Leones(0,0,0)

Contents

1 Data structures

1.1 Segment tree - Lazy

```cpp
struct LazySegtree{
  #define mid (l + r) / 2
  #define left(u) (u + 1)
  #define right(u) (u + ((mid - l + 1) << 1))
  struct Node{
    lli s, lazy;
    Node(lli s = 0, lli lazy = 0): s(s), lazy(lazy) {}
    Node operator + (const Node &n){
      return Node(s + n.s);
    }
  };
  vector<Node> st;
  LazySegtree(int n): st(2 * n) {}
  void push(int u, int l, int r){
    if(st[u].lazy){
      if(l < r){
        st[left(u)].lazy += st[u].lazy;
        st[right(u)].lazy += st[u].lazy;
      }
      st[u].s += st[u].lazy * (r - l + 1);
      st[u].lazy = 0;
    }
  }
  void update(int u, int l, int r, int ll, int rr, lli val){ // O(logN)
    push(u, l, r);
    if(l > r or r < ll or l > rr) return;
    if(ll <= l and r <= rr){
      st[u].lazy += val;
      push(u, l, r);
      return;
    }
    update(left(u), l, mid, ll, rr, val);
    update(right(u), mid + 1, r, ll, rr, val);
    st[u] = st[left(u)] + st[right(u)];
  }
  Node query(int u, int l, int r, int ll, int rr){ // O(logN)
    push(u, l, r);
    if(l > r or r < ll or l > rr) return Node();
    if(ll <= l and r <= rr) return st[u];
    return query(left(u), l, mid, ll, rr) + query(right(u), mid + 1, r, ll, rr);
  }
};
```

1.2 Segment tree - Persistence

```cpp
struct STree { // persistent segment tree for min over integers
  vi st, L, R; int n,tam,rt;
  STree(int n): st(1,NEUT),L(1,0),R(1,0),n(n),rt(0),tam(1){}
  int new_node(int v, int l=0, int r=0){
    int ks=sz(st); st.pb(v); L.pb(l); R.pb(r);
    return ks;
  }
  int upd(int k, int s, int e, int p, int v){
    int ks=new_node(st[k],L[k],R[k]);
    if(s+1==e){st[ks]=v;return ks;}
    int m=(s+e)/2,ps;
    if(p<m)ps=upd(L[ks],s,m,p,v),L[ks]=ps;
    else ps=upd(R[ks],m,e,p,v),R[ks]=ps;
    st[ks]=oper(st[L[ks]],st[R[ks]]);
    return ks;
  }
  int query(int k, int s, int e, int a, int b){
    if(e<=a||b<=s)return NEUT;
    if(a<=s&&e<=b)return st[k];
    int m=(s+e)/2;
    return oper(query(L[k],s,m,a,b),query(R[k],m,e,a,b));
  }
  int upd(int k, int p, int v){return rt=upd(k,0,n,p,v);}
  int upd(int p, int v){return upd(rt,p,v);} // update on last root
  int query(int k,int a, int b){return query(k,0,n,a,b);};
};
```

1.3 Segment tree - 2D

```cpp
int n,m; int a[MAXN][MAXN],st[2*MAXN][2*MAXN];
void build(){
  fore(i,0,n)fore(j,0,m)st[i+n][j+m]=a[i][j];
  fore(i,0,n)for(int j=m-1;j;--j)
    st[i+n][j]=op(st[i+n][j<<1],st[i+n][j<<1|1]);
  for(int i=n-1;i;--i)fore(j,0,2*m)
    st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
}
void upd(int x, int y, int v){
  st[x+n][y+m]=v;
  for(int j=y+m;j>1;j>>=1)st[x+n][j>>1]=op(st[x+n][j],st[x+n][j^1]);
  for(int i=x+n;i>1;i>>=1)for(int j=y+m;j;j>>=1)
    st[i>>1][j]=op(st[i][j],st[i^1][j]);
}
int query(int x0, int x1, int y0, int y1){
  int r=NEUT;
  for(int i0=x0+n,i1=x1+n;i0<i1;i0>>=1,i1>>=1){
    int t[4],q=0;
    if(i0&1)t[q++]=i0++;
    if(i1&1)t[q++]=--i1;
    fore(k,0,q)for(int j0=y0+m,j1=y1+m;j0<j1;j0>>=1,j1>>=1){
```

```
      if(j0&1)r=op(r,st[t[k]][j0++]);
      if(j1&1)r=op(r,st[t[k]][--j1]);
    }
  }
  return r;
}
```

1.4 Fenwick tree

```
int ft[MAXN+1]; // for more dimensions, make ft multi-dimensional
void upd(int i0, int v){ // add v to i0th element (0-based)
  // add extra fors for more dimensions
  for(int i=i0+1;i<=MAXN;i+=i&-i)ft[i]+=v;
}
int get(int i0){ // get sum of range [0,i0)
  int r=0;
  // add extra fors for more dimensions
  for(int i=i0;i;i-=i&-i)r+=ft[i];
  return r;
}
int get_sum(int i0, int i1){ // get sum of range [i0,i1) (0-based)
  return get(i1)-get(i0);
}
```

1.5 STL extended set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update>
    ordered_set;
// find_by_order(i) -> iterator to ith element
// order_of_key(k) -> position (int) of lower_bound of k
```

1.6 Treap (as BST)

```
typedef struct item *pitem;
struct item {
  int pr,key,cnt;
  pitem l,r;
  item(int key):key(key),pr(rand()),cnt(1),l(0),r(0) {}
};
int cnt(pitem t){return t?t->cnt:0;}
void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
void split(pitem t, int key, pitem& l, pitem& r){ // l: < key, r: >= key
  if(!t)l=r=0;
  else if(key<t->key)split(t->l,key,l,t->l),r=t;
  else split(t->r,key,t->r,r),l=t;
  upd_cnt(t);
}
void insert(pitem& t, pitem it){
  if(!t)t=it;
  else if(it->pr>t->pr)split(t,it->key,it->l,it->r),t=it;
  else insert(it->key<t->key?t->l:t->r,it);
  upd_cnt(t);
}
void merge(pitem& t, pitem l, pitem r){
  if(!l||!r)t=l?l:r;
  else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;
  upd_cnt(t);
}
void erase(pitem& t, int key){
  if(t->key==key)merge(t,t->l,t->r);
  else erase(key<t->key?t->l:t->r,key);
  upd_cnt(t);
}
void unite(pitem &t, pitem l, pitem r){
  if(!l||!r){t=l?l:r;return;}
  if(l->pr<r->pr)swap(l,r);
  pitem p1,p2;split(r,l->key,p1,p2);
  unite(l->l,l->l,p1);unite(l->r,l->r,p2);
  t=l;upd_cnt(t);
}
pitem kth(pitem t, int k){
  if(!t)return 0;
  if(k==cnt(t->l))return t;
  return k<cnt(t->l)?kth(t->l,k):kth(t->r,k-cnt(t->l)-1);
}
pair<int,int> lb(pitem t, int key){ // position and value of lower_bound
  if(!t)return {0,1<<30}; // (special value)
  if(key>t->key){
    auto w=lb(t->r,key);w.f+=cnt(t->l)+1;return w;
  }
  auto w=lb(t->l,key);
  if(w.f==cnt(t->l))w.s=t->key;
  return w;
}
```

1.7 Treap (implicit key)

```
// example that supports range reverse and addition updates, and range sum query
// (commented parts are specific to this problem)
typedef struct item *pitem;
struct item {
  int pr,cnt,val;
//  int sum; // (paramters for range query)
//  bool rev;int add; // (parameters for lazy prop)
  pitem l,r;
  item(int val): pr(rand()),cnt(1),val(val),l(0),r(0)/*,sum(val),rev(0),add(0)*/ {}
};
void push(pitem it){
  if(it){
    /*if(it->rev){
      swap(it->l,it->r);
      if(it->l)it->l->rev^=true;
      if(it->r)it->r->rev^=true;
      it->rev=false;
    }
    it->val+=it->add;it->sum+=it->cnt*it->add;
    if(it->l)it->l->add+=it->add;
```

```
    if(it->r)it->r->add+=it->add;
    it->add=0;*/
  }
}
int cnt(pitem t){return t?t->cnt:0;}
// int sum(pitem t){return t?push(t),t->sum:0;}
void upd_cnt(pitem t){
  if(t){
    t->cnt=cnt(t->l)+cnt(t->r)+1;
    // t->sum=t->val+sum(t->l)+sum(t->r);
  }
}
void merge(pitem& t, pitem l, pitem r){
  push(l);push(r);
  if(!l||!r)t=l?l:r;
  else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;
  upd_cnt(t);
}
void split(pitem t, pitem& l, pitem& r, int sz)
    { // sz:desired size of l
  if(!t){l=r=0;return;}
  push(t);
  if(sz<=cnt(t->l))split(t->l,l,t->l,sz),r=t;
  else split(t->r,t->r,r,sz-1-cnt(t->l)),l=t;
  upd_cnt(t);
}
void output(pitem t){ // useful for debugging
  if(!t)return;
  push(t);
  output(t->l);printf(" %d",t->val);output(t->r
      );
}
// use merge and split for range updates and
    queries
```

1.8 Link-Cut tree

```
const int N_DEL = 0, N_VAL = 0; //delta, value
inline int mOp(int x, int y){return x+y;}//
    modify
inline int qOp(int lval, int rval){return lval
    + rval;}//query
inline int dOnSeg(int d, int len){return d==
    N_DEL ? N_DEL : d*len;}
//mostly generic
inline int joinD(int d1, int d2){
  if(d1==N_DEL)return d2;if(d2==N_DEL)return d1
      ;return mOp(d1, d2);}
inline int joinVD(int v, int d){return d==N_DEL
     ? v : mOp(v, d);}
struct Node_t{
  int cnt, nVal, tVal, d;
  bool rev;
  Node_t *c[2], *p;
  Node_t(int v) : cnt(1), nVal(v), tVal(v), d(
      N_DEL), rev(0), p(0){
    c[0]=c[1]=0;
  }
  bool isRoot(){return !p || (p->c[0] != this
      && p->c[1] != this);}
  void push(){
    if(rev){
      rev=0; swap(c[0], c[1]);
      fore(x,0,2)if(c[x])c[x]->rev^=1;
    }
    nVal=joinVD(nVal, d); tVal=joinVD(tVal,
        dOnSeg(d, cnt));
    fore(x,0,2)if(c[x])c[x]->d=joinD(c[x]->d, d
        );
    d=N_DEL;
  }
  void upd();
};
typedef Node_t* Node;
int getSize(Node r){return r ? r->cnt : 0;}
int getPV(Node r){
  return r ? joinVD(r->tVal, dOnSeg(r->d,r->cnt
      )) : N_VAL;}
void Node_t::upd(){
  tVal = qOp(qOp(getPV(c[0]), joinVD(nVal, d)),
       getPV(c[1]));
  cnt = 1 + getSize(c[0]) + getSize(c[1]);
}
void conn(Node c, Node p, int il){if(c)c->p=p;
    if(il>=0)p->c[!il]=c;}
void rotate(Node x){
  Node p = x->p, g = p->p;
  bool gCh=p->isRoot(), isl = x==p->c[0];
  conn(x->c[isl],p,isl); conn(p,x,!isl);
  conn(x,g,gCh?-1:(p==g->c[0])); p->upd();
}
void spa(Node x){//splay
  while(!x->isRoot()){
    Node p = x->p, g = p->p;
    if(!p->isRoot())g->push();
    p->push(); x->push();
    if(!p->isRoot())rotate((x==p->c[0])==(p==g
        ->c[0])? p : x);
    rotate(x);
  }
  x->push(); x->upd();
}
Node exv(Node x){//expose
  Node last=0;
  for(Node y=x; y; y=y->p)spa(y),y->c[0]=last,y
      ->upd(),last=y;
  spa(x);
  return last;
}
void mkR(Node x){exv(x);x->rev^=1;}//makeRoot
Node getR(Node x){exv(x);while(x->c[1])x=x->c
    [1];spa(x);return x;}
Node lca(Node x, Node y){exv(x); return exv(y)
    ;}
bool connected(Node x, Node y){exv(x);exv(y);
    return x==y?1:x->p!=0;}
void link(Node x, Node y){mkR(x); x->p=y;}
void cut(Node x, Node y){mkR(x); exv(y); y->c
    [1]->p=0; y->c[1]=0;}
Node father(Node x){
  exv(x);
  Node r=x->c[1];
  if(!r)return 0;
  while(r->c[0])r=r->c[0];
  return r;
}
void cut(Node x){ // cuts x from father keeping
```

```
    tree root
  exv(father(x));x->p=0;}
int query(Node x, Node y){mkR(x); exv(y);
    return getPV(y);}
void modify(Node x, Node y, int d){mkR(x);exv(y
    );y->d=joinD(y->d,d);}
Node lift_rec(Node x, int t){
  if(!x)return 0;
  if(t==getSize(x->c[0])){spa(x);return x;}
  if(t<getSize(x->c[0]))return lift_rec(x->c
      [0],t);
  return lift_rec(x->c[1],t-getSize(x->c[0])-1)
      ;
}
Node lift(Node x, int t){ // t-th ancestor of x
    (lift(x,1) is x's father)
  exv(x);return lift_rec(x,t);}
int depth(Node x){ // distance from x to its
    tree root
  exv(x);return getSize(x)-1;}
```

1.9 Convex hull trick (static)

