static Path compress(Path l,Path r);
    static Point rake(Point l,Point r);
    static Point add_edge(Path p);
    static Path add_vertex(Point p,int u);
    static Path vertex(int u);
};
*/

template<class G,class TreeDP>
struct StaticTopTreeRerootingDP{
    using Path = typename TreeDP::Path;
    using Point = typename TreeDP::Point;
    StaticTopTree<G> stt;
    vector<Path> path,rpath;
    vector<Point> point;
    StaticTopTreeRerootingDP(G &adj):stt(adj){
        int n=stt.n;
        path.resize(n);
        point.resize(n);
        rpath.resize(n);
        dfs(stt.root);
    }
    void _update(int u){
        if(stt.type[u]==stt.Vertex){
            path[u]=rpath[u]=TreeDP::vertex(u);
        }else if(stt.type[u]==stt.Compress){
            path[u]=TreeDP::compress(path[stt.lch[u]],path[stt.
                rch[u]]);
            rpath[u]=TreeDP::compress(rpath[stt.rch[u]],rpath[
                stt.lch[u]]);
        }else if(stt.type[u]==stt.Rake){
            point[u]=TreeDP::rake(point[stt.lch[u]],point[stt.
                rch[u]]);
        }else if(stt.type[u]==stt.AddEdge){
            point[u]=TreeDP::add_edge(path[stt.lch[u]]);
        }else{
            path[u]=rpath[u]=TreeDP::add_vertex(point[stt.lch[u
                ]],u);
        }
    }
    void dfs(int u){
        if(u==-1)return;
        dfs(stt.lch[u]);
        dfs(stt.rch[u]);
        _update(u);
    }
    void update(int u){
        for(;u!=-1;u=stt.par[u])_update(u);
    }
    Path query_all(){
        return path[stt.root];
    }
    Path query_subtree(int u){
        Path res=path[u];
        while(true){
            int p=stt.par[u];
            if(p==-1||stt.type[p]!=stt.Compress)break;
            if(stt.lch[p]==u)res=TreeDP::compress(path[stt.rch[
                p]],res);
        }
        return res;
    }
    Path query_reroot(int u){
        auto rec=[&](auto &&rec,int u)->Point {
            int p=stt.par[u];
            Path below=Path::unit(),above=Path::unit();
            while(p!=-1&&stt.type[p]==stt.Compress){
                int l=stt.lch[p],r=stt.rch[p];
                if(l==u)below=TreeDP::compress(below,path[r]);
                else above=TreeDP::compress(above,rpath[l]);
                u=p;
                p=stt.par[u];
            }
            if(p!=-1){
                u=p;
                p=stt.par[u];
                Point sum=Point::unit();
                while(stt.type[p]==stt.Rake){
                    int l=stt.lch[p],r=stt.rch[p];
                    sum=TreeDP::rake(sum,u==r?point[l]:point[r
                        ]);
                    u=p;
                    p=stt.par[u];
                }
                sum=TreeDP::rake(sum,rec(rec,p));
                above=TreeDP::compress(above,TreeDP::add_vertex
                    (sum,p));
            }
            return TreeDP::rake(TreeDP::add_edge(below),TreeDP
                ::add_edge(above));
        };
        Point res=rec(rec,u);
        if(stt.type[u]==stt.AddVertex){
            res=TreeDP::rake(res,point[stt.lch[u]]);
        }
        return TreeDP::add_vertex(res,u);
    }
};
```

Flows (5)

dinic.hpp

Description: Dinic's Algorithm for finding the maximum flow.
Time: $\mathcal{O}(VE\log U)$ where U is the maximum flow. 2b9ab1, 88 lines

