Team notebook

Universidad Mayor de San Simón - Club de frontón 2880

March 17, 2023

Contents

1 data structures

1.1 BIT 2D

```
#include <iostream>

using namespace std;

const int tam = 1000;

int BIT[tam][tam];
int n, m;

void update(int row, int col, int val)
{
	row++; col++;
	for (int i = row; i <= n; i += (i & -i))
	{
		for (int j = col; j <= m; j += (j & -j))
		{
			BIT[i][j] += val;
		}
	}
}

int query(int row, int col)
{
	int res = 0;
	row++; col++;
	for (int i = row; i > 0; i -= (i & -i))
	{
		for (int j = col; j > 0; j -= (j & -j))
		{
			res += BIT[i][j];
		}
	}
	return res;
}
```

1.2 Binary Indexed (Fenwick) Tree

```
int ft[tam+1]; // for more dimensions, make ft multi-dimensional
void update(int pos, int val){ // add val to pos-th element (0-based)
	// add extra fors for more dimensions
	for(int i=pos+1;i<=tam;i+=i&-i)ft[i]+=val;
}
int get(int pos){ // get sum of range [0,pos)
	int r=0;
	// add extra fors for more dimensions
	for(int i=pos;i;i-=i&-i)r+=ft[i];
	return r;
}
int query(int b, int e){ // get sum of range [b,e) (0-based)
	return get(e)-get(b);
}
```

1.3 HLD LCA con heavy paths

```
/*
Nota:
- el segment tree no esta implementado, solo HLD

Uso:
- poner grafo en G
- correr init(N)
- para cada nodo v -> segBase[pos[v]] = val[v] (segBase es un nuevo
    arreglo)
- inicializar segment tree para [0, N-1]
- implementar la funcion segQuery(int l, int r)
*/
```

```
vector<vi> G;
vi parent, depth, heavy, head, pos;
int curPos;

int dfs(int v) {
    int size = 1;
    int maxChildrenSize = 0;
    for (int c : G[v]) {
        if (c == parent[v]) continue;
        parent[c] = v;
        depth[c] = depth[v] + 1;
        int cSize = dfs(c);
        size += cSize;
        if (cSize > maxChildrenSize) {
            maxChildrenSize = cSize;
            heavy[v] = c;
        }
    }
    return size;
}

void decompose(int v, int h) {
    head[v] = h;
    pos[v] = curPos++;
    if (heavy[v] != -1) {
        decompose(heavy[v], h);
    }
    for (int c : G[v]) {
        if (c != parent[v] && c != heavy[v]) {
            decompose(c, c);
        }
    }
}

void init(int nodes) {
    parent.assign(nodes, -1);
    depth.resize(nodes);
    heavy.assign(nodes, -1);
    head.resize(nodes);
    pos.resize(nodes);
    curPos = 0;
    // Raiz 0
    dfs(0);
    decompose(0, 0);
}
```

```
int segQuery(int l, int r);

int hldQuery(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]]) {
            swap(a, b);
        }
        int pathMax = segQuery(pos[head[b]], pos[b]);
        res = max(pathMax, res);
    }
    if (depth[a] > depth[b]) swap(a, b);
    int lastPath = segQuery(pos[a], pos[b]);
    res = max(res, lastPath);
    return res;
}
```

1.4 Persistent Treap

```
#include <bits/stdc++.h>
#define f first
#define s second
#define pb push_back
#define mp make_pair

#define pitem item*
#define tam 203456
#define tamN 30000000
using namespace std;
typedef long long ll;
struct item
{
    ll tot;
    int key,cnt;
    pitem l;
    pitem r;
    item(){l=r=NULL;}
    item(int key):key(key),l(0),r(0){
        tot=key;
        cnt=1;
    }
};
```

```
int cnt(pitem t){return t?t->cnt:0;}
ll gtot(pitem t){return t?t->tot:0;}
void upd_cnt(pitem t)
{
    if (!t)return;
    t->cnt=cnt(t->l)+cnt(t->r)+1;
    t->tot=t->key+gtot(t->l)+gtot(t->r);
}
void splitat(pitem newT,pitem t,pitem &l,pitem &r,int key)
{
    if (!t)
        return void(l=r=NULL);
    (*newT)=(*t);//.-.
    int cur_at=cnt(t->l);
    if (cur_at>key)
        newT->l=new item(),splitat(newT->l,t->l,l,newT->l,key),r=newT;
    else
        newT->r=new
            item(),splitat(newT->r,t->r,newT->r,r,key-cur_at-1),l=newT;
    upd_cnt(newT);
}
void merge(pitem &newT,pitem l,pitem r)//.-.
{
    if (!l && !r)  return void(newT=NULL);
    if (!l||!r)
        return void((*newT)=l?(*l):(*r));

    int prior=rand()%(cnt(l)+cnt(r));//andrew_part
    if (prior<cnt(l))
        (*newT)=(*l),newT->r=new item(),merge(newT->r,l->r,r);
    else
        (*newT)=(*r),newT->l=new item(),merge(newT->l,l,r->l);
    upd_cnt(newT);
}
ll query(pitem t,int iz,int der,int &riz,int &rder)
{
    if (!t)return 0;
    if (iz>rder|| der<riz)return 0;
    if (iz>=riz&& der<=rder)
        {
            return t->tot;
        }
    return query(t->l,iz,cnt(t->l)+iz,riz,rder)+
           query(t->r,cnt(t->l)+iz+1,der,riz,rder)+(((cnt(t->l)+iz>=riz&&
                cnt(t->l)+iz<=rder)?t->key:0));
}
```

1.5 Queue with min

```
struct quemin
{
        stack<pair<int,int>> bo, to;
        void push(int n)
        {
                if(bo.empty())
                        bo.push(mp(n, n));
                else
                        bo.push(mp(n, min(bo.top().s, n)));
        }
        void pop()
        {
                if(to.empty())
                {
                        while(!bo.empty())
                        {
                                if(to.empty())
                                        to.push(mp(bo.top().f, bo.top().f));
                                else
                                        to.push(mp(bo.top().f,
                                            min(bo.top().f, to.top().s)));
                                bo.pop();
                        }
                }
                to.pop();
        }
        int mini()
        {
                int mini = MOD;
                if(!bo.empty())
                        mini = bo.top().s;
                if(!to.empty())
                        mini = min(mini, to.top().s);
                return mini;
        }
};

struct quemin
{
```

```
	pair<int,int> bo[100010], to[100010];
	int boto = -1, toto = -1, ax;
	void push(int n)
	{
		ax = boto + 1;
		if(boto == -1)
			bo[ax] = mp(n, n);
		else
			bo[ax] = mp(n, min(bo[boto].s, n));
		boto++;
	}
	void pop()
	{
		if(toto == -1)
		{
			while(boto > -1)
			{
				ax = toto + 1;
				if(toto == -1)
					to[ax] = mp(bo[boto].f, bo[boto].f);
				else
					to[ax] = mp(bo[boto].f,
					    min(bo[boto].f, to[toto].s));
				toto++;
				boto--;
			}
		}
		if(toto > -1)
			toto--;
	}
	int mini()
	{
		int mini = MOD;
		if(boto > -1)
			mini = bo[boto].s;
		if(toto > -1)
			mini = min(mini, to[toto].s);
		return mini;
	}
};
```

1.6 Segment Tree Iterativo

```
const int N = 1e5; // limit for array size
int n; // array size
int t[2 * N];

void build() { // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
}

void modify(int p, int value) { // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
}

int query(int l, int r) { // sum on interval [l, r)
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res += t[l++];
    if (r&1) res += t[--r];
  }
  return res;
}
```