```
typedef lli tc;
struct Line{tc m,h;};
struct CHT { // for minimum (for maximum just
    change the sign of lines)
  vector<Line> c;
  int pos=0;
  tc in(Line a, Line b){
    tc x=b.h-a.h,y=a.m-b.m;
    return x/y+(x%y?!((x>0)^(y>0)):0); // ==
        ceil(x/y)
  }
  void add(tc m, tc h){ // m's should be non
      increasing
    Line l=(Line){m,h};
    if(c.size()&&m==c.back().m){
      l.h=min(h,c.back().h);c.pop_back();if(pos
          )pos--;
    }
    while(c.size()>1&&in(c.back(),l)<=in(c[c.
        size()-2],c.back())){
      c.pop_back();if(pos)pos--;
    }
    c.pb(l);
  }
  inline bool fbin(tc x, int m){return in(c[m],
      c[m+1])>x;}
  tc eval(tc x){
    // O(log n) query:
    int s=0,e=c.size();
    while(e-s>1){int m=(s+e)/2;
      if(fbin(x,m-1))e=m;
      else s=m;
    }
    return c[s].m*x+c[s].h;
    // O(1) query (for ordered x's):
    while(pos>0&&fbin(x,pos-1))pos--;
    while(pos<c.size()-1&&!fbin(x,pos))pos++;
    return c[pos].m*x+c[pos].h;
  }
};
```

1.10 Convex hull trick (dynamic)

```
typedef lli tc;
const tc is_query=-(1LL<<62); // special value
    for query
struct Line {
  tc m,b;
  mutable multiset<Line>::iterator it,end;
  const Line* succ(multiset<Line>::iterator it)
      const {
    return (++it==end? NULL : &*it);}
  bool operator<(const Line& rhs) const {
    if(rhs.b!=is_query)return m<rhs.m;
    const Line *s=succ(it);
    if(!s)return 0;
    return b-s->b<(s->m-m)*rhs.m;
  }
};
struct HullDynamic : public multiset<Line> { //
    for maximum
  bool bad(iterator y){
    iterator z=next(y);
    if(y==begin()){
      if(z==end())return false;
      return y->m==z->m&&y->b<=z->b;
    }
    iterator x=prev(y);
    if(z==end())return y->m==x->m&&y->b<=x->b;
    return (x->b-y->b)*(z->m-y->m)>=(y->b-z->b)
        *(y->m-x->m);
  }
  iterator next(iterator y){return ++y;}
  iterator prev(iterator y){return --y;}
  void add(tc m, tc b){
    iterator y=insert((Line){m,b});
    y->it=y;y->end=end();
    if(bad(y)){erase(y);return;}
    while(next(y)!=end()&&bad(next(y)))erase(
        next(y));
    while(y!=begin()&&bad(prev(y)))erase(prev(y
        ));
  }
  tc eval(tc x){
    Line l=*lower_bound((Line){x,is_query});
    return l.m*x+l.b;
  }
};
```

1.11 Disjoint intervals

```
// stores disjoint intervals as [first, second)
struct disjoint_intervals {
  set<pair<int,int> > s;
  void insert(pair<int,int> v){
    if(v.f>=v.s) return;
    auto at=s.lower_bound(v);auto it=at;
    if(at!=s.begin()&&(--at)->s>=v.f)v.f=at->f
        ,--it;
    for(;it!=s.end()&&it->f<=v.s;s.erase(it++))
      v.s=max(v.s,it->s);
    s.insert(v);
  }
};
```

2 Graphs

2.1 Bellman-Ford

```
int n;
vector<pair<int,int> > g[MAXN]; // u->[(v,cost)]
long long dist[MAXN];
void bford(int src){ // O(nm)
  fill(dist,dist+n,INF);dist[src]=0;
  for(int i = 0; i<n; i+=1)fore(x,0,n)if(dist[x]!=INF)for(auto t:g[x]){
    dist[t.f]=min(dist[t.f],dist[x]+t.s);
  }
  fore(x,0,n)if(dist[x]!=INF)for(auto t:g[x]){
    if(dist[t.f]>dist[x]+t.s){
      // neg cycle: all nodes reachable from t.fst have -INF distance
      // to reconstruct neg cycle: save "prev" of each node, go up from t.fst until repeating a node. this node and all nodes between the two occurences form a neg cycle
    }
  }
}
```

2.2 Strongly connected components (+ 2-SAT)

```
// MAXN: max number of nodes or 2 * max number of variables (2SAT)
bool truth[MAXN]; // truth[cmp[i]]=value of variable i (2SAT)
int nvar;int neg(int x){return MAXN-1-x;} // (2SAT)
vector<int> g[MAXN];
int n,lw[MAXN],idx[MAXN],qidx,cmp[MAXN],qcmp;
stack<int> st;
void tjn(int u){
  lw[u]=idx[u]=++qidx;
  st.push(u);cmp[u]=-2;
  for(int v:g[u]){
    if(!idx[v]||cmp[v]==-2){
      if(!idx[v]) tjn(v);
      lw[u]=min(lw[u],lw[v]);
    }
  }
  if(lw[u]==idx[u]){
    int x,l=-1;
    do{x=st.top();st.pop();cmp[x]=qcmp;if(min(x,neg(x))<nvar)l=x;}
    while(x!=u);
    if(l!=-1)truth[qcmp]=(cmp[neg(l)]<0); // (2SAT)
    qcmp++;
  }
}
void scc(){
  memset(idx,0,sizeof(idx));qidx=0;
  memset(cmp,-1,sizeof(cmp));qcmp=0;
  fore(i,0,n)if(!idx[i])tjn(i);
}
// Only for 2SAT:
void addor(int a, int b){g[neg(a)].pb(b);g[neg(b)].pb(a);}
bool satisf(int _nvar){
  nvar=_nvar;n=MAXN;scc();
  fore(i,0,nvar)if(cmp[i]==cmp[neg(i)])return false;
  return true;
}
```

2.3 Articulation - Bridges - Biconnected

```
vector<int> g[MAXN];int n;
struct edge {int u,v,comp;bool bridge;};
vector<edge> e;
void add_edge(int u, int v){
  g[u].pb(e.size());g[v].pb(e.size());
  e.pb((edge){u,v,-1,false});
}
int D[MAXN],B[MAXN],T;
int nbc;  // number of biconnected components
int art[MAXN];  // articulation point iff !=0
stack<int> st;  // only for biconnected
void dfs(int u,int pe){
  B[u]=D[u]=T++;
  for(int ne:g[u])if(ne!=pe){
    int v=e[ne].u^e[ne].v^u;
    if(D[v]<0){
      st.push(ne);dfs(v,ne);
      if(B[v]>D[u])e[ne].bridge = true; // bridge
      if(B[v]>=D[u]){
        art[u]++; // articulation
        int last; // start biconnected
        do {
          last=st.top();st.pop();
          e[last].comp=nbc;
        } while(last!=ne);
        nbc++;    // end biconnected
      }
      B[u]=min(B[u],B[v]);
    }
    else if(D[v]<D[u])st.push(ne),B[u]=min(B[u],D[v]);
  }
}
void doit(){
  memset(D,-1,sizeof(D));
    memset(art,0,sizeof(art));
  nbc=T=0;
  fore(i,0,n)if(D[i]<0)dfs(i,-1),art[i]--;
}
```

2.4 Heavy-Light decomposition

```
vector<int> g[MAXN];
int wg[MAXN],dad[MAXN],dep[MAXN]; // weight, father,depth
void dfs1(int x){
  wg[x]=1;
  for(int y:g[x])if(y!=dad[x]){
    dad[y]=x;dep[y]=dep[x]+1;
        dfs1(y);
    wg[x]+=wg[y];
  }
}
```

```
int curpos,pos[MAXN],head[MAXN];
void hld(int x, int c){
  if(c<0)c=x;
  pos[x]=curpos++;head[x]=c;
  int mx=-1;
  for(int y:g[x])if(y!=dad[x]&&(mx<0||wg[mx]<wg
      [y]))mx=y;
  if(mx>=0)hld(mx,c);
  for(int y:g[x])if(y!=mx&&y!=dad[x])hld(y,-1);
}
void hld_init(){dad[0]=-1;dep[0]=0;dfs1(0);
    curpos=0;hld(0,-1);}
int query(int x, int y, STree& rmq){
  int r=NEUT;
  while(head[x]!=head[y]){
    if(dep[head[x]]>dep[head[y]])swap(x,y);
    r=oper(r,rmq.query(pos[head[y]],pos[y]+1));
    y=dad[head[y]];
  }
  if(dep[x]>dep[y])swap(x,y); // now x is lca
  r=oper(r,rmq.query(pos[x],pos[y]+1));
  return r;
}
// for updating: rmq.upd(pos[x],v);
// queries on edges: - assign values of edges
    to "child" node
//                   - change pos[x] to pos[x
    ]+1 in query (line 28)
```

2.5 Centroid decomposition

```
vector<int> g[MAXN];int n;
bool tk[MAXN];
int fat[MAXN]; // father in centroid
    decomposition
int szt[MAXN]; // size of subtree
int calcsz(int x, int f){
  szt[x]=1;
  for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=
      calcsz(y,x);
  return szt[x];
}
void cdfs(int x=0, int f=-1, int sz=-1){ // O(
    nlogn)
  if(sz<0)sz=calcsz(x,-1);
  for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
    szt[x]=0;cdfs(y,f,sz);return;
  }
  tk[x]=true;fat[x]=f;
  for(auto y:g[x])if(!tk[y])cdfs(y,x);
}
void centroid(){memset(tk,false,sizeof(tk));
    cdfs();}
```

2.6 Parallel DFS

```
struct Tree {
  int n,z[2];
  vector<vector<int>> g;
  vector<int> ex,ey,p,w,f,v[2];
  Tree(int n):g(n),w(n),f(n){}
  void add_edge(int x, int y){
    p.pb(g[x].size());g[x].pb(ex.size());ex.pb(
        x);ey.pb(y);
    p.pb(g[y].size());g[y].pb(ex.size());ex.pb(
        y);ey.pb(x);
  }
  bool go(int k){ // returns true if it finds
      new node
    int& x=z[k];
    while(x>=0&&
      (w[x]==g[x].size()||w[x]==g[x].size()
          -1&&(g[x].back()^1)==f[x]))
      x=f[x]>=0?ex[f[x]]:-1;
    if(x<0)return false;
    if((g[x][w[x]]^1)==f[x])w[x]++;
    int e=g[x][w[x]],y=ey[e];
    f[y]=e;w[x]++;w[y]=0;x=y;
    v[k].pb(x);
    return true;
  }
  vector<int> erase_edge(int e){
    e*=2; // erases eth edge, returns smaller
        component
    int x=ex[e],y=ey[e];
    p[g[x].back()]=p[e];
    g[x][p[e]]=g[x].back();g[x].pop_back();
    p[g[y].back()]=p[e^1];
    g[y][p[e^1]]=g[y].back();g[y].pop_back();
    f[x]=f[y]=-1;
    w[x]=w[y]=0;
    z[0]=x;z[1]=y;
    v[0]={x};v[1]={y};
    bool d0=true,d1=true;
    while(d0 and d1)d0=go(0),d1=go(1);
    if(d1)return v[0];
    return v[1];
  }
};
```

2.7 Eulerian path

```
// Directed version (uncomment commented code
    for undirected)
struct edge {
  int y;
//  list<edge>::iterator rev;
  edge(int y):y(y){}
};
list<edge> g[MAXN];
void add_edge(int a, int b){
  g[a].push_front(edge(b));//auto ia=g[a].begin
      ();
//  g[b].push_front(edge(a));auto ib=g[b].begin
    ();
//  ia->rev=ib;ib->rev=ia;
}
vector<int> p;
void go(int x){
  while(g[x].size()){
    int y=g[x].front().y;
    //g[y].erase(g[x].front().rev);
    g[x].pop_front();
    go(y);
  }
  p.push_back(x);
}
vector<int> get_path(int x){ // get a path that
```

```
    begins in x
// check that a path exists from x before
    calling to get_path!
  p.clear();go(x);
    reverse(p.begin(),p.end());
  return p;
}
```

2.8 Dynamic connectivity

```
struct UnionFind {
  int n,comp;
  vector<int> uf,si,c;
  UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1)
      {
    fore(i,0,n)uf[i]=i;}
  int find(int x){return x==uf[x]?x:find(uf[x])
      ;}
  bool join(int x, int y){
    if((x=find(x))==(y=find(y)))return false;
    if(si[x]<si[y])swap(x,y);
    si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
    return true;
  }
  int snap(){return c.size();}
  void rollback(int snap){
    while(c.size()>snap){
      int x=c.back();c.pop_back();
      si[uf[x]]-=si[x];uf[x]=x;comp++;
    }
  }
};
enum {ADD,DEL,QUERY};
struct Query {int type,x,y;};
struct DynCon {
  vector<Query> q;
  UnionFind dsu;
  vector<int> mt;
  map<pair<int,int>,int> last;
  DynCon(int n):dsu(n){}
  void add(int x, int y){
    if(x>y)swap(x,y);
    q.pb((Query){ADD,x,y});mt.pb(-1);
      last[{x,y}]=q.size()-1;
  }
  void remove(int x, int y){
    if(x>y)swap(x,y);
    q.pb((Query){DEL,x,y});
    int pr=last[{x,y}];mt[pr]=q.size()-1;mt.pb(
      pr);
  }
  void query(){q.pb((Query){QUERY,-1,-1});
    mt.pb(-1);}
  void process(){ // answers all queries in
     order
    if(!q.size())return;
    fore(i,0,q.size())if(q[i].type==ADD&&mt[i
      ]<0)mt[i]=q.size();
    go(0,q.size());
  }
  void go(int s, int e){
    if(s+1==e){
      if(q[s].type==QUERY) // answer query
        using DSU
          cout<<dsu.comp<<ENDL;
      return;
    }
    int k=dsu.snap(),m=(s+e)/2;
    for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s
      )dsu.join(q[i].x,q[i].y);
    go(s,m);
      dsu.rollback(k);
    for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join
      (q[i].x,q[i].y);
    go(m,e);
      dsu.rollback(k);
  }
};
```

2.9 Dominator tree