```
template<class T,bool directed=true,bool scaling=true>
struct Dinic{
    static constexpr T INF=numeric_limits<T>::max()/2;
    struct Edge{
        int to;
        T flow,cap;
        Edge(int _to,T _cap):to(_to),flow(0),cap(_cap){}
        T remain(){return cap-flow;}
    };
    int n,s,t;
    T U;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> ptr,lv;
    bool calculated;
    T max_flow;
    Dinic(){}
    Dinic(int n,int s,int t){init(n,s,t);}
    void init(int _n,int _s,int _t){
        n=_n,s=_s,t=_t;
        U=0;
        e.clear();
        g.assign(n,{});
        calculated=false;
    }
    void add_edge(int from,int to,T cap){
        assert(0<=from&&from<n&&0<=to&&to<n);
        g[from].emplace_back(e.size());
        e.emplace_back(to,cap);
        g[to].emplace_back(e.size());
        e.emplace_back(from,directed?0:cap);
        U=max(U,cap);
    }
    bool bfs(T scale){
        lv.assign(n,-1);
        vector<int> q{s};
        lv[s]=0;
        for(int i=0;i<(int)q.size();i++){
            int u=q[i];
            for(int j:g[u]){
                int v=e[j].to;
                if(lv[v]==-1&&e[j].remain()>=scale){
                    q.emplace_back(v);
                    lv[v]=lv[u]+1;
                }
            }
        }
        return lv[t]!=-1;
    }
    T dfs(int u,int t,T f){
        if(u==t||f==0)return f;
        for(int &i=ptr[u];i<(int)g[u].size();i++){
            int j=g[u][i];
            int v=e[j].to;
            if(lv[v]==lv[u]+1){
                T res=dfs(v,t,min(f,e[j].remain()));
                if(res>0){
                    e[j].flow+=res;
                    e[j^1].flow-=res;
                    return res;
                }
            }
        }
        return 0;
    }
    T flow(){
        if(calculated)return max_flow;
        calculated=true;
        max_flow=0;
        for(T scale=scaling?1LL<<(63-__builtin_clzll(U)):1LL;
            scale>0;scale>>=1){
            while(bfs(scale)){
                ptr.assign(n,0);
                while(true){
                    T f=dfs(s,t,INF);
                    if(f==0)break;
                    max_flow+=f;
                }
            }
        }
        return max_flow;
    }
    pair<T,vector<int>> cut(){
        flow();
        vector<int> res(n);
        for(int i=0;i<n;i++)res[i]=(lv[i]==-1);
        return {max_flow,res};
    }
};
```

binary-optimization.hpp

Description: Binary Optimization. minimize $\kappa + \sum_i \theta_i(x_i) + \sum_{i<j} \phi_{ij}(x_i,x_j) + \sum_{i<j<k} \psi_{ijk}(x_i,x_j,x_k)$ where $x_i \in \{0,1\}$ and ϕ_{ij},ψ_{ijk} are submodular functions. a set function f is submodular if $f(S)+f(T) \geq f(S\cap T)+f(S\cup T)$ for all S,T. $\phi_{ij}(0,1)+\phi_{ij}(1,0) \geq \phi_{ij}(1,1)+\phi_{ij}(0,0)$.

"src/flows/dinic.hpp" 0d9c72, 95 lines