1.7 Sparse Table

```
const int tam = 1000010;
const int logTam = 21;
int n;
int ar[tam], table[logTam][tam];
void inispar()
{
	fore(i, 0, n) table[0][i] = ar[i];
	for(int k = 0; (1 << k) < n; k++)
		for(int i = 0; i + (1 << k) < n; i++)
			table[k + 1][i] = min(table[k][i], table[k][i + (1
			    << k)]);
}
int query(int b, int e)
{
	int lev = 31 - __builtin_clz(e - b + 1);
	return min(table[lev][b], table[lev][e - (1 << lev) + 1]);
}
```

1.8 UnionFind

```
struct unionFind {
  vi p;
  unionFind(int n) : p(n, -1) {}
  int findParent(int v) {
    if (p[v] == -1) return v;
    return p[v] = findParent(p[v]);
  }
  bool join(int a, int b) {
    a = findParent(a);
    b = findParent(b);
    if (a == b) return false;
    p[a] = b;
    return true;
  }
};
```

1.9 disjoint$_{i}$ntervals

```
// 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);
              segs.insert(v);
       }
};
```

1.10 kdtree

```
const int maxn=200005;

struct kdtree
{
       int xl,xr,yl,yr,zl,zr,max,flag; // flag=0:x axis 1:y 2:z
} tree[5000005];

int N,M,lastans,xq,yq;
int a[maxn],pre[maxn],nxt[maxn];
int x[maxn],y[maxn],z[maxn],wei[maxn];
int xc[maxn],yc[maxn],zc[maxn],wc[maxn],hash[maxn],biao[maxn];

bool cmp1(int a,int b)
{
       return x[a]<x[b];
}

bool cmp2(int a,int b)
{
       return y[a]<y[b];
}

bool cmp3(int a,int b)
{
       return z[a]<z[b];
}

void makekdtree(int node,int l,int r,int flag)
{
       if (l>r)
       {
              tree[node].max=-maxlongint;
              return;
       }
       int xl=maxlongint,xr=-maxlongint;
       int yl=maxlongint,yr=-maxlongint;
       int zl=maxlongint,zr=-maxlongint,maxc=-maxlongint;
       for (int i=l;i<=r;i++)
              xl=min(xl,x[i]),xr=max(xr,x[i]),
              yl=min(yl,y[i]),yr=max(yr,y[i]),
              zl=min(zl,z[i]),zr=max(zr,z[i]),
              maxc=max(maxc,wei[i]),
              xc[i]=x[i],yc[i]=y[i],zc[i]=z[i],wc[i]=wei[i],biao[i]=i;
       tree[node].flag=flag;
       tree[node].xl=xl,tree[node].xr=xr,tree[node].yl=yl;
       tree[node].yr=yr,tree[node].zl=zl,tree[node].zr=zr;
       tree[node].max=maxc;
       if (l==r) return;
       if (flag==0) sort(biao+l,biao+r+1,cmp1);
       if (flag==1) sort(biao+l,biao+r+1,cmp2);
       if (flag==2) sort(biao+l,biao+r+1,cmp3);
```

```
        for (int i=l;i<=r;i++)
                x[i]=xc[biao[i]],y[i]=yc[biao[i]],
                z[i]=zc[biao[i]],wei[i]=wc[biao[i]];
        makekdtree(node*2,l,(l+r)/2,(flag+1)%3);
        makekdtree(node*2+1,(l+r)/2+1,r,(flag+1)%3);
}

int getmax(int node,int xl,int xr,int yl,int yr,int zl,int zr)
{
        xl=max(xl,tree[node].xl);
        xr=min(xr,tree[node].xr);
        yl=max(yl,tree[node].yl);
        yr=min(yr,tree[node].yr);
        zl=max(zl,tree[node].zl);
        zr=min(zr,tree[node].zr);
        if (tree[node].max==-maxlongint) return 0;
        if ((xr<tree[node].xl)||(xl>tree[node].xr)) return 0;
        if ((yr<tree[node].yl)||(yl>tree[node].yr)) return 0;
        if ((zr<tree[node].zl)||(zl>tree[node].zr)) return 0;
        if ((tree[node].xl==xl)&&(tree[node].xr==xr)&&
                (tree[node].yl==yl)&&(tree[node].yr==yr)&&
                (tree[node].zl==zl)&&(tree[node].zr==zr))
        return tree[node].max;
        else
        return max(getmax(node*2,xl,xr,yl,yr,zl,zr),
                                getmax(node*2+1,xl,xr,yl,yr,zl,zr));
}

int main()
{
        // N 3D-rect with weights
        // find the maximum weight containing the given 3D-point
        return 0;
}
```

1.11 lazy creation

```
struct st
{
        int val;
        st* r;
        st* l;
        st() {r = l = NULL; val = 0;}
        st(int v) {r = l = NULL; val = v;}
        st(st* L, st* R) {l = L; r = R, val = l->val + r->val;}
};
typedef st* pst;
void update(pst &t, int b, int e, int pos, int val)
{
    if(!t) t = new st();
        if(b == e)
    {
                t->val += val;
        return;
    }
        int mid = (b + e) / 2;
        if(mid >= pos)
                update(t->l, b, mid, pos, val);
    else
        update(t->r, mid + 1, e, pos, val);
    t->val = (t->l?t->l->val:0) + (t->r?t->r->val:0);
}
int query(pst t, int b, int e, int i, int j)
{
    if(!t)
        return 0;
        if(b >= i && e <= j)
                return t->val;
        int mid = (b + e) / 2;
        if(mid >= j)
                return query(t->l, b, mid, i, j);
        if(mid < i)
                return query(t->r, mid +1, e, i, j);
        return query(t->l, b, mid, i, j) + query(t->r, mid +1, e, i, j);
}
```

1.12 lazy segment tree

```
int ar[tam], t[4 * tam], l[4 * tam];
void push(int b, int e, int node)
{
    if(l[node])
    {
        t[node] += l[node];
        if(b < e)
```

```
            l[node * 2 + 1] += l[node], l[node * 2 + 2] += l[node];
        l[node] = 0;
    }
}
void init(int b, int e, int node)
{
    if(b == e)
    {
        t[node] = ar[b];
        return;
    }
    index;
    init(b, mid, l);
    init(mid + 1, e, r);
    t[node] = max(t[l], t[r]);
}
int query(int b, int e, int node, int i, int j)
{
    push(b, e, node);
    if(b >= i && e <= j)
        return t[node];
    index;
    if(mid >= j)
        return query(b, mid, l, i, j);
    if(mid < i)
        return query(mid + 1, e, r, i, j);
    return max(query(b, mid, l, i, j), query(mid + 1, e, r, i, j));
}
void update(int b, int e, int node, int i, int j, int val)
{
    if(b > e) return;
    push(b, e, node);
    if(e < i || b > j)
        return;
    if(b >= i && e <= j)
    {
        l[node] += val;
        push(b, e, node);
        return;
    }
    index;
    update(b, mid, l, i, j, val);
    update(mid + 1, e, r, i, j, val);
    t[node] = max(t[l], t[r]);
}
```