```
//idom[i]=parent of i in dominator tree with
    root=rt, or -1 if not exists
int n,rnk[MAXN],pre[MAXN],anc[MAXN],idom[MAXN],
    semi[MAXN],low[MAXN];
vector<int> g[MAXN],rev[MAXN],dom[MAXN],ord;
void dfspre(int pos){
  rnk[pos]=sz(ord); ord.pb(pos);
  for(auto x:g[pos]){
    rev[x].pb(pos);
    if(rnk[x]==n) pre[x]=pos,dfspre(x);
  }
}
int eval(int v){
  if(anc[v]<n and anc[anc[v]]<n){
    int x=eval(anc[v]);
    if(rnk[semi[low[v]]]>rnk[semi[x]]) low[v]=x
      ;
    anc[v]=anc[anc[v]];
  }
  return low[v];
}
void dominators(int rt){
  fore(i,0,n){
    dom[i].clear(); rev[i].clear();
    rnk[i]=pre[i]=anc[i]=idom[i]=n;
    semi[i]=low[i]=i;
  }
  ord.clear();
    dfspre(rt);
  for(int i=sz(ord)-1;i;i--){
    int w=ord[i];
    for(int v:rev[w]){
      int u=eval(v);
      if(rnk[semi[w]]>rnk[semi[u]])semi[w]=semi
        [u];
    }
    dom[semi[w]].pb(w); anc[w]=pre[w];
    for(int v:dom[pre[w]]){
      int u=eval(v);
      idom[v]=(rnk[pre[w]]>rnk[semi[u]]?u:pre[w
        ]);
    }
    dom[pre[w]].clear();
  }
  for(int w:ord) if(w!=rt&&idom[w]!=semi[w])
      idom[w]=idom[idom[w]];
  fore(i,0,n) if(idom[i]==n)idom[i]=-1;
```

```
}
```

3 Math

3.1 Catalan

```
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan
                [i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}
```

3.2 Pollard's rho

```
long long gcd(long long a, long long b){return
    a?gcd(b%a,a):b;}
long long mulmod(long long a, long long b, long
     long m) {
  long long r=a*b-(long long)((long double)a*b/
      m+.5)*m;
  return r<0?r+m:r;
}
long long expmod(long long b, long long e, long
     long m){
  if(!e)return 1;
  long long q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return e&1?mulmod(b,q,m):q;
}
bool is_prime_prob(long long n, long long a){
  if(n==a)return true;
  long long s=0,d=n-1;
  while(d%2==0)s++,d/=2;
  long long x=expmod(a,d,n);
  if((x==1)||(x+1==n))return true;
  for(int tt = 0; tt < s - 1; ++tt){
    x=mulmod(x,x,n);
    if(x==1)return false;
    if(x+1==n)return true;
  }
  return false;
}
bool rabin(long long n){ // true iff n is prime
  if(n==1)return false;
  int ar[]={2,3,5,7,11,13,17,19,23};
  for(int i = 0; i < 9; ++i)if(!is_prime_prob(n
      ,ar[i]))return false;
  return true;
}
long long rho(long long n){
  if(!(n&1))return 2;
  long long x=2,y=2,d=1;
  long long c=rand()%n+1;
  while(d==1){
    x=(mulmod(x,x,n)+c)%n;
    y=(mulmod(y,y,n)+c)%n;
    y=(mulmod(y,y,n)+c)%n;
    if(x>=y)d=gcd(x-y,n);
    else d=gcd(y-x,n);
  }
  return d==n?rho(n):d;
}
void fact(long long n, map<long long,int>& f){
    //O (lg n)^3
  if(n==1)return;
  if(rabin(n)){f[n]++;return;}
  long long q=rho(n);fact(q,f);fact(n/q,f);
}
// optimized version: replace rho and fact with
     the following:
const int MAXP=1e6+1; // sieve size
int sv[MAXP]; // sieve
long long add(long long a, long long b, long
    long m){return (a+=b)<m?a:a-m;}
long long rho(long long n){
  static long long s[MAXP];
  while(1){
    long long x=rand()%n,y=x,c=rand()%n;
    long long *px=s,*py=s,v=0,p=1;
    while(1){
      *py++=y=add(mulmod(y,y,n),c,n);
      *py++=y=add(mulmod(y,y,n),c,n);
      if((x=*px++)==y)break;
      long long t=p;
      p=mulmod(p,abs(y-x),n);
      if(!p)return gcd(t,n);
      if(++v==26){
        if((p=gcd(p,n))>1&&p<n)return p;
        v=0;
      }
    }
    if(v&&(p=gcd(p,n))>1&&p<n)return p;
  }
}
void init_sv(){
  for(int i = 2; i < MAXP; ++i)if(!sv[i])for(
      long long j=i;j<MAXP;j+=i)sv[j]=i;
}
void fact(long long n, map<long long,int>& f){
    // call init_sv first!!!
  for(auto&& p:f){
    while(n%p.first==0){
      p.second++;
      n/=p.first;
    }
  }
  if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
  else if(rabin(n))f[n]++;
  else {long long q=rho(n);fact(q,f);fact(n/q,f
      );}
}
```

3.3 Simpson's rule

```
double integrate(double f(double), double a,
    double b, int n=10000){
  double r=0,h=(b-a)/n,fa=f(a),fb;
  for(int i = 0; i < n; ++i){fb=f(a+h*(i+1));r
      +=fa+4*f(a+h*(i+0.5))+fb;fa=fb;}
  return r*h/6.;
```

```
}
```

3.4 Polynomials

```
///Needs a EPS
typedef int tp; // type of polynomial
template<class T=tp>
struct poly {  // poly<> : 1 variable, poly<poly<>>: 2 variables, etc.
  vector<T> c;
  T& operator[](int k){return c[k];}
  poly(vector<T>& c):c(c){}
  poly(initializer_list<T> c):c(c){}
  poly(int k):c(k){}
  poly(){}
  poly operator+(poly<T> o){
    int m=c.size(),n=o.c.size();
    poly res(max(m,n));
    for(int i = 0; i < m; ++i)res[i]=res[i]+c[i];
    for(int i = 0; i < n; ++i)res[i]=res[i]+o.c[i];
    return res;
  }
  poly operator*(tp k){
    poly res(c.size());
    for(int i = 0; i < (int)c.size(); ++i)res[i]=c[i]*k;
    return res;
  }
  poly operator*(poly o){
    int m=c.size(),n=o.c.size();
    poly res(m+n-1);
    for(int i = 0; i < m; ++i) for(int j = 0; j < n; ++j)res[i+j]=res[i+j]+c[i]*o.c[j];
    return res;
  }
  poly operator-(poly<T> o){return *this+(o*-1);}
  T operator()(tp v){
    T sum(0);
    for(int i=(int)c.size()-1;i>=0;--i)sum=sum*v+c[i];
    return sum;
  }
};
// example: p(x,y)=2*x^2+3*x*y-y+4
// poly<poly<>> p={{4,-1},{0,3},{2}}
// printf("%d\n",p(2)(3)) // 27 (p(2,3))
set<tp> roots(poly<> p){ // only for integer polynomials
  set<tp> r;
  while(!p.c.empty()&&!p.c.back())p.c.pop_back();
  if(!p(0))r.insert(0);
  if(p.c.empty())return r;
  tp a0=0,an=abs(p[p.c.size()-1]);
  for(int k=0;!a0;a0=abs(p[k++]));
  vector<tp> ps,qs;
  for(int i = 1; i < sqrt(a0)+1; ++i)if(a0%i==0)ps.push_back(i),ps.push_back(a0/i);
  for(int i = 1; i < sqrt(an)+1; ++i)if(an%i==0)qs.push_back(i),qs.push_back(an/i);
  for(auto pt:ps)for(auto qt:qs)if(pt%qt==0){
    tp x=pt/qt;
    if(!p(x))r.insert(x);
    if(!p(-x))r.insert(-x);
  }
  return r;
}
pair<poly<>,tp> ruffini(poly<> p, tp r){ // returns pair (result,rem)
  int n=p.c.size()-1;
  vector<tp> b(n);
  b[n-1]=p[n];
  for(int k=n-2;k>=0;--k)b[k]=p[k+1]+r*b[k+1];
  return {poly<>(b),p[0]+r*b[0]};
}
// only for double polynomials
pair<poly<>,poly<> > polydiv(poly<> p, poly<> q){ // returns pair (result,rem)
  int n=p.c.size()-q.c.size()+1;
  vector<tp> b(n);
  for(int k=n-1;k>=0;--k){
    b[k]=p.c.back()/q.c.back();
    for(int i = 0; i < (int)q.c.size(); ++i)p[i+k]-=b[k]*q[i];
    p.c.pop_back();
  }
  while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();
  return {poly<>(b),p};
}
// only for double polynomials
poly<> interpolate(vector<tp> x, vector<tp> y){ //TODO TEST
  poly<> q={1},S={0};
  for(tp a:x)q=poly<>({-a,1})*q;
  for(int i = 0; i < (int)x.size(); ++i){
    poly<> Li=ruffini(q,x[i]).first;
    Li=Li*(1.0/Li(x[i])); // change for int polynomials
    S=S+Li*y[i];
  }
  return S;
}
```

3.5 Fast Fourier Transform

```
// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
typedef long long tf;
typedef vector<tf> poly;
const tf MOD=2305843009255636993,RT=5;
// FFT
struct CD {
  double r,i;
  CD(double r=0, double i=0):r(r),i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
CD operator*(const CD& a, const CD& b){
  return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
CD operator-(const CD& a, const CD& b){return
```

```
    CD(a.r-b.r,a.i-b.i);}
const double pi=acos(-1.0);
// NTT
/*
struct CD {
  tf x;
  CD(tf x):x(x){}
  CD(){}
};
CD operator*(const CD& a, const CD& b){return
    CD(mulmod(a.x,b.x));}
CD operator+(const CD& a, const CD& b){return
    CD(addmod(a.x,b.x));}
CD operator-(const CD& a, const CD& b){return
    CD(submod(a.x,b.x));}
vector<tf> rts(MAXN+9,-1);
CD root(int n, bool inv){
  tf r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
  return CD(inv?pm(r,MOD-2):r);
}
*/
const int MAXN = 1;
CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  for(int i = 0; i < n; ++i) if(R[i]<i)swap(a[R
      [i]],a[i]);
  for(int m=2;m<=n;m*=2){
    double z=2*pi/m*(inv?-1:1); // FFT
    CD wi=CD(cos(z),sin(z)); // FFT
    // CD wi=root(m,inv); // NTT
    for(int j=0;j<n;j+=m){
      CD w(1);
      for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
        CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u
            -v;w=w*wi;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) a[i]/=n;
      // FFT
  //if(inv){ // NTT
  //  CD z(pm(n,MOD-2)); // pm: modular
      exponentiation
  //  fore(i,0,n)a[i]=a[i]*z;
  //}
}
poly multiply(poly& p1, poly& p2){
  int n=(int)p1.size()+(int)p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  for(int i = 0; i < m; ++i){R[i]=0;for(int j =
       0; j < cnt; ++j)R[i]=(R[i]<<1)|((i>>j)
      &1);}
  for(int i = 0; i < m; ++i)cp1[i]=0,cp2[i]=0;
  for(int i = 0; i < (int)(p1.size()); ++i)cp1[
      i]=p1[i];
  for(int i = 0; i < (int)(p2.size()); ++i)cp2[
      i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  for(int i = 0; i < m; ++i)cp1[i]=cp1[i]*cp2[i
      ];
  dft(cp1,m,true);
  poly res;
  n-=2;
  for(int i = 0; i < n; ++i)res.push_back((tf)
      floor(cp1[i].real()+0.5)); // FFT
  //fore(i,0,n)res.pb(cp1[i].x); // NTT
  return res;
}
```

3.6 Fast Fourier Transform Operations

```
long long powm(long long a, long long b, long
    long mod){
  long long res =1;
  while(b){ if(b&1) res = (res * a) % mod; a =
      (a*a) % mod; b/=2; }
  return res;
}
long long inv(long long a, long long mod) {
  return powm(a, mod - 2, mod);
}
// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
typedef long long tf;
typedef vector<tf> poly;
const tf MOD=2305843009255636993,RT=5;
// FFT
struct CD {
  double r,i;
  CD(double r=0, double i=0):r(r),i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
CD operator*(const CD& a, const CD& b){
  return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
CD operator+(const CD& a, const CD& b){return
    CD(a.r+b.r,a.i+b.i);}
CD operator-(const CD& a, const CD& b){return
    CD(a.r-b.r,a.i-b.i);}
const double pi=acos(-1.0);
// NTT
/*
struct CD {
  tf x;
  CD(tf x):x(x){}
  CD(){}
};
CD operator*(const CD& a, const CD& b){return
    CD(mulmod(a.x,b.x));}
CD operator+(const CD& a, const CD& b){return
    CD(addmod(a.x,b.x));}
CD operator-(const CD& a, const CD& b){return
    CD(submod(a.x,b.x));}
vector<tf> rts(MAXN+9,-1);
CD root(int n, bool inv){
  tf r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
  return CD(inv?pm(r,MOD-2):r);
}
*/
const int MAXN = 1;
CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  for(int i = 0; i < n; ++i) if(R[i]<i)swap(a[R
      [i]],a[i]);
```