```
template<class T,bool minimize=true>
struct BinaryOptimization{
    static constexpr T INF=numeric_limits<T>::max()/2;
    int n,s,t,buf;
    T base;
    map<pair<int,int>,T> edges;
    BinaryOptimization(int _n):n(_n),s(n),t(n+1),buf(n+2),base
        (0){}
    void add_edge(int u,int v,T w){
        assert(w>=0);
        if(u==v||w==0)return;
        auto &e=edges[{u,v}];
        e=min(e+w,INF);
    }
    void add0(T w){
        base+=w;
    }
    void _add1(int i,T a,T b){
        if(a<=b){
            add0(a);
            add_edge(s,i,b-a);
        }else{
            add0(b);
            add_edge(i,t,a-b);
        }
    }
    void add1(int i,T x0,T x1){
        assert(0<=i&&i<n);
        if(!minimize)x0=-x0,x1=-x1;
        _add1(i,x0,x1);
    }
    void _add2(int i,int j,T a,T b,T c,T d){
        assert(b+c>=a+d);
        add0(a);
        _add1(i,0,c-a);
        _add1(j,0,d-c);
        add_edge(i,j,b+c-a-d);
    }
    void add2(int i,int j,T x00,T x01,T x10,T x11){
        assert(i!=j&&0<=i&&i<n&&0<=j&&j<n);
        if(!minimize)x00=-x00,x01=-x01,x10=-x10,x11=-x11;
        _add2(i,j,x00,x01,x10,x11);
    }
    void _add3(int i,int j,int k,T a,T b,T c,T d,T e,T f,T g,T
        h){
        T p=a+d+f+g-b-c-e-h;
        if(p>=0){
            add0(a);
            _add1(i,0,f-b);
            _add1(j,0,g-e);
            _add1(k,0,d-c);
            _add2(i,j,0,c+e-a-g,0,0);
            _add2(i,k,0,0,b+e-a-f,0);
            _add2(j,k,0,b+c-a-d,0,0);
            int u=buf++;
            add0(-p);
            add_edge(i,u,p);
            add_edge(j,u,p);
            add_edge(k,u,p);
            add_edge(u,t,p);
        }else{
            add0(h);
            _add1(i,c-g,0);
            _add1(j,b-d,0);
            _add1(k,e-f,0);
            _add2(i,j,0,0,d+f-b-h,0);
            _add2(i,k,0,0,d+g-c-h,0,0);
            _add2(j,k,0,0,f+g-e-h,0);
            int u=buf++;
            add0(p);
            add_edge(s,u,-p);
            add_edge(u,i,-p);
            add_edge(u,j,-p);
            add_edge(u,k,-p);
        }
    }
    void add3(int i,int j,int k,T x000,T x001,T x010,T x011,T
        x100,T x101,T x110,T x111){
        assert(i!=j&&j!=k&&k!=i&&0<=i&&i<n&&0<=j&&j<n&&0<=k&&k<
            n);
        if(!minimize){
            x000=-x000,x001=-x001,x010=-x010,x011=-x011;
            x100=-x100,x101=-x101,x110=-x110,x111=-x111;
        }
        _add3(i,j,k,x000,x001,x010,x011,x100,x101,x110,x111);
    }
    pair<T,vector<int>> solve(){
        Dinic<T> dinic(buf,s,t);
        for(auto &[p,w]:edges){
            auto [u,v]=p;
            dinic.add_edge(u,v,w);
        }
        auto [ans,cut]=dinic.cut();
        ans+=base;
        ans=min(ans,INF);
        cut.resize(n);
        return {minimize?ans:-ans,cut};
    }
};
```

k-ary-optimization.hpp

Description: k-ary Optimization. minimize $\kappa + \sum_i \theta_i(x_i) + \sum_{i<j} \phi_{ij}(x_i,x_j)$ where $x_i \in \{0,1,\ldots,k-1\}$ and $\phi_{i,j}$ is monge. A function f is monge if $f(a,c)+f(b,d) \leq f(a,d)+f(b,c)$ for all $a<b$ and $c<d$. $\phi_{ij}(x-1,y)+\phi_{ij}(x,y+1) \leq \phi_{ij}(x-1,y+1)+\phi_{ij}(x,y)$. $\phi_{ij}(x,y)+\phi_{ij}(x-1,y+1)-\phi_{ij}(x-1,y)-\phi_{ij}(x,y+1) \geq 0$.

"src/flows/dinic.hpp" 5139ce, 88 lines