1.13 lichao

```
#include <cstdio>
#include <vector>
#define INF 0x3f3f3f3f3f3f3f3f
#define MAXN 1009
using namespace std;

typedef long long ll;

/*
 * LiChao Segment Tree
 */

class LiChao {
        vector<ll> m, b;
        int n, sz; ll *x;
#define gx(i) (i < sz ? x[i] : x[sz-1])
        void update(int t, int l, int r, ll nm, ll nb) {
                ll xl = nm * gx(l) + nb, xr = nm * gx(r) + nb;
                ll yl = m[t] * gx(l) + b[t], yr = m[t] * gx(r) + b[t];
        if (yl >= xl && yr >= xr) return;
                if (yl <= xl && yr <= xr) {
                        m[t] = nm, b[t] = nb; return;
                }
                int mid = (l + r) / 2;
                update(t<<1, l, mid, nm, nb);
                update(1+(t<<1), mid+1, r, nm, nb);
        }
public:
        LiChao(ll *st, ll *en) : x(st) {
                sz = int(en - st);
                for(n = 1; n < sz; n <<= 1);
                m.assign(2*n, 0); b.assign(2*n, -INF);
        }
        void insert_line(ll nm, ll nb) {
                update(1, 0, n-1, nm, nb);
        }
        ll query(int i) {
                ll ans = -INF;
                for(int t = i+n; t; t >>= 1)
                        ans = max(ans, m[t] * x[i] + b[t]);
                return ans;
        }
};
```

```
/*
 * UVa 12524
 */

ll w[MAXN], x[MAXN], A[MAXN], B[MAXN], dp[MAXN][MAXN];

int main(){
	int N, K;
	while(scanf("%d %d", &N, &K)!=EOF) {
		for(int i=0; i<N; i++){
			scanf("%lld %lld", x+i, w+i);
			A[i] = w[i] + (i>0 ? A[i-1] : 0);
			B[i] = w[i]*x[i] + (i>0 ? B[i-1] : 0);
			dp[i][1] = x[i]*A[i] - B[i];
		}
		for(int k=2; k<=K; k++){
			dp[0][k] = 0;
	    LiChao lc(x, x+N);
			for(int i=1; i<N; i++){
				lc.insert_line(A[i-1], -dp[i-1][k-1]-B[i-1]);
				dp[i][k] = x[i]*A[i] - B[i] - lc.query(i);
			}
		}
		printf("%lld\n", dp[N-1][K]);
	}
	return 0;
}
```

1.14 link-cut

```
//no querys on path
//Most changes required by problems of linkcut with values on vertex and
//query on path are usually only on the 4 lines below
const int N_DELTA = 0, N_VALUE = 0;
inline int modifyOp(int x, int y){return x+y;}
inline int queryOp(int lval, int rval){return lval + rval;}
inline int dOnSeg(int d, int len){return d==N_DELTA ? N_DELTA : d*len;}
//all code below is mostly generic
//join delta with value or another delta
inline int joinVD(int v, int d){ return d==N_DELTA ? v : modifyOp(v, d);}
inline int joinDeltas(int d1, int d2){
  if(d1==N_DELTA)return d2; if(d2==N_DELTA)return d1; return modifyOp(d1,
      d2);
}
//node structure
struct Node_t{
  int sz, nVal, stVal, d;
  bool rev;
  Node_t *ch[2], *p;
  Node_t(int v) : sz(1), nVal(v), stVal(v), d(N_DELTA), rev(0), p(0){
  ch[0]=ch[1]=0;
  }
  bool isRoot(){return !p || (p->ch[0] != this && p->ch[1] != this);}
  void push(){
  if(rev){
    rev=0; swap(ch[0], ch[1]);
    fore(x,0,2)if(ch[x])ch[x]->rev^=1;
  }
  nVal=joinVD(nVal, d); stVal=joinVD(stVal, dOnSeg(d, sz));
  fore(x,0,2)if(ch[x])ch[x]->d=joinDeltas(ch[x]->d, d);
  d=N_DELTA;
  }
  void upd();
};
typedef Node_t* Node;
int getSize(Node r){return r ? r->sz : 0;}
int getstVal(Node r){
  return r ? joinVD(r->stVal, dOnSeg(r->d,r->sz)) : N_VALUE;}
void Node_t::upd(){
  stVal = queryOp(queryOp(getstVal(ch[0]), joinVD(nVal, d)),
      getstVal(ch[1]));
  sz = 1 + getSize(ch[0]) + getSize(ch[1]);
}
//splay related functions
void connect(Node ch, Node p, int isl){if(ch)ch->p=p;
    if(isl>=0)p->ch[1-isl]=ch;}
void rotate(Node x){
  Node p = x->p, g = p->p;
  bool gCh=p->isRoot(), isl = x==p->ch[0];
  connect(x->ch[isl],p,isl);connect(p,x,!isl);connect(x,g,gCh?-1:(p==g->ch[0]));
  p->upd();
}
void splay(Node x){
  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->ch[0])==(p==g->ch[0])? p : x);
  rotate(x);
  }
  x->push(); x->upd();
}
Node expose(Node x){
  Node last=0;
  for(Node y=x; y; y=y->p)splay(y),y->ch[0]=last,y->upd(),last=y;
  splay(x);
  return last;
}
//only new Node_t(v) and the functions below should be used
Node findRoot(Node x){expose(x); while(x->ch[1])x=x->ch[1]; splay(x);
    return x;}
Node lca(Node x, Node y){expose(x); return expose(y);}
void makeRoot(Node x){expose(x); x->rev^=1;}
bool connected(Node x, Node y){if(x==y)return 1; expose(x);expose(y);
    return x->p;}
void link(Node x, Node y){makeRoot(x); x->p=y;}
void cut(Node x, Node y){makeRoot(x); expose(y); y->ch[1]->p = 0;
    y->ch[1]=0;}
int query(Node x, Node y){makeRoot(x); expose(y); return getstVal(y);}
void modify(Node x, Node y, int d){makeRoot(x); expose(y); y->d =
    joinDeltas(y->d, d);}
```

1.15 merge sort tree

```
// Mergesort Tree - Time <O(nlogn), O(log^2n)> - Memory O(nlogn)
// Mergesort Tree is a segment tree that stores the sorted subarray
// on each node.
vi st[4*N];

void build(int p, int l, int r) {
  if (l == r) { st[p].pb(s[l]); return; }
  build(2*p, l, (l+r)/2);
  build(2*p+1, (l+r)/2+1, r);
  st[p].resize(r-l+1);
  merge(st[2*p].begin(), st[2*p].end(),
        st[2*p+1].begin(), st[2*p+1].end(),
        st[p].begin());
}
int query(int p, int l, int r, int i, int j, int a, int b) {
  if (j < l or i > r) return 0;
  if (i <= l and j >= r)
    return upper_bound(st[p].begin(), st[p].end(), b) -
           lower_bound(st[p].begin(), st[p].end(), a);
  return query(2*p, l, (l+r)/2, i, j, a, b) +
         query(2*p+1, (l+r)/2+1, r, i, j, a, b);
}
```

1.16 $\min_{queue}$

```
// O(1) complexity for all operations, except for clear,
// which could be done by creating another deque and using swap

struct MinQueue {
  int plus = 0;
  int sz = 0;
  deque<pair<int, int>> dq;

  bool empty() { return dq.empty(); }
  void clear() { plus = 0; sz = 0; dq.clear(); }
  void add(int x) { plus += x; } // Adds x to every element in the queue
  int min() { return dq.front().first + plus; } // Returns the minimum
      element in the queue
  int size() { return sz; }

  void push(int x) {
    x -= plus;
    int amt = 1;
    while (dq.size() and dq.back().first >= x)
      amt += dq.back().second, dq.pop_back();
    dq.push_back({ x, amt });
    sz++;
  }

  void pop() {
    dq.front().second--, sz--;
    if (!dq.front().second) dq.pop_front();
  }
};
```

1.17 persistent convex hull trick

```
// add lines of the form kx+m, and query maximum values at points x.
struct per_hull{
    const int logtam = 18;
    vector<int> k, m;
    vector<vector<int>> pars;
    per_hull(){pars.push_back(vi(logtam)), k.push_back(0),
        m.push_back(0);}
    int query(int x, int head)
    {
        auto f = [&](int x, int head) {
            return x * k[head] + m[head];
        };
        for(int i = logtam - 1; i > -1; i--)
            if(f(x, pars[head][i]) < f(x, pars[pars[head][i]][1]))
                head = pars[pars[head][i]][1];
        return f(x, head);
    }
    int add(int k1, int m1, int head)
    {
        for(int i = logtam - 1; i > -1; i--)
        {
            if(pars[head][i] == 0) continue;
            int b = pars[head][i];
            int a = pars[pars[head][i]][1];
            if((m[b] - m[a]) * (k[a] - k1) > (m1 - m[a]) * (k[a] - k[b]))
                head = pars[pars[head][i]][1];
        }
        int node = k.size();
        k.push_back(k1), m.push_back(m1);
        pars.push_back(vector<int>(logtam));
        pars[node][0] = node;
        pars[node][1] = head;
        for(int i = 2; i < logtam; i++)
            pars[node][i] = pars[pars[node][i - 1]][i - 1];
        return node;
    }
};
```