```
  for(int m=2;m<=n;m*=2){
    double z=2*pi/m*(inv?-1:1); // FFT
    CD wi=CD(cos(z),sin(z)); // FFT
    // CD wi=root(m,inv); // NTT
    for(int j=0;j<n;j+=m){
      CD w(1);
      for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
        CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u
            -v;w=w*wi;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) a[i]/=n;
      // FFT
  //if(inv){ // NTT
  //  CD z(pm(n,MOD-2)); // pm: modular
      exponentiation
  //  fore(i,0,n)a[i]=a[i]*z;
  //}
}
poly multiply(poly& p1, poly& p2){
  int n=(int)p1.size()+(int)p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  for(int i = 0; i < m; ++i){R[i]=0;for(int j =
       0; j < cnt; ++j)R[i]=(R[i]<<1)|((i>>j)
      &1);}
  for(int i = 0; i < m; ++i)cp1[i]=0,cp2[i]=0;
  for(int i = 0; i < (int)(p1.size()); ++i)cp1[
      i]=p1[i];
  for(int i = 0; i < (int)(p2.size()); ++i)cp2[
      i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  for(int i = 0; i < m; ++i)cp1[i]=cp1[i]*cp2[i
      ];
  dft(cp1,m,true);
  poly res;
  n-=2;
  for(int i = 0; i < n; ++i)res.push_back((tf)
      floor(cp1[i].real()+0.5)); // FFT
  //fore(i,0,n)res.pb(cp1[i].x); // NTT
  return res;
}

//Polynomial division: O(n*log(n))
//Multi-point polynomial evaluation: O(n*log^2(
    n))
//Polynomial interpolation: O(n*log^2(n))
long long addmod(long long a, long long b) {
  if(a + b > MOD) return (a + b) - MOD;
  return a + b;
}
long long submod(long long a, long long b) {
  if(a - b < 0) return (a - b) + MOD;
  return a - b;
}
long long mulmod(long long a, long long b) {
  return (a * b) % MOD;
}
//Works with NTT. For FFT, just replace addmod,
    submod,mulmod,inv
poly add(poly &a, poly &b){
  int n=(int)a.size(),m=(int)b.size();
  poly ans(max(n,m));
  for(int i = 0; i < max(n, m); ++i){
    if(i<n) ans[i]=addmod(ans[i],a[i]);
    if(i<m) ans[i]=addmod(ans[i],b[i]);
  }
  while((int)(ans.size())>1&&!ans.back())ans.
      pop_back();
  return ans;
}

poly invert(poly &b, int d){
 poly c = {inv(b[0], MOD)};
 while((int)(c.size())<=d){
  int j=2*(int)(c.size());
  auto bb=b; bb.resize(j);
  poly cb=multiply(c,bb);
  for(int i = 0; i < (int)(cb.size()); ++i) cb[
      i]=submod(0,cb[i]);
  cb[0]=addmod(cb[0],2);
  c=multiply(c,cb);
  c.resize(j);
 }
 c.resize(d+1);
 return c;
}

pair<poly,poly> divslow(poly &a, poly &b){
  poly q,r=a;
  while((int)(r.size())>=(int)(b.size())){
    q.push_back(mulmod(r.back(),inv(b.back(),
        MOD)));
    if(q.back()) for(int i = 0; i < (int)(b.
        size()); ++i){
      r[(int)(r.size())-i-1]=submod(r[(int)(r.
          size())-i-1],mulmod(q.back(),b[(int)(
          b.size())-i-1]));
    }
    r.pop_back();
  }
  reverse(q.begin(), q.end());
  return {q,r};
}

pair<poly,poly> divide(poly &a, poly &b){ //
    returns {quotient,remainder}
  int m=(int)(a.size()),n=(int)(b.size()),MAGIC
      =750;
  if(m<n) return {{0},a};
  if(min(m-n,n)<MAGIC)return divslow(a,b);
  poly ap=a; reverse(ap.begin(), ap.end());
  poly bp=b; reverse(bp.begin(), bp.end());
  bp=invert(bp,m-n);
  poly q=multiply(ap,bp);
  q.resize((int)(q.size())+m-n-(int)(q.size())
      +1,0);
  reverse(q.begin(), q.end());
  poly bq=multiply(b,q);
  for(int i = 0; i < (int)(bq.size()); ++i) bq[
      i]=submod(0,bq[i]);
  poly r=add(a,bq);
  return {q,r};
}

vector<poly> tree;

```

```
void filltree(vector<tf> &x){
  int k=(int)(x.size());
  tree.resize(2*k);
  for(int i = k; i < 2 * k; ++i)  tree[i]={
      submod(0,x[i-k]),1};
  for(int i=k-1;i;i--) tree[i]=multiply(tree[2*
      i],tree[2*i+1]);
}

vector<tf> evaluate(poly &a, vector<tf> &x){
  filltree(x);
  int k=(int)(x.size());
  vector<poly> ans(2*k);
  ans[1]=divide(a,tree[1]).second;
  for(int i = 2; i < 2 * k; ++i) ans[i]=divide(
      ans[i>>1],tree[i]).second;
  vector<tf> r; for(int i = 0; i < k; ++i) r.
      push_back(ans[i+k][0]);
  return r;
}

poly derivate(poly &p){
  poly ans((int)(p.size())-1);
  for(int i = 1; i < (int)(p.size()); ++i) ans[
      i-1]=mulmod(p[i],i);
  return ans;
}

poly interpolate(vector<tf> &x, vector<tf> &y){
  filltree(x);
  poly p=derivate(tree[1]);
  int k=(int)(y.size());
  vector<tf> d=evaluate(p,x);
  vector<poly> intree(2*k);
  for(int i = k; i < 2 * k; ++i) intree[i]={
      mulmod(y[i-k],inv(d[i-k], MOD))};
  for(int i=k-1;i;i--){
    poly p1=multiply(tree[2*i],intree[2*i+1]);
    poly p2=multiply(tree[2*i+1],intree[2*i]);
    intree[i]=add(p1,p2);
  }
  return intree[1];
}
```

3.7 Fast Hadamard Transform

```
const int MAXN = 1;
long long c1[MAXN+9],c2[MAXN+9];  // MAXN must
    be power of 2 !!
void fht(long long* p, int n, bool inv){
  for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*
      l)for(int j = 0; j < l; ++j){
    long long u=p[i+j],v=p[i+l+j];
    if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
    else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
    //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
    //else p[i+j]=-u+v,p[i+l+j]=u;
    //if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
    //else p[i+j]=v,p[i+l+j]=u-v;
  }
}
// like polynomial multiplication, but XORing
    exponents
// instead of adding them (also ANDing, ORing)
vector<long long> multiply(vector<long long>&
    p1, vector<long long>& p2){
  int n=1<<(32-__builtin_clz(max((int)(p1.size
      ()),(int)(p2.size())-1)));
  for(int i = 0; i < n; ++i)c1[i]=0,c2[i]=0;
  for(int i = 0; i < (int)(p1.size()); ++i) c1[
      i]=p1[i];
  for(int i = 0; i < (int)(p2.size()); ++i) c2[
      i]=p2[i];
  fht(c1,n,false);fht(c2,n,false);
  for(int i = 0; i < n; ++i) c1[i]*=c2[i];
  fht(c1,n,true);
  return vector<long long>(c1,c1+n);
}
```

3.8 Karatsuba

```
typedef long long tp;
#define add(n,s,d,k) for(int i = 0; i < n; ++i)
    (d)[i]+=(s)[i]*k
tp* ini(int n){tp *r=new tp[n];fill(r,r+n,0);
    return r;}
void karatsura(int n, tp* p, tp* q, tp* r){
  if(n<=0)return;
  if(n<35)for(int i = 0; i < n; ++i)for(int j =
      0; j < n; ++j)r[i+j]+=p[i]*q[j];
  else {
    int nac=n/2,nbd=n-n/2;
    tp *a=p,*b=p+nac,*c=q,*d=q+nac;
    tp *ab=ini(nbd+1),*cd=ini(nbd+1),*ac=ini(
        nac*2),*bd=ini(nbd*2);
    add(nac,a,ab,1);add(nbd,b,ab,1);
    add(nac,c,cd,1);add(nbd,d,cd,1);
    karatsura(nac,a,c,ac);karatsura(nbd,b,d,bd)
        ;
    add(nac*2,ac,r+nac,-1);add(nbd*2,bd,r+nac
        ,-1);
    add(nac*2,ac,r,1);add(nbd*2,bd,r+nac*2,1);
    karatsura(nbd+1,ab,cd,r+nac);
    free(ab);free(cd);free(ac);free(bd);
  }
}
vector<tp> multiply(vector<tp> p0, vector<tp>
    p1){
  int n=max(p0.size(),p1.size());
  tp *p=ini(n),*q=ini(n),*r=ini(2*n);
  for(int i = 0; i < (int)(p0.size()); ++i) p[i
      ]=p0[i];
  for(int i = 0; i < (int)(p1.size()); ++i) q[i
      ]=p1[i];
  karatsura(n,p,q,r);
  vector<tp> rr(r,r+p0.size()+p1.size()-1);
  free(p);free(q);free(r);
  return rr;
}
```

3.9 Diophantine

```
//Need gcd
pair<long long,long long> extendedEuclid (long
    long a, long long b){ //a * x + b * y = gcd
    (a,b)
  long long x,y;
  if (b==0) return {1,0};
  auto p=extendedEuclid(b,a%b);
```

```
  x=p.second;
  y=p.first-(a/b)*x;
  if(a*x+b*y==-gcd(a,b)) x=-x, y=-y;
  return {x,y};
}
pair<pair<long long,long long>,pair<long long,
    long long> > diophantine(long long a,long
    long b, long long r) {
  //a*x+b*y=r where r is multiple of gcd(a,b);
  long long d=gcd(a,b);
  a/=d; b/=d; r/=d;
  auto p = extendedEuclid(a,b);
  p.first*=r; p.second*=r;
  assert(a*p.first+b*p.second==r);
  return {p,{-b,a}}; // solutions: p+t*ans.snd
}
```

3.10 Chinese remainder theorem

```
//Needs gcd

pair<long long,long long> extendedEuclid (long
    long a, long long b){ //a * x + b * y = gcd
    (a,b)
  long long x,y;
  if (b==0) return {1,0};
  auto p=extendedEuclid(b,a%b);
  x=p.second;
  y=p.first-(a/b)*x;
  if(a*x+b*y==-gcd(a,b)) x=-x, y=-y;
  return {x,y};
}
pair<pair<long long,long long>,pair<long long,
    long long> > diophantine(long long a,long
    long b, long long r) {
  //a*x+b*y=r where r is multiple of gcd(a,b);
  long long d=gcd(a,b);
  a/=d; b/=d; r/=d;
  auto p = extendedEuclid(a,b);
  p.first*=r; p.second*=r;
  assert(a*p.first+b*p.second==r);
  return {p,{-b,a}}; // solutions: p+t*ans.snd
}

long long inv(long long a, long long m) {
  assert(gcd(a,m)==1);
  long long x = diophantine(a,m,1).first.first;
  return ((x%m)+m)%m;
}

#define mod(a,m) (((a)%m+m)%m)
pair<long long,long long> sol(tuple<long long,
    long long,long long> c){ //requires inv,
    diophantine
    long long a=get<0>(c), x1=get<1>(c), m=get
        <2>(c), d=gcd(a,m);
    if(d==1) return {mod(x1*inv(a,m),m), m};
    else return x1%d ? pair<long long, long
        long>({-1LL,-1LL}) : sol(make_tuple(a/d
        ,x1/d,m/d));
}
pair<long long,long long> crt(vector< tuple<
    long long,long long,long long> > cond) { //
     returns: (sol, lcm)
  long long x1=0,m1=1,x2,m2;
  for(auto t:cond){
    tie(x2,m2)=sol(t);
    if((x1-x2)%gcd(m1,m2))return {-1,-1};
    if(m1==m2)continue;
    long long k=diophantine(m2,-m1,x1-x2).first
        .second,l=m1*(m2/gcd(m1,m2));
    x1=mod((__int128)m1*k+x1,l);m1=l;
  }
  return sol(make_tuple(1,x1,m1));
} //cond[i]={ai,bi,mi} ai*xi=bi (mi); assumes
    lcm fits in ll
```

3.11 Discrete log

```
long long powm(long long a, long long b, long
    long mod){
  long long res =1;
  while(b){ if(b&1) res = (res * a) % mod; a =
      (a*a) % mod; b/=2; }
  return res;
}
long long discrete_log(long long a,long long b,
    long long m) {
    a%=m, b%=m;
    if(b == 1) return 0;
    int cnt=0;
    long long tmp=1;
    for(int g=__gcd(a,m);g!=1;g=__gcd(a,m)) {
        if(b%g) return -1;
        m/=g, b/=g;
        tmp = tmp*a/g%m;
        ++cnt;
        if(b == tmp) return cnt;
    }
    map<long long,int> w;
    int s = ceil(sqrt(m));
    long long base = b;
    for(int i = 0; i < s; ++i) {
        w[base] = i;
        base=base*a%m;
    }
    base=powm(a,s,m);
    long long key=tmp;
    for(int i = 1; i < s + 2; ++i) {
        key=base*key%m;
        if(w.count(key)) return i*s-w[key]+cnt;
    }
    return -1;
}
```

3.12 Matrix reduce