```
template<class T,bool minimize=true>
struct K_aryOptimization{
    static constexpr T INF=numeric_limits<T>::max()/2;
    int n,s,t,buf;
    T base;
    vector<int> ks;
    vector<vector<int>> id;
    map<pair<int,int>,T> edges;
    K_aryOptimization(int n,int k){init(vector<int>(n,k));}
    K_aryOptimization(const vector<int> &_ks){init(_ks);}
    void init(const vector<int> &_ks){
        ks=_ks;
        n=ks.size();
        s=0,t=1,buf=2;
        base=0;
        id.clear();
        edges.clear();
        for(auto &k:ks){
            assert(k>=1);
            vector<int> a(k+1);
            a[0]=s,a[k]=t;
            for(int i=1;i<k;i++)a[i]=buf++;
            id.emplace_back(a);
            for(int i=2;i<k;i++)add_edge(a[i],a[i-1],INF);
        }
    }
    void add_edge(int u,int v,T w){
        assert(w>=0);
        if(u==v||w==0)return;
        auto &e=edges[{u,v}];
        e=min(e+w,INF);
    }
    void add0(T w){
        base+=w;
    }
    void _add1(int i,vector<T> cost){
        add0(cost[0]);
        for(int j=1;j<ks[i];j++){
            T x=cost[j]-cost[j-1];
            if(x>0)add_edge(id[i][j],t,x);
            if(x<0)add0(x),add_edge(s,id[i][j],-x);
        }
    }
    void add1(int i,vector<T> cost){
        assert(0<=i&&i<n&&(int)cost.size()==ks[i]);
        if(!minimize)for(auto &x:cost)x=-x;
        _add1(i,cost);
    }
    void _add2(int i,int j,vector<vector<T>> cost){
        int h=ks[i],w=ks[j];
        _add1(j,cost[0]);
        for(int x=h-1;x>=0;x--)for(int y=0;y<w;y++)cost[x][y]-=
            cost[0][y];
        vector<T> a(h);
        for(int x=0;x<h;x++)a[x]=cost[x][w-1];
        _add1(i,a);
        for(int x=0;x<h;x++)for(int y=0;y<w;y++)cost[x][y]-=a[x
            ];
        for(int x=1;x<h;x++){
            for(int y=0;y<w-1;y++){
                T w=cost[x][y]+cost[x-1][y+1]-cost[x-1][y]-cost
                    [x][y+1];
                assert(w>=0); // monge
                add_edge(id[i][x],id[j][y+1],w);
            }
        }
    }
    void add2(int i,int j,vector<vector<T>> cost){
        assert(0<=i&&i<n&&0<=j&&j<n&&i!=j);
        assert((int)cost.size()==ks[i]);
        for(auto &v:cost)assert((int)v.size()==ks[j]);
        if(!minimize)for(auto &v:cost)for(auto &x:v)x=-x;
        _add2(i,j,cost);
    }
    pair<T,vector<int>> solve(){
        Dinic<T> dinic(buf,s,t);
        for(auto &[p,w]:edges){
            auto [u,v]=p;
            dinic.add_edge(u,v,w);
        }
        auto [val,cut]=dinic.cut();
        val+=base;
        if(!minimize)val=-val;
        vector<int> ans(n);
        for(int i=0;i<n;i++){
            ans[i]=ks[i]-1;
            for(int j=1;j<ks[i];j++)ans[i]-=cut[id[i][j]];
        }
        return {val,ans};
    }
};
```

min-cost-flow.hpp

Description: minimum-cost flow algorithm.
Time: $\mathcal{O}(FE \log V)$ where F is max flow.

ca28ef, 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<vector<int>> adj;
    vector<C> pot,dist;
    vector<int> 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(e.size());
        e.emplace_back(v,cap,cost);
        adj[v].emplace_back(e.size());
        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<e.size();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};
    }
};
```

hopcroft-karp.hpp

Description: Fast bipartite matching algorithm.
Time: $\mathcal{O}\left(E\sqrt{V}\right)$

456024, 52 lines

```
struct HopcroftKarp{
    int n,m;
    vector<int> match,lv,ptr;
    vector<vector<int>> adj;
    HopcroftKarp(){}
    HopcroftKarp(int _n,int _m){init(_n,_m);}
    void init(int _n,int _m){
        n=_n,m=_m;
        adj.assign(n+m,vector<int>{});
    }
    void add_edge(int u,int v){
        adj[u].emplace_back(v+n);
    }
    void bfs(){
        lv.assign(n,-1);
        queue<int> q;
        for(int i=0;i<n;i++)if(match[i]==-1){
            lv[i]=0;
            q.emplace(i);
        }
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(int v:adj[u])if(match[v]!=-1&&lv[match[v]]==-1)
                {
                lv[match[v]]=lv[u]+1;
                q.emplace(match[v]);
            }
        }
    }
    bool dfs(int u){
        for(int &i=ptr[u];i<adj[u].size();i++){
            int v=adj[u][i];
            if(match[v]==-1||(lv[match[v]]==lv[u]+1&&dfs(match[
                v]))){
                match[u]=v,match[v]=u;
                return true;
            }
        }
        return false;
    }
    int max_matching(){
        int ans=0,cnt=0;
        match.assign(n+m,-1);
        do{
            ptr.assign(n,0);
            bfs();
            cnt=0;
            for(int i=0;i<n;i++)if(match[i]==-1&&dfs(i))cnt++;
            ans+=cnt;
        }while(cnt);
        return ans;
    }
};
```

String (6)

suffix-array.hpp

Description: Suffix Array.