1.18 persistent segment tree pacha

```
struct st
{
        int val;
        st* r;
        st* l;
        st() {r = l = NULL; val = 0;}
        st(int v) {r = l = NULL; val = v;}
        st(st* L, st* R) {l = L; r = R, val = l->val + r->val;}
};
int ar[tam];
typedef st* pst;
pst init(int b, int e)
{
        if(b == e)
                return new st(ar[b]);
        int mid = (b + e) / 2;
        return new st(init(b, mid), init(mid + 1, e));
}
pst update(pst t, int b, int e, int pos, int val)
{
        if(b == e)
                return new st(t->val + val);
        int mid = (b + e) / 2;
        if(mid >= pos)
                return new st(update(t->l, b, mid, pos, val), t->r);
        return new st(t->l, update(t->r, mid + 1, e, pos, val));
}
int query(pst t, int b, int e, int i, int j)
{
        if(b >= i && e <= j)
                return t->val;
        int mid = (b + e) / 2;
        if(mid >= j)
                return query(t->l, b, mid, i, j);
        if(mid < i)
                return query(t->r, mid +1, e, i, j);
        return query(t->l, b, mid, i, j) + query(t->r, mid +1, e, i, j);
}
```

1.19 steiner$_tree$

```
// Steiner-Tree O(2^t*n^2 + n*3^t + APSP)

// N - number of nodes
```

```
// T - number of terminals
// dist[N][N] - Adjacency matrix
// steiner_tree() = min cost to connect first t nodes, 1-indexed
// dp[i][bit_mask] = min cost to connect nodes active in bitmask rooting
    in i
// min{dp[i][bit_mask]}, i <= n if root doesn't matter

int n, t, dp[N][(1 << T)], dist[N][N];

int steiner_tree() {
  for (int k = 1; k <= n; ++k)
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);

  for(int i = 1; i <= n; i++)
    for(int j = 0; j < (1 << t); j++)
      dp[i][j] = INF;
  for(int i = 1; i <= t; i++) dp[i][1 << (i-1)] = 0;

  for(int msk = 0; msk < (1 << t); msk++) {
    for(int i = 1; i <= n; i++) {
      for(int ss = msk; ss > 0; ss = (ss - 1) & msk)
        dp[i][msk] = min(dp[i][msk], dp[i][ss] + dp[i][msk - ss]);

      if(dp[i][msk] != INF)
        for(int j = 1; j <= n; j++)
          dp[j][msk] = min(dp[j][msk], dp[i][msk] + dist[i][j]);
    }
  }

  int mn = INF;
  for(int i = 1; i <= n; i++) mn = min(mn, dp[i][(1 << t) - 1]);
  return mn;
}
```

1.20 treap

```
struct item {
    int key, pri, siz;
    item *l, *r;
    item() {}
    item(int key) : key(key), siz(1), pri(rand()), l(0), r(0) {}
};
typedef item* pitem;
int sz(pitem t) {
    return (t?t->siz:0);
}
void up_sz(pitem t) {
    if(t) t->siz = sz(t->l) + 1 + sz(t->r);
}
void split(pitem t, pitem &l, pitem &r, int val) {
    if(!t) r = l = NULL;
    else if(t->key < val) split(t->r, t->r, r, val), l = t;
    else split(t->l, l, t->l, val), r = t;
    up_sz(t);
}
void merge(pitem &t, pitem l, pitem r) {
    if(!l || !r) t=(l?l:r);
    else if(l->pri >= r->pri) merge(l->r, l->r, r), t = l;
    else merge(r->l, l, r->l),t=r;
    up_sz(t);
}
```

1.21 union find partial persistent

```
/*********************************************************************************
* DSU (DISJOINT SET UNION / UNION-FIND)
    *
* Time complexity: Unite - O(log n)
    *
*                  Find - O(log n)
    *
* Usage: find(node), unite(node1, node2), sz[find(node)]
    *
* Notation: par: vector of parents
    *
*          sz:  vector of subsets sizes, i.e. size of the subset a node
    is in *
*          his: history: time when it got a new parent
    *
*          t: current time
    *
*********************************************************************************

int t, par[N], sz[N], his[N];
```

```
int find(int a, int t){
    if(par[a] == a) return a;
    if(his[a] > t) return a;
    return find(par[a], t);
}

void unite(int a, int b){
    if(find(a, t) == find(b, t)) return;
    a = find(a, t), b = find(b, t), t++;
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b], par[b] = a, his[b] = t;
}

//in main
for(int i = 0; i < N; i++) par[i] = i, sz[i] = 1, his[i] = 0;
```

1.22 union find rollback

```
/**********************************************************************************
* DSU (DISJOINT SET UNION / UNION-FIND)
    *
* Time complexity: Unite - O(alpha n)
    *
*                 Rollback - O(1)
    *
*                 Find - O(alpha n)
    *
* Usage: find(node), unite(node1, node2), sz[find(node)]
    *
* Notation: par: vector of parents
    *
*          sz:  vector of subsets sizes, i.e. size of the subset a node
    is in *
*          sp:  stack containing node and par from last op
    *
*          ss:  stack containing node and size from last op
    *
**********************************************************************************/

int par[N], sz[N];
stack <pii> sp, ss;

int find (int a) { return par[a] == a ? a : find(par[a]); }

void unite (int a, int b) {
    if ((a = find(a)) == (b = find(b))) return;
    if (sz[a] < sz[b]) swap(a, b);
    ss.push({a, sz[a]});
    sp.push({b, par[b]});
    sz[a] += sz[b];
    par[b] = a;
}

void rollback() {
    par[sp.top().st] = sp.top().nd; sp.pop();
    sz[ss.top().st] = ss.top().nd; ss.pop();
}

int main(){
    for (int i = 0; i < N; i++) par[i] = i, sz[i] = 1;
    return 0;
}
```

2 dp

2.1 Divide and conquer

```
/*
DP[i][j] = min( DP[i-1][k] + C[k][j] )
K[i][j] <= K[i][j+1]
*/
ll lastDP[tam], DP[tam];
int C[tam][tam]; // Cambiar a una funcion de costo si pre-procesar ocupa
    mucha memoria
void DC(int b, int e, int KL, int KR) {
        int mid = (b + e) / 2;
        pair<ll, int> best = mp(-1, KL);
        for (int k = KL; k < min(mid, KR+1); k++)
                best = max( best, mp(lastDP[k] + C[k+1][mid], k) );
        DP[mid] = best.first;
        int K = best.second;
        if (b <= mid-1)
                DC(b, mid-1, KL, K);
        if (mid+1 <= e)
```

```
            DC(mid+1, e, K, KR);
}
```