```
double reduce(vector<vector<double> >& x){ //
    returns determinant
  int n=x.size(),m=x[0].size();
  int i=0,j=0;double r=1.;
  while(i<n&&j<m){
    int l=i;
    fore(k,i+1,n)if(abs(x[k][j])>abs(x[l][j]))l
        =k;
    if(abs(x[l][j])<EPS){j++;r=0.;continue;}
    if(l!=i){r=-r;swap(x[i],x[l]);}
    r*=x[i][j];
    for(int k=m-1;k>=j;k--)x[i][k]/=x[i][j];
```

```
    fore(k,0,n){
      if(k==i)continue;
      for(int l=m-1;l>=j;l--)x[k][l]-=x[k][j]*x
          [i][l];
    }
    i++;j++;
  }
  return r;
}
```

3.13 Berlekamp Massey

```
long long MOD = 1e9 + 7;
long long powm(long long a, long long b, long
    long mod){
  long long res =1;
  while(b){ if(b&1) res = (res * a) % mod; a =
      (a*a) % mod; b/=2; }
  return res;
}
vector<int> BM(vector<int> x){
vector<int> ls,cur;int lf,ld;
for(int i = 0; i < (int)(x.size()); ++i){
  long long t=0;
  for(int j = 0; j < (int)(cur.size()); ++j) t
      =(t+x[i-j-1]*(long long)cur[j])%MOD;
  if((t-x[i])%MOD==0)continue;
  if(!(int)cur.size()){cur.resize(i+1);lf=i;ld
      =(t-x[i])%MOD;continue;}
  long long k=-(x[i]-t)*powm(ld,MOD-2, MOD)%MOD
      ;
  vector<int> c(i-lf-1);c.push_back(k);
  for(int j = 0; j < (int)(ls.size()); ++j) c.
      push_back(-ls[j]*k%MOD);
  if((int)c.size()<(int)(cur.size()))c.resize((
      int)(cur.size()));
  for(int j = 0; j < (int)(cur.size()); ++j) c[
      j]=(c[j]+cur[j])%MOD;
  if(i-lf+(int)(ls.size())>=(int)(cur.size()))
      ls=cur,lf=i,ld=(t-x[i])%MOD;
  cur=c;
}
for(int i = 0; i < (int)(cur.size()); ++i) cur[
    i]=(cur[i]%MOD+MOD)%MOD;
return cur;
}
```

3.14 Linear Rec

```
//Needs MOD and LOG
struct LinearRec{
  typedef vector<int> vi;
  int n; vi terms, trans; vector<vi> bin;
  vi add(vi &a, vi &b){
    vi res(n*2+1);
    for(int i = 0; i < n + 1; ++i) for(int j =
        0; j < n + 1; ++j) res[i+j]=(res[i+j]*1
        LL+(long long)a[i]*b[j])%MOD;
    for(int i=2*n; i>n; --i){
      for(int j = 0; j < n; ++j) res[i-1-j]=(
          res[i-1-j]*1LL+(long long)res[i]*
          trans[j])%MOD;
      res[i]=0;
    }
    res.erase(res.begin()+n+1,res.end());
    return res;
  }
  LinearRec(vi &terms, vi &trans):terms(terms),
      trans(trans){
    n=(int)(trans.size());vi a(n+1);a[1]=1;
    bin.push_back(a);
    for(int i = 1; i < LOG; ++i) bin.push_back(
        add(bin[i-1],bin[i-1]));
  }
  int calc(int k){
    vi a(n+1);a[0]=1;
    for(int i = 0; i < LOG; ++i) if((k>>i)&1)a=
        add(a,bin[i]);
    int ret=0;
    for(int i = 0 ; i < n ; ++i) ret=((long
        long)ret+(long long)a[i+1]*terms[i])%
        MOD;
    return ret;
  }
};
```

3.15 Points Under Line

```
long long f(long long a, long long b, long long
    c){
  if(c<=0) return 0;
  if(a<b) swap(a, b);
  long long m=c/a;
  if(a==b) return m*(m-1)/2;
  long long k=(a-1)/b, h=(c-a*m)/b;
  return f(b,a-b*k,c-b*(k*m+h))+k*m*(m-1)/2+m*h
      ;
}

// # of lattice points s.t. ax+by<=c, 0<x<=X,
    0<y<=Y (a,b is positive integer)
long long g(long long a, long long b, long long
    c, long long X, long long Y){
  if(a*X+b*Y<=c) return X*Y;
  return f(a,b,c)-f(a,b,c-a*X)-f(a,b,c-b*Y)+f(a
      ,b,c-a*X-b*Y);
}
```

3.16 Theorems and Formulas

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\sum_{i=0}^{k}\binom{n+i}{i} = \binom{n+k+1}{k}$$

$\left[\begin{matrix} n \\ k \end{matrix}\right]$ = perm of n elements with k cycles

$$\left[\begin{matrix} n+1 \\ k \end{matrix}\right] = n\left[\begin{matrix} n \\ k \end{matrix}\right] + \left[\begin{matrix} n \\ k-1 \end{matrix}\right]$$

$\left\{\begin{matrix} n \\ k \end{matrix}\right\}$ = partitions of an n-element set into k parts

$$\left\{\begin{matrix} n \\ k \end{matrix}\right\} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n.$$

$$\left\{\begin{matrix} n+1 \\ k \end{matrix}\right\} = k\left\{\begin{matrix} n \\ k \end{matrix}\right\} + \left\{\begin{matrix} n \\ k-1 \end{matrix}\right\}$$

Integers $d_1 \geq \cdots \geq d_n \geq 0$ can be the degree sequence of a finite simple graph on n vertices $\iff$ $d_1 + \cdots + d_n$ is even and for every k in $1 \leq k \leq n$

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

$a^n = a^{\varphi(m)+n \mod \varphi(m)} \pmod m$ if $n > lg(m)$
Misere Nim: $ai >= 1$;if $\exists a_i > 1$ then normal nim; else if n impar iff lose
Derangements: Num of permutations of $n = 0, 1, 2, ...$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, ...$ Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = n * D_{n-1} + (-1)^n$. Collary: number of permutations with exactly k fixed points $nCkD_{n-k}$
Eulerian numbers: $E(n,k)$ is the number of permutations with exactly k descents $(i : \pi_i < \pi_{i+1})$, ascents $(\pi_i > \pi_{i+1})$ / excedances $(\pi > i)$ / $k+1$ weak excedances $(\pi \geq i)$.
$E_{n,k} = (k+1)E_{n-1,k)} + (n-k)E_{n-1,k-1}$

4 Geometry

4.1 Point

```
//Needs a EPS

struct pt {  // for 3D add z coordinate
  double x,y;
  pt(double x, double y):x(x),y(y){}
  pt(){}
  double norm2(){return *this**this;}
  double norm(){return sqrt(norm2());}
  bool operator==(pt p){return abs(x-p.x)<=EPS
      &&abs(y-p.y)<=EPS;}
  pt operator+(pt p){return pt(x+p.x,y+p.y);}
  pt operator-(pt p){return pt(x-p.x,y-p.y);}
  pt operator*(double t){return pt(x*t,y*t);}
  pt operator/(double t){return pt(x/t,y/t);}
  double operator*(pt p){return x*p.x+y*p.y;}
//  pt operator^(pt p){ // only for 3D
//    return pt(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y
    *p.x);}
  double angle(pt p){ // redefine acos for
      values out of range
    return acos(*this*p/(norm()*p.norm()));}
  pt unit(){return *this/norm();}
  double operator%(pt p){return x*p.y-y*p.x;}
  // 2D from now on
  bool operator<(pt p)const{ // for convex hull
    return x<p.x-EPS||(abs(x-p.x)<=EPS&&y<p.y-
        EPS);}
  bool left(pt p, pt q){ // is it to the left
      of directed line pq?
    return (q-p)%(*this-p)>EPS;}
  pt rot(pt r){return pt(*this%r,*this*r);}
  pt rot(double a){return rot(pt(sin(a),cos(a))
      );}
};
pt ccw90(1,0);
pt cw90(-1,0);
```

4.2 Line

```
//Needs EPS, INF and DINF

struct ln {
  pt p,pq;
  ln(pt p, pt q):p(p),pq(q-p){}
  ln(){}
  bool has(pt r){return dist(r)<=EPS;}
  bool seghas(pt r){return has(r)&&(r-p)*(r-(p+
      pq))<=EPS;}
//  bool operator /(ln l){return (pq.unit()^l.
    pq.unit()).norm()<=EPS;} // 3D
  bool operator/(ln l){return abs(pq.unit()%l.
      pq.unit())<=EPS;} // 2D
  bool operator==(ln l){return *this/l&&has(l.p
      );}
  pt operator^(ln l){ // intersection
    if(*this/l)return pt(DINF,DINF);
    pt r=l.p+l.pq*((p-l.p)%pq/(l.pq%pq));
//    if(!has(r)){return pt(NAN,NAN,NAN);} //
    check only for 3D
    return r;
  }
  double angle(ln l){return pq.angle(l.pq);}
  int side(pt r){return has(r)?0:sgn2(pq%(r-p))
      ;} // 2D
  pt proj(pt r){return p+pq*((r-p)*pq/pq.norm2
      ());}
  pt ref(pt r){return proj(r)*2-r;}
  double dist(pt r){return (r-proj(r)).norm();}
//  double dist(ln l){ // only 3D
//    if(*this/l)return dist(l.p);
//    return abs((l.p-p)*(pq^l.pq))/(pq^l.pq).
    norm();
//  }
  ln rot(double a){return ln(p,p+pq.rot(a));}
      // 2D
  ln rot(pt a){return ln(p,p+pq.rot(a));}
};
ln bisector(ln l, ln m){ // angle bisector
  pt p=l^m;
  return ln(p,p+l.pq.unit()+m.pq.unit());
}
ln bisector(pt p, pt q){ // segment bisector (2
    D)
  return ln((p+q)*.5,p).rot(ccw90);
}
```

4.3 Circle

```
struct circle {
  pt o;double r;
  circle(pt o, double r):o(o),r(r){}
  circle(pt x, pt y, pt z){o=bisector(x,y)^
      bisector(x,z);r=(o-x).norm();}
  bool has(pt p){return (o-p).norm()<=r+EPS;}
  vector<pt> operator^(circle c){ // ccw
    vector<pt> s;
    double d=(o-c.o).norm();
    if(d>r+c.r+EPS||d+min(r,c.r)+EPS<max(r,c.r)
        )return s;
    double x=(d*d-c.r*c.r+r*r)/(2*d);
    double y=sqrt(r*r-x*x);
    pt v=(c.o-o)/d;
    s.push_back(o+v*x-v.rot(ccw90)*y);
    if(y>EPS)s.push_back(o+v*x+v.rot(ccw90)*y);
    return s;
  }
  vector<pt> operator^(ln l){
    vector<pt> s;
    pt p=l.proj(o);
    double d=(p-o).norm();
```

```
    if(d-EPS>r)return s;
    if(abs(d-r)<=EPS){s.push_back(p);return s;}
    d=sqrt(r*r-d*d);
    s.push_back(p+l.pq.unit()*d);
    s.push_back(p-l.pq.unit()*d);
    return s;
  }
  vector<pt> tang(pt p){
    double d=sqrt((p-o).norm2()-r*r);
    return *this^circle(p,d);
  }
  bool in(circle c){ // non strict
    double d=(o-c.o).norm();
    return d+r<=c.r+EPS;
  }
  double intertriangle(pt a, pt b){ // area of
      intersection with oab
    if(abs((o-a)%(o-b))<=EPS)return 0.;
    vector<pt> q={a},w=*this^ln(a,b);
    if(w.size()==2)for(auto p:w)if((a-p)*(b-p)
        <-EPS)q.push_back(p);
    q.push_back(b);
    if(q.size()==4&&(q[0]-q[1])*(q[2]-q[1])>EPS
        )swap(q[1],q[2]);
    double s=0;
    for(int i = 0; i < (int)q.size() - 1; ++i)
        {
      if(!has(q[i])||!has(q[i+1]))s+=r*r*(q[i]-
          o).angle(q[i+1]-o)/2;
      else s+=abs((q[i]-o)%(q[i+1]-o)/2);
    }
    return s;
  }
};
```

4.4 Polygon