58c0a5, 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)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);
    // }
};
```

suffix-automaton.hpp

Description: Suffix Automaton.

37a4fa, 47 lines

```
template<class STR>
struct SuffixAutomaton{
    using T = typename STR::value_type;
    struct Node{
        map<T,int> nxt;
        int link,len;
```

```
        Node(int link,int len):link(link),len(len){}
    };
    vector<Node> nodes;
    int last;
    SuffixAutomaton():nodes{Node(-1,0)},last(0){}
    SuffixAutomaton(const STR &s):SuffixAutomaton(){
        for(auto c:s)extend(c);
    }
    int new_node(int link,int len){
        nodes.emplace_back(Node(link,len));
        return (int)nodes.size()-1;
    }
    void extend(T c){
        int cur=new_node(0,nodes[last].len+1);
        int p=last;
        while(p!=-1&&!nodes[p].nxt.count(c)){
            nodes[p].nxt[c]=cur;
            p=nodes[p].link;
        }
        if(p!=-1){
            int q=nodes[p].nxt[c];
            if(nodes[p].len+1==nodes[q].len){
                nodes[cur].link=q;
            }else{
                int r=new_node(nodes[q].link,nodes[p].len+1);
                nodes[r].nxt=nodes[q].nxt;
                while(p!=-1&&nodes[p].nxt[c]==q){
                    nodes[p].nxt[c]=r;
                    p=nodes[p].link;
                }
                nodes[q].link=nodes[cur].link=r;
            }
        }
        last=cur;
    }
    ll distinct_substrings(){
        ll res=0;
        for(int i=1;i<(int)nodes.size();i++)res+=nodes[i].len-
            nodes[nodes[i].link].len;
        return res;
    }
};
```

aho-corasick.hpp

Description: Aho-Corasick. f2b759, 54 lines

```
template<class T>
struct AhoCorasick{
    struct Node{
        array<int,26> ch;
        int fail;
        T val;
        Node(){
            fill(ch.begin(),ch.end(),-1);
            fail=-1;
            val=0;
        }
    };
    vector<Node> nodes;
    AhoCorasick(){new_node();}
    int new_node(){
        nodes.emplace_back(Node());
        return nodes.size()-1;
    }
    void insert(const string &s,const T &val){
        int u=0;
        for(auto x:s){
            int c=x-'a';
            if(nodes[u].ch[c]==-1)nodes[u].ch[c]=new_node();
            u=nodes[u].ch[c];
        }
        nodes[u].val+=val;
    }
    void build(){
        vector<int> q{0};
        for(int i=0;i<q.size();i++){
            int u=q[i];
            int v;
            for(int c=0;c<26;c++){
                if((v=nodes[u].ch[c])!=-1){
                    int p=nodes[u].fail;
                    while(p!=-1&&nodes[p].ch[c]==-1)p=nodes[p].
                        fail;
                    p=p!=-1?nodes[p].ch[c]:0;
                    nodes[v].fail=p;
                    nodes[v].val+=nodes[p].val;
                    q.emplace_back(v);
                }
            }
        }
        for(auto u:q){
            for(int c=0;c<26;c++){
                if(nodes[u].ch[c]==-1){
                    int p=nodes[u].fail;
                    while(p!=-1&&nodes[p].ch[c]==-1)p=nodes[p].
                        fail;
                    nodes[u].ch[c]=p!=-1?nodes[p].ch[c]:0;
                }
            }
        }
    }
};
```

z-algorithm.hpp

Description: Z Algorithm. z[i] := the length of the longest common prefix between s and s[i:]. 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-l]);
        while(i+z[i]<n&&s[z[i]]==s[i+z[i]])z[i]++;
        if(i+z[i]>r)l=i,r=i+z[i];
    }
    return z;
}
```