2.2 KnuthOptimization

```
/*Basado en problema: Minimo costo de cortar una barra en K lugares
    pre-definidos
                Costo de corte: largo de la barra que se esta cortando

A[i] -> posicion del i-esimo corte, A[0] = 0, A[n-1] = Largo total
DP[i][j] = min( DP[i][k] + DP[k][j] ) + C[i][j], para i <= k <= j donde
    C[i][j] es el largo de la barra A[i] <-> A[j]
K[i][j-1] <= K[i][j] <= K[i+1][j]*/
// Cambiar C[i][j] a una funcion de costo si pre-calcular todos los
    valores es muy costoso
fore(sz, 0, n) {
  for (int i = 0; i + sz < n; i++) {
    int j = i+sz;
    // CASOS BASE
    if (sz <= 1) { // Barra inexistente o con cero cortes en medio
      DP[i][j] = 0;
      continue;
    }
    if (sz == 2) { // Barra con un solo corte posible en medio
      K[i][j] = i+1;
      DP[i][j] = C[i][j];
      continue;
    }
    int KL = K[i][j-1];
    int KR = K[i+1][j];
    DP[i][j] = INF;
    for (int k = KL; k <= KR; k++) {
      int newVal = DP[i][k] + DP[k][j] + C[i][j];
      if (newVal < DP[i][j]) {
        K[i][j] = k;
        DP[i][j] = newVal;
      }
    }
  }
}
```

2.3 SOS

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
        dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
        for(int i = 0;i < N; ++i){
                if(mask & (1<<i))
                        dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
                else
                        dp[mask][i] = dp[mask][i-1];
        }
        F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
}
```

2.4 lineContainer

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m, and
     query maximum values at points x.
 *  Useful for dynamic programming.
 * Time: O(\log N)
 * Status: tested
 */
#pragma once
//copiado
struct Line {
        mutable ll k, m, p;
        bool operator<(const Line& o) const { return k < o.k; }
        bool operator<(ll x) const { return p < x; }
};
```

```
struct LineContainer : multiset<Line, less<>> {
	// (for doubles, use inf = 1/.0, div(a,b) = a/b)
	const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		return a / b - ((a ^ b) < 0 && a % b); }
	bool isect(iterator x, iterator y) {
		if (y == end()) { x->p = inf; return false; }
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}
	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
		if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
		while ((y = x) != begin() && (--x)->p >= y->p)
			isect(x, erase(y));
	}
	ll query(ll x) {
		assert(!empty());
		auto l = *lower_bound(x);
		return l.k * x + l.m;
	}
};
```

2.5 lis

```
// Longest Increasing Subsequence - O(nlogn)
//
// dp(i) = max j<i { dp(j) | a[j] < a[i] } + 1
//

// int dp[N], v[N], n, lis;

memset(dp, 63, sizeof dp);
for (int i = 0; i < n; ++i) {
  // increasing: lower_bound
  // non-decreasing: upper_bound
  int j = lower_bound(dp, dp + lis, v[i]) - dp;
  dp[j] = min(dp[j], v[i]);
  lis = max(lis, j + 1);
}
```

2.6 matrix-fast-pow

```
typedef vector<vector<ll> > Matrix;
Matrix ones(int n) {
  Matrix r(n,vector<ll>(n));
  fore(i,0,n)r[i][i]=1;
  return r;
}
Matrix operator*(Matrix &a, Matrix &b) {
  int n=SZ(a),m=SZ(b[0]),z=SZ(a[0]);
  Matrix r(n,vector<ll>(m));
  fore(i,0,n)fore(j,0,m)fore(k,0,z)
    r[i][j]+=a[i][k]*b[k][j],r[i][j]%=mod;
  return r;
}
Matrix be(Matrix b, ll e) {
  Matrix r=ones(SZ(b));
  while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
  return r;
}
```

3 fft

3.1 fft iterativo

```
typedef complex<double> cd;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
```

```
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

3.2 fft operations

```
typedef vector<int> poly;
void normalize(poly &p)
{
        while(p.size() > 1 && p.back() == 0) p.pop_back();
}
poly derivate(poly v)
{
        fore(i, 0, v.size() - 1)
                v[i] = 1ll * v[i + 1] * (i + 1) % MOD;
        v.pop_back();
        return v;
}
poly sum(poly a, poly b)
{
        if(a.size() < b.size())
                swap(a, b);
        fore(i, 0, b.size())
                a[i] = (a[i] + b[i]) % MOD;
        return a;
}
//input : B = b0 + b1x + ... + bnx^n with b0, bn != 0 and an integer d  0
//output: First d + 1 terms of 1 / B(x), it is: q0 + q1x + . . . + qdx^d
poly invert(poly b, int d)
{
        if(d == 0) return {pot(b[0], MOD - 2)};
        poly c = invert(b, d / 2);
        b.resize(sz(c) * 2);
        poly q = multiply(c, b);
        q[0] = (q[0] + MOD - 2) % MOD;
        fore(i, 0, q.size())
                q[i] = (-q[i] % MOD + MOD) % MOD;
        q = multiply(c, q);
        q.resize(d + 1);
        return q;
}

//input : A(x) and B(x) polynomials of degree m and n
//output: {Q, R} the quotient and remainder of dividing A by B

pair<poly,poly> divslow(poly &a, poly &b){
        poly q, r = a;
        while(sz(r) >= sz(b)){
                q.pb(1ll * r.back() * pot(b.back(), MOD - 2) % MOD);
                if(q.back()) fore(i, 0, sz(b)){
                        r[sz(r) - i - 1] = (r[sz(r)-i-1] + MOD - 1ll *
                            q.back() * b[sz(b) - i - 1] % MOD) % MOD;
                }
```

```
                r.pop_back();
        }
        reverse(all(q));
        return {q, r};
}

pair<poly, poly> divide(poly a, poly b)
{
        int m = sz(a), n = sz(b), magic = 750;
        if(m < n) return {{0}, a};
        if(min(m - n, n)< magic) return divslow(a, b);
        reverse(all(a));
        reverse(all(b));
        poly q = invert(b, m - n);
        q = multiply(a, q);
        q.resize(m - n + 1, 0);
        reverse(all(q));
        reverse(all(a));
        reverse(all(b));
        poly r = multiply(b, q);
        if(r.size() < a.size())
                r.resize(a.size());
        fore(i, 0, a.size())
                r[i] = ((a[i] - r[i]) % MOD + MOD) % MOD;
        normalize(r);
        return {q, r};
}

//input : A(x) of degree m and a n-uple x = (x1, . . . , xn)
//output: (A(x1), A(x2), . . . , A(xn))
vector<poly> init(poly &x)
{
        int n = sz(x);
        vector<poly> t(2 * n);
        fore(i, n, 2 * n) t[i] = {MOD - x[i - n], 1};
        for(int i = n - 1; i > 0; i--) t[i] = multiply(t[i<<1], t[i<<1|1]);
                return t;
}
poly evaluate(poly a, poly x)
{
        int n = x.size();
        vector<poly> tree = init(x), ans(2 * n);
        ans[1] = divide(a, tree[1]).s;
        fore(i, 2, 2 * n)
                ans[i] = divide(ans[i>>1], tree[i]).s;
        poly r;
        fore(i, 0, n) r.pb(ans[i + n][0]);
        return r;
}
// interpola ps n + 1 puntos para polinomio de grado n
poly interpol(poly x, poly y)
{
        int n = x.size();
        vector<poly> tree = init(x), tree1(2 * n);
        poly d = evaluate(derivate(tree[1]), x);
        fore(i, n, 2 * n)
                tree1[i] = {(int)(1ll * y[i - n] * pot(d[i - n], MOD - 2)
                    % MOD)};
        for(int i = n - 1; i > 0; i--)
                tree1[i] = sum(multiply(tree[i<<1], tree1[i<<1|1]),
                    multiply(tree1[i<<1], tree[i<<1|1]));
        return tree1[1];
}


//para meter sin fft (ineficientes)
// n^2
poly evaluate1(poly a, poly b)
{
        poly res(b.size());
        fore(i, 0, b.size())
        {
                ll x = 1;
                res[i] = 0;
                fore(j, 0, a.size())
                {
                        res[i] = (res[i] + a[j] * x) % MOD;
                        x = 1ll * x * b[i] % MOD;
                }
                res[i] = (res[i] %    MOD + MOD) % MOD;
        }
        return res;
}
//interpolar n^4 creo
poly polo(poly p, int ind, int ini)
{
        poly res = {ini}, a;
        fore(i, 0, p.size())
        {
                if(i != ind)
```