```
//Need CHULL
int sgn(double x){return x<-EPS?-1:x>EPS;}
struct pol {
  int n;vector<pt> p;
  pol(){}
  pol(vector<pt> _p){p=_p;n=p.size();}
  double area(){
    double r=0.;
    for(int i = 0; i < n; ++i)r+=p[i]%p[(i+1)%n
        ];
    return abs(r)/2; // negative if CW,
        positive if CCW
  }
  pt centroid(){ // (barycenter)
    pt r(0,0);double t=0;
    for(int i = 0; i < n; ){
      r=r+(p[i]+p[(i+1)%n])*(p[i]%p[(i+1)%n]);
      t+=p[i]%p[(i+1)%n];
    }
    return r/t/3;
  }
  bool has(pt q){ // O(n)
    for(int i = 0; i < n; ++i)if(ln(p[i],p[(i
        +1)%n]).seghas(q))return true;
    int cnt=0;
    for(int i = 0; i < n; ++i){
      int j=(i+1)%n;
      int k=sgn((q-p[j])%(p[i]-p[j]));
      int u=sgn(p[i].y-q.y),v=sgn(p[j].y-q.y);
      if(k>0&&u<0&&v>=0)cnt++;
      if(k<0&&v<0&&u>=0)cnt--;
    }
    return cnt!=0;
  }
  void normalize(){ // (call before haslog,
      remove collinear first)
    if(p[2].left(p[0],p[1]))reverse(p.begin(),p
        .end());
    int pi=min_element(p.begin(),p.end())-p.
        begin();
    vector<pt> s(n);
    for(int i = 0; i < n; ++i) s[i]=p[(pi+i)%n
        ];
    p.swap(s);
  }
  bool haslog(pt q){ // O(log(n)) only CONVEX.
      Call normalize first
    if(q.left(p[0],p[1])||q.left(p.back(),p[0])
        )return false;
    int a=1,b=p.size()-1;  // returns true if
        point on boundary
    while(b-a>1){          // (change sign of
        EPS in left
      int c=(a+b)/2;       //  to return false
          in such case)
      if(!q.left(p[0],p[c]))a=c;
      else b=c;
    }
    return !q.left(p[a],p[a+1]);
  }
  pt farthest(pt v){ // O(log(n)) only CONVEX
    if(n<10){
      int k=0;
      for(int i = 1; i < n; ++i)if(v*(p[i]-p[k
          ])>EPS)k=i;
      return p[k];
    }
    if(n==(int)p.size()) p.push_back(p[0]);
    pt a=p[1]-p[0];
    int s=0,e=n,ua=v*a>EPS;
    if(!ua&&v*(p[n-1]-p[0])<=EPS)return p[0];
    while(1){
      int m=(s+e)/2;pt c=p[m+1]-p[m];
      int uc=v*c>EPS;
      if(!uc&&v*(p[m-1]-p[m])<=EPS)return p[m];
      if(ua&&(!uc||v*(p[s]-p[m])>EPS))e=m;
      else if(ua||uc||v*(p[s]-p[m])>=-EPS)s=m,a
          =c,ua=uc;
      else e=m;
      assert(e>s+1);
    }
  }
  pol cut(ln l){   // cut CONVEX polygon by
      line l
    vector<pt> q;  // returns part at left of l
        .pq
    for(int i = 0; i < n; ++i){
      int d0=sgn(l.pq%(p[i]-l.p)),d1=sgn(l.pq%(
          p[(i+1)%n]-l.p));
      if(d0>=0)q.push_back(p[i]);
      ln m(p[i],p[(i+1)%n]);
```

```
      if(d0*d1<0&&!(l/m))q.push_back(l^m);
    }
    return pol(q);
  }
  double intercircle(circle c){ // area of
      intersection with circle
    double r=0.;
    for(int i = 0; i < n; ++i){
      int j=(i+1)%n;double w=c.intertriangle(p[
          i],p[j]);
      if((p[j]-c.o)%(p[i]-c.o)>0)r+=w;
      else r-=w;
    }
    return abs(r);
  }
  double callipers(){ // square distance of
      most distant points
    double r=0;     // prereq: convex, ccw, NO
        COLLINEAR POINTS
    for(int i=0,j=n<2?0:1;i<j;++i){
      for(;;j=(j+1)%n){
        r=max(r,(p[i]-p[j]).norm2());
        if((p[(i+1)%n]-p[i])%(p[(j+1)%n]-p[j])
            <=EPS)break;
      }
    }
    return r;
  }
};
// Dynamic convex hull trick
vector<pol> w;
void add(pt q){ // add(q), O(log^2(n))
  vector<pt> p={q};
  while(!w.empty()&&(int)(w.back().p).size()
      <2*(int)p.size()){
    for(pt v:w.back().p)p.push_back(v);
    w.pop_back();
  }
  w.push_back(pol(chull(p)));
}
long long query(pt v){ // max(q*v:q in w), O(
    log^2(n))
  long long r=-INF;
  for(auto& p:w)r=max(r,(long long)(p.farthest(
      v)*v));
  return r;
}
```

4.5 Plane

```
struct plane {
  pt a,n; // n: normal unit vector
  plane(pt a, pt b, pt c):a(a),n(((b-a)^(c-a)).
      unit()){}
  plane(){}
  bool has(pt p){return abs((p-a)*n)<=EPS;}
  double angle(plane w){return acos(n*w.n);}
  double dist(pt p){return abs((p-a)*n);}
  pt proj(pt p){inter(ln(p,p+n),p);return p;}
  bool inter(ln l, pt& r){
    double x=n*(l.p+l.pq-a),y=n*(l.p-a);
    if(abs(x-y)<=EPS)return false;
    r=(l.p*x-(l.p+l.pq)*y)/(x-y);
    return true;
  }
  bool inter(plane w, ln& r){
    pt nn=n^w.n;pt v=n^nn;double d=w.n*v;
    if(abs(d)<=EPS)return false;
    pt p=a+v*(w.n*(w.a-a)/d);
    r=ln(p,p+nn);
    return true;
  }
};
```

4.6 Radial order of points

```
struct Cmp { // IMPORTANT: add const in pt
    operator -
  pt r;
  Cmp(pt r):r(r){}
  int cuad(const pt &a)const {
    if(a.x>0&&a.y>=0)return 0;
    if(a.x<=0&&a.y>0)return 1;
    if(a.x<0&&a.y<=0)return 2;
    if(a.x>=0&&a.y<0)return 3;
    assert(a.x==0&&a.y==0);
    return -1;
  }
  bool cmp(const pt& p1, const pt& p2)const {
    int c1=cuad(p1),c2=cuad(p2);
    if(c1==c2)return p1.y*p2.x<p1.x*p2.y;
    return c1<c2;
  }
  bool operator()(const pt& p1, const pt& p2)
      const {
    return cmp(pt(p1)-pt(r),pt(p2)-pt(r));
  }
};
```

4.7 Convex hull

```
vector<pt> chull(vector<pt> p){
  if(p.size()<3)return p;
  vector<pt> r;
  sort(p.begin(),p.end()); // first x, then y
  for(int i = 0; i < (int)p.size(); ++i){ //
      lower hull
    while(r.size()>=2&&r.back().left(r[r.size()
        -2],p[i]))r.pop_back();
    r.push_back(p[i]);
  }
  r.pop_back();
  int k=r.size();
  for(int i=p.size()-1;i>=0;--i){ // upper hull
    while((int)r.size()>=k+2&&r.back().left(r[(
        int)r.size()-2],p[i]))r.pop_back();
    r.push_back(p[i]);
  }
  r.pop_back();
  return r;
}
```

4.8 Dual from planar graph

```
const int MAXN = 1;
vector<int> g[MAXN];int n; // input graph (must
    be connected)
vector<int> gd[MAXN];int nd; // output graph
```

```
vector<int> nodes[MAXN]; // nodes delimiting region (in CW order)
map<pair<int,int>,int> ps,es;
void get_dual(vector<pt> p){ // p: points corresponding to nodes
  ps.clear();es.clear();
  for(int x = 0; x < n; ++x){
    Cmp pc(p[x]); // (radial order of points)
    auto comp=[&](int a, int b){return pc(p[a],p[b]);};
    sort(g[x].begin(),g[x].end(),comp);
    for(int i = 0; i < (int)g[x].size(); ++i)ps[{x,g[x][i]}]=i;
  }
  nd=0;
  for(int xx = 0; xx < n; ++xx)for(auto yy:g[xx])if(!es.count({xx,yy})){
    int x=xx,y=yy;gd[nd].clear();nodes[nd].clear();
    while(!es.count({x,y})){
      es[{x,y}]=nd;nodes[nd].push_back(y);
      int z=g[y][(ps[{y,x}]+1)%g[y].size()];x=y;y=z;
    }
    nd++;
  }
  for(auto p:es){
    pair<int,int> q={p.first.second,p.first.first};
    assert(es.count(q));
    if(es[q]!=p.second)gd[p.second].push_back(es[q]);
  }
  for(int i = 0; i < nd; ++i){
    sort(gd[i].begin(),gd[i].end());
    gd[i].erase(unique(gd[i].begin(),gd[i].end()),gd[i].end());
  }
}
```

4.9 Halfplane intersection

```
struct halfplane:public ln{
  double angle;
  halfplane(){}
  halfplane(pt a,pt b){p=a; pq=b-a; angle=atan2(pq.y,pq.x);}
  bool operator<(halfplane b)const{return angle<b.angle;}
  bool out(pt q){return pq%(q-p)<-EPS;}
};
vector<pt> intersect(vector<halfplane> b){
  vector<pt>bx={{DINF,DINF},{-DINF,DINF},{-DINF,-DINF},{DINF,-DINF}};
  for(int i = 0; i < 4; ++i) b.push_back(halfplane(bx[i],bx[(i+1)%4]));
  sort(b.begin(), b.end());
  int n=(int)b.size(),q=1,h=0;
  vector<halfplane> c((int)b.size()+10);
  for(int i = 0; i < n; ++i){
    while(q<h&&b[i].out(c[h]^c[h-1])) h--;
    while(q<h&&b[i].out(c[q]^c[q+1])) q++;
    c[++h]=b[i];
    if(q<h&&abs(c[h].pq%c[h-1].pq)<EPS){
      if(c[h].pq*c[h-1].pq<=0) return {};
      h--;
      if(b[i].out(c[h].p)) c[h]=b[i];
    }
  }
  while(q<h-1&&c[q].out(c[h]^c[h-1]))h--;
  while(q<h-1&&c[h].out(c[q]^c[q+1]))q++;
  if(h-q<=1)return {};
  c[h+1]=c[q];
  vector<pt> s;
  for(int i = q; i < h+1; ++i) s.push_back(c[i]^c[i+1]);
  return s;
}
```

4.10 KD Tree

```
bool onx(pt a, pt b){return a.x<b.x;}
bool ony(pt a, pt b){return a.y<b.y;}

struct Node {
  pt pp;
  long long x0=INF, x1=-INF, y0=INF, y1=-INF;
  Node *first=0, *second=0;
  long long distance(pt p){
    long long x=min(max(x0,(long long)p.x),x1);
    long long y=min(max(y0,(long long)p.y),y1);
    return (pt(x,y)-p).norm2();
  }
  Node(vector<pt>&& vp):pp(vp[0]){
    for(pt p:vp){
      x0=min(x0,(long long)p.x); x1=max(x1,(long long)p.x);
      y0=min(y0,(long long)p.y); y1=max(y1,(long long)p.y);
    }
    if(vp.size() > 1){
      sort(vp.begin(), vp.end(),x1-x0>=y1-y0?onx:ony);
      int m=(int)(vp.size())/2;
      first=new Node({vp.begin(),vp.begin()+m});
      second=new Node({vp.begin()+m,vp.end()});
    }
  }
};
struct KDTree {
  Node* root;
  KDTree(const vector<pt>& vp):root(new Node({vp.begin(), vp.end()})) {}
  pair<long long,pt> search(pt p, Node *node){
    if(!node->first){
      //avoid query point as answer
      //if(p==node->pp) {INF,pt()};
      return {(p-node->pp).norm2(),node->pp};
    }
    Node *f=node->first, *s=node->second;
    long long bf=f->distance(p), bs=s->distance(p);
    if(bf>bs)swap(bf,bs),swap(f,s);
    auto best=search(p,f);
    if(bs<best.first) best=min(best,search(p,s));
    return best;
```

```
  }
  pair<long long,pt> nearest(pt p){return
      search(p,root);}
};
```

4.11 Theorems and Formulas

The n-dimensional volume of a ball of radius r is

$$V_n(r) = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)} r^n$$

where $\Gamma(n) = (n-1)!$ and $\Gamma(n+\frac{1}{2}) = (n-\frac{1}{2})\cdot\ldots\cdot\frac{1}{2}\cdot\pi^{\frac{1}{2}}$

If $\mathbf{v} \in \mathbb{R}^3$ and k is a unit vector describing an axis of rotation about which $\mathbf{v}$ rotates by an angle θ according to the right hand rule:
$\mathbf{v}_{\text{rot}} = \mathbf{v}\cos\theta + (\mathbf{k}\times\mathbf{v})\sin\theta + \mathbf{k}\,(\mathbf{k}\cdot\mathbf{v})(1-\cos\theta)$

Spherical cone with sphere radius r, height h and radius of circle a:
$A = \pi r(2h+a)$ (includes lateral area)
$V = \frac{2}{3}\pi r^2 h$

Surface/Solid of revolution
$A = 2\pi L d$, L = length of curve
$V = 2\pi A d$, A = area of surface
d = distance of centroid to axis

5 Strings

5.1 KMP

```
vector<int> kmppre(string& t){ // r[i]: longest
    border of t[0,i)
  vector<int> r(t.size()+1);r[0]=-1;
  int j=-1;
  for(int i = 0; i < (int)(t.size()); ++i){
    while(j>=0&&t[i]!=t[j])j=r[j];
    r[i+1]=++j;
  }
  return r;
}
void kmp(string& s, string& t){ // find t in s
  int j=0;vector<int> b=kmppre(t);
  for(int i = 0; i < (int)(s.size()); ++i){
    while(j>=0&&s[i]!=t[j])j=b[j];
    if(++j==(int)t.size())printf("Match␣at␣%d\n
        ",i-j+1),j=b[j];
  }
}
```

5.2 Z function

```
vector<int> z_function(string& s){
  int l=0,r=0,n=s.size();
  vector<int> z(s.size(),0); // z[i] = max k: s
      [0,k) == s[i,i+k)
  for(int i = 1; i < n; ++i){
    if(i<=r)z[i]=min(r-i+1,z[i-l]);
    while(i+z[i]<n&&s[z[i]]==s[i+z[i]])z[i]++;
    if(i+z[i]-1>r)l=i,r=i+z[i]-1;
  }
  return z;
}
```

5.3 Hashing