prefix-function.hpp

Description: Prefix function. pi[i] := the length of the longest proper prefix of s[0:i] which is also a suffix of s[0:i]. 3d65fe, 11 lines

```
template<class STR>
vector<int> prefix_function(const STR &s){
    int n=(int)s.size();
    vector<int> pi(n);
    for(int i=1,j=0;i<n;i++){
        while(j>0&&s[i]!=s[j])j=pi[j-1];
        if(s[i]==s[j])j++;
        pi[i]=j;
    }
    return pi;
}
```

manacher.hpp

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

Convolutions (7)

and-convolution.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: $\mathcal{O}(N \log N)$. 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;){
        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;
}
```

or-convolution.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: $\mathcal{O}(N \log N)$. 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;
}
```

xor-convolution.hpp

Description: Bitwise XOR Convolution. Fast Walsh-Hadamard Transform: $A'[S] = \sum_T (-1)^{|S\&T|} A[T]$.
Time: $\mathcal{O}(N \log N)$.

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;
}
```

gcd-convolution.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: $\mathcal{O}(N \log \log N)$.

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;
}
```

lcm-convolution.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: $\mathcal{O}(N \log \log N)$.

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;
}
```

max-plus-convolution.hpp

Description: Max Plus Convolution. Find $C[k] = max_{i+j=k}\{A[i] + B[j]\}$.
Time: $\mathcal{O}(N)$.

7176a2, 94 lines

```
// SMAWCK algorithm for finding row-wise maxima.
// f(i,j,k) checks if M[i][j] <= M[i][k].
// f(i,j,k) checks if M[i][k] is at least as good as M[i][j].
// higher is better.
template<class F>
vector<int> smawck(const F &f,const vector<int> &rows,const
     vector<int> &cols){
    int n=(int)rows.size(),m=(int)cols.size();
    if(max(n,m)<=2){
        vector<int> ans(n,-1);
        for(int i=0;i<n;i++){
            for(int j:cols){
                if(ans[i]==-1||f(rows[i],ans[i],j)){
                    ans[i]=j;
                }
            }
        }
        return ans;
    }
    if(n<m){
        // reduce
        vector<int> st;
        for(int j:cols){
            while(true){
                if(st.empty()){
                    st.emplace_back(j);
                    break;
                }else if(f(rows[(int)st.size()-1],st.back(),j))
                    {
                    st.pop_back();
                }else if(st.size()<n){
                    st.emplace_back(j);
                    break;
                }else{
                    break;
                }
            }
        }
        return smawck(f,rows,st);
    }
    vector<int> ans(n,-1);
    vector<int> new_rows;
    for(int i=1;i<n;i+=2){
        new_rows.emplace_back(rows[i]);
    }
    auto res=smawck(f,new_rows,cols);
    for(int i=0;i<new_rows.size();i++){
        ans[2*i+1]=res[i];
    }
    for(int i=0,l=0,r=0;i<n;i+=2){
        if(i+1==n)r=m;
        while(r<m&&cols[r]<=ans[i+1])r++;
        ans[i]=cols[l++];
        for(;l<r;l++){
```

```
            if(f(rows[i],ans[i],cols[l])){
                ans[i]=cols[l];
            }
        }
        l--;
    }
    return ans;
}

template<class F>
vector<int> smawck(const F &f,int n,int m){
    vector<int> rows(n),cols(m);
    iota(rows.begin(),rows.end(),0);
    iota(cols.begin(),cols.end(),0);
    return smawck(f,rows,cols);
}

// Max Plus Convolution.
// b must be convex, i.e. b[i]-b[i-1]>=b[i+1]-b[i].
template<class T>
vector<T> max_plus_convolution_arbitary_convex(vector<T> a,
     const vector<T> &b){
    if(a.empty()||b.empty())return {};
    if((int)b.size()==1){
        for(auto &x:a)x+=b[0];
        return a;
    }
    int n=(int)a.size(),m=(int)b.size();
    auto f=[&](int i,int j){
        return a[j]+b[i-j];
    };
    auto cmp=[&](int i,int j,int k){
        if(i<k)return false;
        if(i-j>=m)return true;
        return f(i,j)<=f(i,k);
    };
    auto best=smawck(cmp,n+m-1,n);
    vector<T> ans(n+m-1);
    for(int i=0;i<n+m-1;i++){
        ans[i]=f(i,best[i]);
    }
    return ans;
}
```