```
		{
			a = {MOD - p[i], 1};
			res = multiply(res, a);
		}
	}
	return res;
}
poly interpol1(poly x, poly y)
{
	poly d = polo(x, -1, 1);
	d = derivate(d);
	d = evaluate1(d, x);
	poly res, a;
	fore(i, 0, y.size())
	{
		a = polo(x, i, 1ll * y[i] * pot(d[i], MOD - 2) % MOD);
		res = sum(res, a);
	}
	return res;
}
```

3.3 fft-operations

```
// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
typedef int tf;
typedef vector<tf> poly;
const tf MOD=998244353,RT=3,MAXN=1<<16;
tf addmod(tf a, tf b){tf r=a+b;if(r>=MOD)r-=MOD;return r;}
tf submod(tf a, tf b){tf r=a-b;if(r<0)r+=MOD;return r;}
tf mulmod(ll a, ll b){return a*b%MOD;}
tf pm(ll a, ll b){
  ll r=1;
  while(b){
    if(b&1) r=mulmod(r,a); b>>=1;
    a=mulmod(a,a);
  }
  return r;
}
tf inv(tf a){return pm(a,MOD-2);}
// 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);
}
CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  fore(i,0,n)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)fore(i,0,n)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=p1.size()+p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
  fore(i,0,m)cp1[i]=0,cp2[i]=0;
  fore(i,0,p1.size())cp1[i]=p1[i];
  fore(i,0,p2.size())cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
  poly res;
  n-=2;
  //fore(i,0,n)res.pb((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))
//Works with NTT. For FFT, just replace addmod,submod,mulmod,inv
poly add(poly &a, poly &b){
  int n=SZ(a),m=SZ(b);
  poly ans(max(n,m));
  fore(i,0,max(n,m)){
    if(i<n) ans[i]=addmod(ans[i],a[i]);
    if(i<m) ans[i]=addmod(ans[i],b[i]);
  }
  while(SZ(ans)>1&&!ans.back())ans.pop_back();
  return ans;
}
poly invert(poly &b, int d){
 poly c = {inv(b[0])};
 while(SZ(c)<=d){
   int j=2*SZ(c);
  auto bb=b; bb.resize(j);
  poly cb=multiply(c,bb);
  fore(i,0,SZ(cb)) 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(SZ(r)>=SZ(b)){
    q.pb(mulmod(r.back(),inv(b.back())));
    if(q.back()) fore(i,0,SZ(b)){
      r[SZ(r)-i-1]=submod(r[SZ(r)-i-1],mulmod(q.back(),b[SZ(b)-i-1]));
    }
    r.pop_back();
  }
  reverse(ALL(q));
  return {q,r};
}
pair<poly,poly> divide(poly &a, poly &b){ //returns {quotient,remainder}
  int m=SZ(a),n=SZ(b),MAGIC=750;
  if(m<n) return {{0},a};
  if(min(m-n,n)<MAGIC)return divslow(a,b);
  poly ap=a; reverse(ALL(ap));
  poly bp=b; reverse(ALL(bp));
  bp=invert(bp,m-n);
  poly q=multiply(ap,bp);
  q.resize(SZ(q)+m-n-SZ(q)+1,0);
  reverse(ALL(q));
  poly bq=multiply(b,q);
  fore(i,0,SZ(bq)) bq[i]=submod(0,bq[i]);
  poly r=add(a,bq);
  return {q,r};
}
vector<poly> tree;
void filltree(vector<tf> &x){
  int k=SZ(x);
  tree.resize(2*k);
  fore(i,k,2*k) 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=SZ(x);
  vector<poly> ans(2*k);
  ans[1]=divide(a,tree[1]).snd;
  fore(i,2,2*k) ans[i]=divide(ans[i>>1],tree[i]).snd;
  vector<tf> r; fore(i,0,k) r.pb(ans[i+k][0]);
  return r;
}
poly derivate(poly &p){
  poly ans(SZ(p)-1);
```

```
  fore(i,1,SZ(p)) 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=SZ(y);
  vector<tf> d=evaluate(p,x);
  vector<poly> intree(2*k);
  fore(i,k,2*k) intree[i]={mulmod(y[i-k],inv(d[i-k]))};
  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];
}
int main(){FIN;
  int m,k; cin>>m>>k;
  int top=max(k,m)+2;
  vector<int> x,y;
  int ac=0;
  fore(i,0,top){
    ac=addmod(ac,pm(i,k));
    x.pb(i); y.pb(ac);
  }
  poly p=interpolate(x,y);
  vector<int> xs;
  fore(i,0,m){
    ll x; cin>>x; x%=MOD;
    xs.pb(x);
  }
  while(SZ(xs)!=top) xs.pb(0);
  vector<int> ans=evaluate(p,xs);
  fore(i,0,m)cout<<ans[i]<<" ";cout<<"\n";
}
```

3.4 fht

```
ll c1[tam+9],c2[tam+9]; // tam must be power of 2 !!
void fht(ll* p, int n, bool inv){
        for(int l = 1; 2 * l <= n; l *= 2)
        for(int i = 0; i < n; i += 2 * l)
        fore(j, 0, l)
        {
                ll 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<ll> multiply(vector<ll>& p1, vector<ll>& p2){
        int n = 1<<(32-__builtin_clz(max(sz(p1), sz(p2)) - 1));
        fore(i, 0, n) c1[i] = 0, c2[i] = 0;
        fore(i, 0, sz(p1)) c1[i] = p1[i];
        fore(i, 0, sz(p2)) c2[i] = p2[i];
        fht(c1, n, false); fht(c2, n, false);
        fore(i, 0, n) c1[i] *= c2[i];
        fht(c1, n, true);
        return vector<ll>(c1, c1 + n);
}
void fht(vector<ll>& p, bool inv) {
    fore(i, 0, sz(p)) c1[i] = p[i];
    fht(c1, sz(p), inv);
    fore(i, 0, sz(p)) p[i] = c1[i];
}
```

3.5 karatsuba

```
typedef ll tp;
// #define add(n,s,d,k) fore(i,0,n)(d)[i]+=(s)[i]*k
#define add(n,s,d,k) fore(i,0,n)(d)[i]+=(s)[i]*k%MOD, (d)[i] = ((d)[i] %
    MOD + MOD) % MOD;
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)fore(i,0,n)fore(j,0,n)r[i+j]+=p[i]*q[j];
  if(n<35)fore(i,0,n)fore(j,0,n)r[i+j]+=p[i]*q[j] % MOD, r[i + j] %= MOD;
  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);
  fore(i,0,p0.size())p[i]=p0[i];
  fore(i,0,p1.size())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;
}
```

3.6 ntt

```
// el modulo debe ser un primo de la forma p = c*2^k + 1
const int mod = 998244353; // c*2^k + 1 = 119*2^23 + 1

// 3 es raiz primitiva de mod, root = 3^c % mod = 3^119 % mod
const int root = 15311432;

// root_1 = root^-1
const int root_1 = 469870224;

// 2^k
const int root_pw = 1 << 23;

int modPow(int b, int e)
{
	if (e == 0) return 1;
	int res = modPow(b, e/2);
	if (e%2 == 1)
		return ((1LL * b * res % mod) * res % mod) % mod;
	else
		return (1LL * res * res % mod) % mod;
}

void fft(vector<int> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        // Pre-calcular inversos modulares para mejorar eficiencia
        int n_1 = modPow(n, mod-2);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}

vector<int> fftMul(vector<int> & a, vector<int> & b)
{
	vector<int> fa(a.begin(), a.end()), fb(b.begin(), b.end());
	int n = 1;
	while (n < a.size() + b.size())
		n <<= 1;
```