```
#define bint __int128
struct Hash {
  bint MOD=212345678987654321LL,P=1777771,PI
      =1069557410896595 71LL;
  vector<bint> h,pi;
  Hash(string& s){
    assert((P*PI)%MOD==1);
    h.resize(s.size()+1);pi.resize(s.size()+1);
    h[0]=0;pi[0]=1;
    bint p=1;
    for(int i = 1; i < (int)(s.size()) + 1; ++i
        ){
      h[i]=(h[i-1]+p*s[i-1])%MOD;
      pi[i]=(pi[i-1]*PI)%MOD;
      p=(p*P)%MOD;
    }
  }
  long long get(int s, int e){
    return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
  }
};
```

5.4 Manacher

```
const int MAXN = 1e6;
int d1[MAXN];//d1[i] = max odd palindrome
    centered on i
int d2[MAXN];//d2[i] = max even palindrome
    centered on i
//s  aabbaacaabbaa
//d1 1111117111111
//d2 0103010010301
void manacher(string& s){
  int l=0,r=-1,n=s.size();
  for(int i = 0; i < n; ++i){
    int k=i>r?1:min(d1[l+r-i],r-i);
    while(i+k<n&&i-k>=0&&s[i+k]==s[i-k])k++;
    d1[i]=k--;
    if(i+k>r)l=i-k,r=i+k;
  }
  l=0;r=-1;
  for(int i = 0; i < n; ++i){
    int k=i>r?0:min(d2[l+r-i+1],r-i+1);k++;
    while(i+k<=n&&i-k>=0&&s[i+k-1]==s[i-k])k++;
    d2[i]=--k;
    if(i+k-1>r)l=i-k,r=i+k-1;
  }
}
```

5.5 Aho-Corasick

```
const int A = 26;
struct vertex {
  vi next,go,leaf;
  int p,link,nl;
  char pch;
  vertex(int p=-1, char pch=-1):p(p),pch(pch),
      link(-1),nl(-1), next(A,-1),go(A,0){}
};
vector<vertex> t;
void aho_ini(){t.clear();t.pb(vertex());}
void add(string s, int id){
  int v=0;
  for(auto x:s){
    int c = x-'a';
```

```
    if(t[v].next[c]==-1){
      t[v].next[c]=t[v].go[c]=sz(t);
      t.pb(vertex(v,c));
    }
    v=t[v].next[c];
  }
  t[v].leaf.pb(id);
}
int go(int v, int c){return t[v].go[c];}
void BFS(){
  queue<int>q;
  q.push(0);
  t[0].link=t[0].nl=0;
  while(!q.empty()){
    int x = q.front(); q.pop();
    fore(c,0,A){
      if(t[x].next[c]==-1)continue;
      int y = t[x].next[c];
      t[y].link=x?t[t[x].link].go[c]:0;
      int link = t[y].link;
      t[y].nl = sz(t[link].leaf)?link:t[link].
          nl;
      for(int i = 0; i<A; i++)if(t[y].next[i
          ]==-1)t[y].go[i]=t[link].go[i];
      q.push(y);
    }
  }
}
```

5.6 Suffix automaton

```
struct state {int len,link;map<char,int> next
    ;}; //clear next!!
state st[100005];
int sz,last;
void sa_init(){
  last=st[0].len=0;sz=1;
  st[0].link=-1;
}
void sa_extend(char c){
  int k=sz++,p;
  st[k].len=st[last].len+1;
  for(p=last;p!=-1&&!st[p].next.count(c);p=st[p
      ].link)st[p].next[c]=k;
  if(p==-1)st[k].link=0;
  else {
    int q=st[p].next[c];
    if(st[p].len+1==st[q].len)st[k].link=q;
    else {
      int w=sz++;
      st[w].len=st[p].len+1;
      st[w].next=st[q].next;st[w].link=st[q].
          link;
      for(;p!=-1&&st[p].next[c]==q;p=st[p].link
          )st[p].next[c]=w;
      st[q].link=st[k].link=w;
    }
  }
  last=k;
}
```

5.7 Palindromic Tree

```
struct palindromic_tree{
    static const int SIGMA=26;
    struct Node{
        int len, link, to[SIGMA];
        long long cnt;
        Node(int len, int link=0, long long cnt
            =1):len(len),link(link),cnt(cnt){
            memset(to,0,sizeof(to));
        }
    };
    vector<Node> ns;
    int last;
    palindromic_tree():last(0){ns.push_back(
        Node(-1));ns.push_back(Node(0));}
    void add(int i, string &s){
        int p=last, c=s[i]-'a';
        while(s[i-ns[p].len-1]!=s[i])p=ns[p].
            link;
        if(ns[p].to[c]){
            last=ns[p].to[c];
            ns[last].cnt++;
        }else{
            int q=ns[p].link;
            while(s[i-ns[q].len-1]!=s[i])q=ns[q
                ].link;
            q=max(1,ns[q].to[c]);
            last=ns[p].to[c]=(int)(ns.size());
            ns.push_back(Node(ns[p].len+2,q,1))
                ;
        }
    }
};
```

5.8 Suffix array

```

#define RB(x) (x<n?r[x]:0)
void csort(vector<int>& sa, vector<int>& r, int
     k){
  int n=sa.size();
  vector<int> f(max(255,n),0),t(n);
  fore(i,0,n)f[RB(i+k)]++;
  int sum=0;
  fore(i,0,max(255,n))f[i]=(sum+=f[i])-f[i];
  fore(i,0,n)t[f[RB(sa[i]+k)]++]=sa[i];
  sa=t;
}
vector<int> constructSA(string& s){ // O(n logn
    )
  int n=s.size(),rank;
  vector<int> sa(n),r(n),t(n);
  fore(i,0,n)sa[i]=i,r[i]=s[i];
  for(int k=1;k<n;k*=2){
    csort(sa,r,k);csort(sa,r,0);
    t[sa[0]]=rank=0;
    fore(i,1,n){
      if(r[sa[i]]!=r[sa[i-1]]||RB(sa[i]+k)!=RB(
          sa[i-1]+k))rank++;
      t[sa[i]]=rank;
    }
    r=t;
    if(r[sa[n-1]]==n-1)break;
  }
  return sa;
}
```

5.9 LCP (Longest Common Prefix)

```
vector<int> computeLCP(string& s, vector<int>& sa){
  int n=(int)s.size(),L=0;
  vector<int> lcp(n),plcp(n),phi(n);
  phi[sa[0]]=-1;
  for(int i = 1; i < n; ++i) phi[sa[i]]=sa[i-1];
  for(int i = 0; i < n; ++i){
    if(phi[i]<0){plcp[i]=0;continue;}
    while(s[i+L]==s[phi[i]+L])L++;
    plcp[i]=L;
    L=max(L-1,0);
  }
  for(int i = 0; i < n; ++i) lcp[i]=plcp[sa[i]];
  return lcp; // lcp[i]=LCP(sa[i-1],sa[i])
}
```

5.10 Minimum Lexicographic rotation

```
vi getminlex(vi s){
  int n=sz(s),k=0; fore(i,0,n) s.pb(s[i]);
  vi f(2*n,-1);
  fore(j,1,2*n){
    int i=f[j-k-1];
    while(i>=0&&s[j]!=s[k+i+1]){
      if(s[j]<s[k+i+1]) k=j-i-1;
      i=f[i];
    }
    if(s[j]!=s[k+i+1]){
      if(s[j]<s[k])k=j;
      f[j-k]=-1;
    } else f[j-k]=i+1;
  }
  vi ans; fore(i,0,n) ans.pb(s[k+i]);
  return ans;
}
```

6 Flow

6.1 Matching O(n^2)

```
struct Bipartite_Matching {
  vector<vector<int>> graph;
  vector<int> dist, match, used;
  vector<bool> vv;

  Bipartite_Matching(int n, int m) {
    graph.resize(n);
    match.assign(m, -1);
    used.assign(n, -1);
  }

  void add(int u, int v) { graph[u].push_back(v);}

  void bfs() {
    dist.assign(graph.size(), -1);
    queue< int > que;
    for(int i = 0; i < graph.size(); i++) {
      if(used[i] == -1) {
        que.emplace(i);
        dist[i] = 0;
      }
    }

    while(!que.empty()) {
      int a = que.front();
      que.pop();
      for(auto &b : graph[a]) {
        int c = match[b];
        if(c >= 0 && dist[c] == -1) {
          dist[c] = dist[a] + 1;
          que.emplace(c);
        }
      }
    }
  }

  bool dfs(int a) {
    vv[a] = true;
    for(auto &b : graph[a]) {
      int c = match[b];
      if(c < 0 || (!vv[c] && dist[c] == dist[a] + 1 && dfs(c))) {
        match[b] = a;
        used[a] = b;
        return (true);
      }
    }
    return (false);
  }

  int bipartite_matching() {
    int ret = 0;
    while(true) {
      bfs();
      vv.assign(graph.size(), false);
      int flow = 0;
      for(int i = 0; i < graph.size(); i++) {
        if(used[i] == -1 && dfs(i)) ++flow;
      }
      if(flow == 0) return (ret);
      ret += flow;
    }
  }
};
```

6.2 Hungarian

```
typedef long double td;
typedef vector<int> vi;
typedef vector<td> vd;
const td INF=1e10;//for maximum set INF to 0, and negate costs
bool zero(td x){return fabs(x)<1e-9;}//change to x==0, for ints/ll
struct Hungarian{
    int n; vector<vd> cs; vi L, R;
    Hungarian(int N, int M):n(max(N,M)),cs(n,vd(n)),L(n),R(n){
        fore(x,0,N)fore(y,0,M)cs[x][y]=INF;
    }
    void set(int x,int y,td c){cs[x][y]=c;}
  td assign() {
    int mat = 0; vd ds(n), u(n), v(n); vi dad(n
```

```
        ), sn(n);
    fore(i,0,n)u[i]=*min_element(all(cs[i]));
    fore(j,0,n){
            v[j]=cs[0][j]-u[0];
            fore(i,1,n)v[j]=min(v[j],cs[i][j]-u
                [i]);
        }
    L=R=vector<int>(n, -1);
    fore(i,0,n)fore(j,0,n)
      if(R[j]==-1&&zero(cs[i][j]-u[i]-v[j])){
                L[i]=j;R[j]=i;
                mat++;
                break;
            }
    for(;mat<n;mat++){
        int s=0, j=0, i;
        while(L[s] != -1)s++;
        fill(all(dad),-1);
            fill(all(sn),0);
        fore(k,0,n)ds[k]=cs[s][k]-u[s]-v[k];
        for(;;){
            j = -1;
            fore(k,0,n)if(!sn[k]&&(j==-1||ds[k
                ]<ds[j]))j=k;
            sn[j] = 1; i = R[j];
            if(i == -1) break;
            fore(k,0,n)if(!sn[k]){
                auto new_ds=ds[j]+cs[i][k]-u[i
                    ]-v[k];
                if(ds[k] > new_ds){ds[k]=new_ds
                    ;dad[k]=j;}
            }
        }
        fore(k,0,n)if(k!=j&&sn[k]){
                auto w=ds[k]-ds[j];
                v[k]+=w,u[R[k]]-=w;
            }
        u[s] += ds[j];
        while(dad[j]>=0){
                int d = dad[j];
                R[j]=R[d];
                L[R[j]]=j;j=d;
            }
        R[j]=s;L[s]=j;
    }
    td value=0;
        fore(i,0,n)value+=cs[i][L[i]];
    return value;
  }
};
```

6.3 Dinic

```
// Min cut: nodes with dist>=0 vs nodes with
    dist<0
// Matching MVC: left nodes with dist<0 + right
    nodes with dist>0
struct Dinic{
  int nodes,src,dst;
  vector<int> dist,q,work;
  struct edge {int to,rev;lli f,cap;};
  vector<vector<edge>> g;
  Dinic(int x):nodes(x),g(x),dist(x),q(x),work(
      x){}
  void add_edge(int s, int t, lli cap){
    g[s].pb((edge){t,sz(g[t]),0,cap});
    g[t].pb((edge){s,sz(g[s])-1,0,0});
  }
  bool dinic_bfs(){
    fill(all(dist),-1);dist[src]=0;
    int qt=0;q[qt++]=src;
    for(int qh=0;qh<qt;qh++){
      int u=q[qh];
      fore(i,0,sz(g[u])){
        edge &e=g[u][i];int v=g[u][i].to;
        if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u
            ]+1,q[qt++]=v;
      }
    }
    return dist[dst]>=0;
  }
  lli dinic_dfs(int u, lli f){
    if(u==dst)return f;
    for(int &i=work[u];i<sz(g[u]);i++){
      edge &e=g[u][i];
      if(e.cap<=e.f)continue;
      int v=e.to;
      if(dist[v]==dist[u]+1){
        lli df=dinic_dfs(v,min(f,e.cap-e.f));
        if(df>0){e.f+=df;g[v][e.rev].f-=df;
            return df;}
      }
    }
    return 0;
  }
  lli max_flow(int _src, int _dst){
    src=_src;dst=_dst;
    lli result=0;
    while(dinic_bfs()){
      fill(all(work),0);
      while(lli delta=dinic_dfs(src,INF))result
          +=delta;
    }
    return result;
  }
};
```

6.4 Dinic Crystal

```
template <typename flow_t> struct Dinic {
    const flow_t INF;
    struct edge {int to;flow_t cap;int rev;bool
         isrev;int idx;};
    vector<vector<edge>> graph;
    vector<int> min_cost, iter;
    Dinic(int V) : INF(numeric_limits<flow_t>::
        max()), graph(V) {}
    void add(int from, int to, flow_t cap, int
        idx = -1) {
        graph[from].emplace_back((edge){to, cap
            , (int)graph[to].size(), false, idx
            });
        graph[to].emplace_back((edge){from, 0,
            (int)graph[from].size() - 1, true,
            idx});
    }
    bool bfs(int s, int t) {
        min_cost.assign(graph.size(), -1);
```

```
        queue<int> que;
        min_cost[s] = 0;
        que.push(s);
        while(!que.empty() && min_cost[t] ==
            -1) {
            int p = que.front();
            que.pop();
            for(auto &e : graph[p]) {
                if(e.cap > 0 && min_cost[e.to]
                    == -1) {
                    min_cost[e.to] = min_cost[p
                        ] + 1;
                    que.push(e.to);
                }
            }
        }
        return min_cost[t] != -1;
    }
    flow_t dfs(int idx, const int t, flow_t
        flow) {
        if(idx == t) return flow;
        for(int &i = iter[idx]; i < graph[idx].
            size(); i++) {
            edge &e = graph[idx][i];
            if(e.cap > 0 && min_cost[idx] <
                min_cost[e.to]) {
                flow_t d = dfs(e.to, t, min(
                    flow, e.cap));
                if(d > 0) {
                    e.cap -= d;
                    graph[e.to][e.rev].cap += d
                        ;
                    return d;
                }
            }
        }
        return 0;
    }
    flow_t max_flow(int s, int t) {
        flow_t flow = 0;
        while(bfs(s, t)) {
            iter.assign(graph.size(), 0);
            flow_t f = 0;
            while((f = dfs(s, t, INF)) > 0)
                flow += f;
        }
        return flow;
    }
    void output() {
        for(int i = 0; i < graph.size(); i++) {
            for(auto &e : graph[i]) {
                if(e.isrev) continue;
                auto &rev_e = graph[e.to][e.rev
                    ];
                cout << i << "->" << e.to << "␣
                    (flow:␣" << rev_e.cap << "/
                    " << e.cap + rev_e.cap << "
                    )" << ENDL;
            }
        }
    }
};
```

6.5 Min cost max flow

```
template <typename tf, typename tc>struct MCF{
int n;
  tf INFFLOW;
  tc INFCOST;
  vector<tc> prio, pot;
  vector<tf> curflow;
  vector<int> prevedge,prevnode;
  priority_queue<pair<tc, int>, vector<pair<tc,
       int>>, greater<pair<tc, int>>> q;
  struct edge{int to, rev; tf f, cap; tc cost
      ;};
  vector<vector<edge>> g;
  MCF(int n):n(n),prio(n),curflow(n),prevedge(n
      ),prevnode(n),pot(n),g(n){
    INFFLOW=numeric_limits<tf>::max() / 2;
    INFCOST=numeric_limits<tc>::max() / 2;
  }
  void add(int s, int t, tf cap, tc cost) {
    g[s].pb((edge){t,sz(g[t]),0,cap,cost});
    g[t].pb((edge){s,sz(g[s])-1,0,0,-cost});
  }
  pair<tf,tc> get_flow(int s, int t) {
    tf flow=0; tc flowcost=0;
    while(1){
      q.push({0, s});
      fill(all(prio),INFCOST);
      prio[s]=0; curflow[s]=INFFLOW;
      while(!q.empty()) {
        auto cur=q.top();
        tc d=cur.f;
        int u=cur.s;
        q.pop();
        if(d!=prio[u]) continue;
        for(int i=0; i<sz(g[u]); ++i) {
          edge &e=g[u][i];
          int v=e.to;
          if(e.cap<=e.f) continue;
          tc nprio=prio[u]+e.cost+pot[u]-pot[v
              ];
          if(prio[v]>nprio) {
            prio[v]=nprio;
            q.push({nprio, v});
            prevnode[v]=u; prevedge[v]=i;
            curflow[v]=min(curflow[u], e.cap-e.
                f);
          }
        }
      }
      if(prio[t]==INFCOST) break;
      fore(i,0,n) pot[i]+=prio[i];
      tf df=min(curflow[t], INFFLOW-flow);
      flow+=df;
      for(int v=t; v!=s; v=prevnode[v]) {
        edge &e=g[prevnode[v]][prevedge[v]];
        e.f+=df; g[v][e.rev].f-=df;
        flowcost+=df*e.cost;
      }
    }
    return {flow,flowcost};
  }
};
```

6.6 Min cost max flow cycles

```cpp
typedef int tf;
typedef int tc;
const tc INFCOST = numeric_limits<tc>::max()/2;
const int scale=2;
struct mcSFlow{
  int n, s, t; tc eps;
  vector<int> isq, cur, co; vector<tf> ex;
      vector<tc> h; vector<vector<int>> hs;
  struct edge{int to, rev; tf f; tc c;};
  vector<vector<edge>> g;
  mcSFlow(int n,int s,int t):eps(0),n(n),s(s),t
      (t),g(n){}
  void add_edge(int a, int b, tc cost, tf cap){
    if(a==b){assert(cost>=0); return;}
    cost*=n; eps = max(eps, abs(cost));
    g[a].pb({b, sz(g[b]), cap, cost});
    g[b].pb({a, sz(g[a])-1, 0, -cost});
  }
  void add_flow(edge& e, tf f) {
    edge &back = g[e.to][e.rev];
    if (!ex[e.to] && f) hs[h[e.to]].push_back(e
        .to);
    e.f -= f; ex[e.to] += f;
    back.f += f; ex[back.to] -= f;
  }
  tf max_flow() {
    ex.assign(n, 0), h.assign(n, 0), co.assign
        (2*n, 0), cur.assign(n, 0), hs.resize
        (2*n);
    h[s] = n, ex[t] = 1, co[0] = n - 1;
    for(auto &e:g[s]) add_flow(e, e.f);
    if(hs[0].size()) for (int hi=0;hi>=0;) {
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ex[u] > 0) { // discharge u
        if (cur[u] == g[u].size()) {
          h[u] = 1e9;
          fore(i,0,sz(g[u])){
            auto &e = g[u][i];
            if (e.f && h[u] > h[e.to]+1) h[u] =
                h[e.to]+1, cur[u] = i;
          }
          if(hi==n)break;
          if (++co[h[u]] && !--co[hi] && hi < n
              ) {
            fore(i,0,n) if (hi < h[i] && h[i] <
                n){
              --co[h[i]];
              h[i] = n + 1;
            }
          }
          hi = h[u];
        } else if (g[u][cur[u]].f && h[u] == h[
            g[u][cur[u]].to]+1) {
          add_flow(g[u][cur[u]], min(ex[u], g[u
              ][cur[u]].f));
        } else ++cur[u];
          }
      while (hi>=0 && hs[hi].empty()) --hi;
    }
    return -ex[s];
  }
  void push(edge &e, tf amt){
    if(e.f < amt) amt=e.f;
    e.f-=amt; ex[e.to]+=amt;
    g[e.to][e.rev].f+=amt; ex[g[e.to][e.rev].to
        ]-=amt;
  }
  void relabel(int vertex){
    tc newHeight = -INFCOST;
    fore(i,0,sz(g[vertex])){
      edge const&e = g[vertex][i];
      if(e.f && newHeight < h[e.to]-e.c){
        newHeight = h[e.to] - e.c;
        cur[vertex] = i;
      }
    }
    h[vertex] = newHeight - eps;
  }
  pair<tf, tc> minCostMaxFlow(){
    tc retCost = 0;
    fore(i,0,n) for(edge &e:g[i]) retCost += e.
        c*(e.f);
    tf retFlow = max_flow();
    h.assign(n, 0); ex.assign(n, 0); isq.assign
        (n, 0); cur.assign(n,0);
    queue<int> q;
    for(;eps;eps>>=scale){
      fill(cur.begin(), cur.end(), 0);
      fore(i,0,n) for(auto &e:g[i])
        if(h[i] + e.c - h[e.to] < 0 && e.f)
            push(e, e.f);
      fore(i,0,n) if(ex[i]>0)q.push(i),isq[i
          ]=1;
      while(!q.empty()){
        int u=q.front();q.pop();
        isq[u]=0;
        while(ex[u]>0){
          if(cur[u] == g[u].size()) relabel(u);
          for(int &i=cur[u], max_i = g[u].size
              ();i<max_i;++i){
            edge &e=g[u][i];
            if(h[u] + e.c - h[e.to] < 0){
              push(e, ex[u]);
              if(ex[e.to]>0 && isq[e.to]==0) q.
                  push(e.to), isq[e.to]=1;
              if(ex[u]==0) break;
            }
          }
        }
      }
      if(eps>1 && eps>>scale==0) eps = 1<<scale
          ;
    }
    fore(i,0,n) for(edge &e:g[i])retCost -= e.c
        *(e.f);
    return make_pair(retFlow, retCost/2/n);
  }
  tf getFlow(edge const &e){
    return g[e.to][e.rev].f;
  }
};
```

6.7 Useful stuff

- # of disjoint s-t paths = min-cut
- $\forall X : |Vecinos(X)| >= |X| \iff$ perfect matching
- |Largest antichain| = |smallest chain decomposition|

- $|MaxMatching| = |minVertexCover|$
- Rebuild cover $(A - Z) \cup (B \cap Z)$ Z = unmatched from L + reachable by alternating paths

```
vector<bool> L, R;
void dfs2(int x, int part) {
  if(!part&&L[x]){L[x]=0;for(auto v:g[x])dfs2
      (v,!part);}
  else if(part&&!R[x])R[x]=1,dfs2(mt[x],!part
      );
}
void cover() {L.assign(n, 1); R.assign(m, 0);
  fore(i,0,n)if(mt2[i]<0)dfs2(i,0);
}
```

- Vertices in all matchings. Orient matched edges L-¿R, and others R-¿L, v can be omitted if:
  - v is unmatched
  - v can be reached from an unmatched vertex ON ITS SIDE
  - v can reach an unmatched vertex ON ITS SIDE
- Circulation with demands
  - add new source s and sink t
  - if $d(v) < 0$, $add\_edge(s, v, -d(v))$
  - if $d(v) > 0$, $add\_edge(v, t, d(v))$
  - There is circulation if the all s,t arcs saturate
- Circulation with demans and lowerbounds (l)
  - new $cap(e)$ = $cap(e)$ - $l(e)$
  - new $d(v)$ = $d(v)$ + sum of $l(e)$ out - sum of $l(e)$ in

7 Other

7.1 Mo's algorithm

```
int n,sq,nq; // array size, sqrt(array size), #
    queries
struct qu{int l,r,id;};
qu qs[MAXN]; lli ans[MAXN]; // ans[i] = answer
    to ith query
bool qcomp(const qu &a, const qu &b){
    if(a.l/sq!=b.l/sq) return a.l<b.l;
    return (a.l/sq)&1?a.r<b.r:a.r>b.r;
}
void mos(){
    fore(i,0,nq)qs[i].id=i;
    sq=sqrt(n)+.5; sort(qs,qs+nq,qcomp); int l
        =0,r=0; init();
    fore(i,0,nq){
        qu q=qs[i];
        while(l>q.l)add(--l);
        while(r<q.r)add(r++);
        while(l<q.l)remove(l++);
        while(r>q.r)remove(--r);
        ans[q.id]=get_ans();
    }
}
```

7.2 Other stuff

```
// double inf
const double DINF=numeric_limits<double>::
    infinity();
// Custom comparator for set/map
struct comp {
  bool operator()(const double& a, const double
      & b) const {
    return a+EPS<b;}
};
set<double,comp> w; // or map<double,int,comp>
// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for (int s=0;s=s-m&m;)   // Increasing order
// Return the numbers the numbers of 1-bit in x
int __builtin_popcount (unsigned int x)
// Returns the number of trailing 0-bits in x.
    x=0 is undefined.
int __builtin_ctz (unsigned int x)
// Returns the number of leading 0-bits in x. x
    =0 is undefined.
int __builtin_clz (unsigned int x)
// x of type long long just add 'll' at the end
     of the function.
int __builtin_popcountll (unsigned long long x)
// Get the value of the least significant bit
    that is one.
v=(x&(-x))
```

7.3 Max number of divisors up to 10^n

```
(0,1,1) (1,4,6) (2,12,60) (3,32,840)
    (4,64,7560) (5,128,83160) (6,240,720720)
    (7,448,8648640) (8,768,73513440)
    (9,1344,735134400) (10,2304,6983776800)
    (11,4032,97772875200)
    (12,6720,963761198400)
    (13,10752,9316358251200)
    (14,17280,97821761637600)
    (15,26880,866421317361600)
    (16,41472,8086598962041600)
    (17,64512,74801040398884800)
    (18,103680,897612484786617600)
```

7.4 Ideas

Meet in the Middle; Gauss Elimination (Z_p); Interpolate; Centroid; HLD; SQRT/Mos; Biconnected/AP/Bridge; Matrix Exp; Offline; D&C; EulerTour; DP Optimization(Aliens,Knuth,etc); SparseTable; Strings; 2SAT; Brute Force; 2k̂ jumps; Small-to-large;randomize;ver diferencias;berlekamp; !!! INICIALIZAR, REVISAR COMENTARIOS, COTAS, INT128/LL, VER N=1 !!!