```
        fa.resize(n);
        fb.resize(n);

        fft(fa, false);
        fft(fb, false);

        for (int i = 0; i < n; i++)
                fa[i] = (int)(1LL * fa[i] * fb[i] % mod);
        fft(fa, true);

        return fa;
}
```

4 flow

4.1 Algoritmo hungaro de asignacion

```
int n, m;
int a[n+1][m+1]; // matriz de costos, 1-index
vector<int> u(n+1), v(m+1), p(m+1), way(m+1);
for (int i = 1; i <= n; ++i) {
        p[0] = i;
        int j0 = 0;
        vector<int> minv(m + 1, INF);
        vector<char> used(m + 1, false);
        do {
                used[j0] = true;
                int i0 = p[j0], delta = INF, j1;
                for (int j = 1; j <= m; ++j)
                        if ( !used[j] ) {
                                int cur = a[i0][j] - u[i0] - v[j];
                                if ( cur < minv[j] )
                                        minv[j] = cur, way[j] = j0;
                                if ( minv[j] < delta )
                                        delta = minv[j], j1 = j;
                        }
                for (int j = 0; j <= m; ++j)
                        if ( used[j] )
                                u[ p[j] ] += delta, v[j] -= delta;
                        else
                                minv[j] -= delta;
                j0 = j1;
        } while ( p[j0] != 0 );
        do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
        } while ( j0 ) ;
}

vector<int> ans(n+1); // Para cada fila, en que columna esta el resultado.
for (int j = 1; j <= m; ++j)
        ans[ p[j] ] = j;

int cost = -v[0];
```

4.2 max flow

```
struct Dinitz{
    const int INF = 1e9 + 7;
    Dinitz(){}
    Dinitz(int n, int s, int t) {init(n, s, t);}

    void init(int n, int s, int t)
    {
        S = s, T = t;
        nodes = n;
        G.clear(), G.resize(n);
        Q.resize(n);
    }
    struct flowEdge
    {
        int to, rev, f, cap;
    };

    vector<vector<flowEdge> > G;

    // Aade arista (st -> en) con su capacidad
    void addEdge(int st, int en, int cap) {
        flowEdge A = {en, (int)G[en].size(), 0, cap};
        flowEdge B = {st, (int)G[st].size(), 0, 0};
        G[st].pb(A);
        G[en].pb(B);
    }
```

```
int nodes, S, T; // asignar estos valores al armar el grafo G
                 // nodes = nodos en red de flujo. Hacer G.clear();
                     G.resize(nodes);
vi work, lvl;
vi Q;

bool bfs() {
    int qt = 0;
    Q[qt++] = S;
    lvl.assign(nodes, -1);
    lvl[S] = 0;
    for (int qh = 0; qh < qt; qh++) {
        int v = Q[qh];
        for (flowEdge &e : G[v]) {
            int u = e.to;
            if (e.cap <= e.f || lvl[u] != -1) continue;
            lvl[u] = lvl[v] + 1;
            Q[qt++] = u;
        }
    }
    return lvl[T] != -1;
}

int dfs(int v, int f) {
    if (v == T || f == 0) return f;
    for (int &i = work[v]; i < G[v].size(); i++) {
        flowEdge &e = G[v][i];
        int u = e.to;
        if (e.cap <= e.f || lvl[u] != lvl[v] + 1) continue;
        int df = dfs(u, min(f, e.cap - e.f));
        if (df) {
            e.f += df;
            G[u][e.rev].f -= df;
            return df;
        }
    }
    return 0;
}

int maxFlow() {
    int flow = 0;
    while (bfs()) {
        work.assign(nodes, 0);
        while (true) {
            int df = dfs(S, INF);
            if (df == 0) break;
            flow += df;
        }
    }
    return flow;
}
};
```

4.3 min cost flow

```
// O(min(E^2 V ^2,   EVFLOW  ))
struct CheapDinitz{
    const int INF = 1e9 + 7;

    CheapDinitz() {}
    CheapDinitz(int n, int s, int t) {init(n, s, t);}

    int nodes, S, T;
    vi dist;
    vi pot, curFlow, prevNode, prevEdge, Q, inQue;

    struct flowEdge{
        int to, rev, flow, cap, cost;
    };
    vector<vector<flowEdge>> G;
    void init(int n, int s, int t)
    {
        nodes = n, S = s, T = t;
        curFlow.assign(n, 0), prevNode.assign(n, 0), prevEdge.assign(n, 0);
        Q.assign(n, 0), inQue.assign(n, 0);
        G.clear();
        G.resize(n);
    }

    void addEdge(int s, int t, int cap, int cost)
    {
        flowEdge a = {t, (int)G[t].size(), 0, cap, cost};
        flowEdge b = {s, (int)G[s].size(), 0, 0, -cost};
        G[s].pb(a);
        G[t].pb(b);
    }

    void bellmanFord()
```

```
{
    pot.assign(nodes, INF);
    pot[S] = 0;
    int qt = 0;
    Q[qt++] = S;
    for (int qh = 0; (qh - qt) % nodes != 0; qh++)
    {
        int u = Q[qh % nodes];
        inQue[u] = 0;
        for (int i = 0; i < (int)G[u].size(); i++)
        {
            flowEdge &e = G[u][i];
            if (e.cap <= e.flow) continue;
            int v = e.to;
            int newDist = pot[u] + e.cost;
            if (pot[v] > newDist)
            {
                pot[v] = newDist;
                if (!inQue[v])
                {
                    Q[qt++ % nodes] = v;
                    inQue[v] = 1;
                }
            }
        }
    }
}

ii MinCostFlow()
{
    bellmanFord();
    int flow = 0;
    int flowCost = 0;
    while (true) // always a good start for an algorithm :v
    {
        set<ii> s;
        s.insert({0, S});
        dist.assign(nodes, INF);
        dist[S] = 0;
        curFlow[S] = INF;
        while (s.size() > 0)
        {
            int u = s.begin() -> s;
            int actDist = s.begin() -> f;
            s.erase(s.begin());
            if (actDist > dist[u]) continue;
            for (int i = 0; i < (int)G[u].size(); i++)
            {
                flowEdge &e = G[u][i];
                int v = e.to;
                if (e.cap <= e.flow) continue;
                int newDist = actDist + e.cost + pot[u] - pot[v];
                if (newDist < dist[v])
                {
                    dist[v] = newDist;
                    s.insert({newDist, v});
                    prevNode[v] = u;
                    prevEdge[v] = i;
                    curFlow[v] = min(curFlow[u], e.cap - e.flow);
                }
            }
        }
        if (dist[T] == INF)
            break;
        for (int i = 0; i < nodes; i++)
            pot[i] += dist[i];
        int df = curFlow[T];
        flow += df;
        for (int v = T; v != S; v = prevNode[v])
        {
            flowEdge &e = G[prevNode[v]][prevEdge[v]];
            e.flow += df;
            G[v][e.rev].flow -= df;
            flowCost += df * e.cost;
        }
    }
    return {flow, flowCost};
}
};
```

4.4 simplex

```
vector<int> X,Y;
vector<vector<double> > A;
vector<double> b,c;
double z;
int n,m;
void pivot(int x,int y){
```

```
  swap(X[y],Y[x]);
  b[x]/=A[x][y];
  fore(i,0,m)if(i!=y)A[x][i]/=A[x][y];
  A[x][y]=1/A[x][y];
  fore(i,0,n)if(i!=x&&abs(A[i][y])>EPS){
    b[i]-=A[i][y]*b[x];
    fore(j,0,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j];
    A[i][y]=-A[i][y]*A[x][y];
  }
  z+=c[y]*b[x];
  fore(i,0,m)if(i!=y)c[i]-=c[y]*A[x][i];
  c[y]=-c[y]*A[x][y];
}
pair<double,vector<double> > simplex( // maximize c^T x s.t. Ax<=b, x>=0
    vector<vector<double> > _A, vector<double> _b, vector<double> _c){
  // returns pair (maximum value, solution vector)
  A=_A;b=_b;c=_c;
  n=b.size();m=c.size();z=0.;
  X=vector<int>(m);Y=vector<int>(n);
  fore(i,0,m)X[i]=i;
  fore(i,0,n)Y[i]=i+m;
  while(1){
    int x=-1,y=-1;
    double mn=-EPS;
    fore(i,0,n)if(b[i]<mn)mn=b[i],x=i;
    if(x<0)break;
    fore(i,0,m)if(A[x][i]<-EPS){y=i;break;}
    assert(y>=0); // no solution to Ax<=b
    pivot(x,y);
  }
  while(1){
    double mx=EPS;
    int x=-1,y=-1;
    fore(i,0,m)if(c[i]>mx)mx=c[i],y=i;
    if(y<0)break;
    double mn=1e200;
    fore(i,0,n)if(A[i][y]>EPS&&b[i]/A[i][y]<mn)mn=b[i]/A[i][y],x=i;
    assert(x>=0); // c^T x is unbounded
    pivot(x,y);
  }
  vector<double> r(m);
  fore(i,0,n)if(Y[i]<m)r[Y[i]]=b[i];
  return mp(z,r);
}
```

5 geometry

5.1 ConvexHull(Monotone Chain)

```
// devuelve horario
vector<point> hull(vector<point> p)
{
        int n = p.size();
        vector<point> h;
        sort(all(p));
        fore(i, 0, n)
        {
                while(h.size() >= 2 && p[i].left(h[sz(h) - 2], h.back()))
                    h.pop_back();
                h.push_back(p[i]);
        }
        h.pop_back();
        int k = h.size();
        for(int i = n-1; i > -1; i--)
        {
                while(h.size() >= k + 2 && p[i].left(h[sz(h) - 2],
                    h.back())) h.pop_back();
                h.pb(p[i]);
        }
        h.pop_back();
        return h;
}
```

5.2 Lattice points

```
//calculates number of integer points (x ; y) such for 0  x < n and 0 < y
     k * x +  b
int count_lattices(double k, double b, long long n) {
    auto fk = floor(k);
    auto fb = floor(b);
    auto cnt = 0LL;
    if (k > 1 - EPS || b > 1 -EPS) {
        cnt += (fk * (n - 1) + 2 * fb) * n / 2;
        k -= fk;
        b -= fb;
    }
    auto t = k * n + b;
```

```
    auto ft = floor(t);
    if (ft > 1 - EPS) {
        cnt += count_lattices(1 / k, (t - floor(t)) / k, floor(t));
    }
    return cnt;
}
pair<double, double> kb(point a, point b)
{
	return mp((b.y - a.y) / (b.x - a.x), (a.y * b.x - a.x * b.y) /
	    (b.x - a.x));
}

// # of lattice points s.t. ax+by<=c, x,y>0 (a,b is positive integer)
ll f(ll a, ll b, ll c){
	if(c<=0) return 0;
	if(a<b) swap(a, b);
	ll m=c/a;
	if(a==b) return m*(m-1)/2;
	ll 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)
ll g(ll a, ll b, ll c, ll X, ll 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);
}
```

5.3 Minkowski sum of convex polygons

```
typedef vector<point> poly;
void norm(poly &pol) {
	int pos = 0;
	fore(i, 0, pol.size()) {
		if(pol[i] < pol[pos])
			pos = i;
	}
	rotate(pol.begin(), pol.begin() + pos, pol.end());
}
poly minkos(poly &a, poly &b) {
	norm(a);
	norm(b);
	int posa = 0, posb = 0, ta = a.size(), tb = b.size();
	poly res;
	ll cro;
	while(posa < ta || posb < tb) {
		res.pb(a[(posa) % ta] + b[(posb) % tb]);
		cro = (a[(posa + 1) % ta] - a[posa % ta]) ^ (b[(posb + 1)
		    % tb] - b[posb % tb]);
		if(cro == 0)
			posa++, posb++;
		else if(cro < 0)
			posb++;
		else posa++;
	}
	return res;
}
```

5.4 Point in Poly (convex)

```
// logaritmico counterclockwise
bool inpol(poly &pol, point p) {
	int n = pol.size();
	if(((pol[1] - pol[0]) ^ (p - pol[0])) < 0 || ((pol[n - 1] -
	    pol[0]) ^ (p - pol[0])) > 0)
		return 0;
	int lo = 1, hi = n - 2, mid, res;
	while(lo <= hi) {
		mid = (lo + hi) / 2;
		if(((pol[mid] - pol[0]) ^ (p - pol[0])) >= 0)
			res = mid, lo = mid + 1;
		else
			hi = mid - 1;
	}
	return ((pol[res + 1] - pol[res]) ^ (p - pol[res])) >= 0;
}
```

5.5 Points and Lines

```
struct point
{
	double x,y;
	point() {x=y=0;}
```

```
        point(double _x, double _y) : x(_x), y(_y) {}
        point operator+(point a) const {
                a.x+=x;
                a.y+=y;
                return a;
        }
};
double dist(point a, point b) {
        return hypot(a.x-b.x,a.y-b.y);
}
point tovec(point a, point b) {
        return point(b.x-a.x,b.y-a.y);
}
point translate (point p, point v) {
        return point(p.x+v.x,p.y+v.y);
}
point scale(point v, double sc) {
        return point(v.x*sc,v.y*sc);
}
point rotate(point v, double theta) {//rotacion antihorario ccw
        theta *= acos(-1)/180.0;
        return point(v.x*cos(theta)-v.y*sin(theta),v.x*sin(theta)+
            v.y*cos(theta));
}
point clos(point a, line l, line &pe) {//closest point in a line and
    perpendicular line from a
        if(fabs(l.a) < EPS) {
                pe.a = 1, pe.b = 0, pe.c = -a.x;
                return point(a.x,-l.c);
        }
        if(fabs(l.b) < EPS) {
                pe.a = 0, pe.b = 1, pe.c = -a.y;
                return point(-l.c,a.y);
        }
        tolinegr(a, 1/(l.a),pe);
        areIntersect(l,pe,a);
        return a;
}
point reflexion(point p, point a, point b) {// del punto p a linea ab
        line l,li;
        toline(a,b,li);
        point p1 = clos(p,li,l);
        p1 = p1+(tovec(p,p1));
        return p1;
}
```

```
double norm_sq(point a) {
        return a.x * a.x + a.y * a.y;
}
double dot(point a, point b) {
        return a.x*b.x + b.y*a.y;
}
double angle(point a, point b, point c) {//b el del medio
        a = tovec(b,a), b = tovec(b,c);
        double res = dot(a,b);
        res = acos(res / (sqrt(norm_sq(a))*sqrt(norm_sq(b))));
        res*= 180.0/acos(-1);
        return res;
}
double cross(point a, point b) {//prosucto cruz
        return a.x * b.y - a.y * b.x;
}
double distToLine(point p, point a, point b, point &c) {//con producto
    punto halla el punto
        //c = a + u*ab
        point ab = tovec(a,b), ap = tovec(a,p);
        double u = dot(ab,ap) / norm_sq(ab);
        c = translate(a, scale(ab,u));
        return dist(p,c);
}
double distToline1(point p, point a, point b) {//con producto cruz solo
    distancia
        point ap = tovec(a,p), ab = tovec(a,b);
        return fabs(cross(ab,ap)/hypot(ab.x,ab.y));
}
```

5.6 Polygons

```
struct point{
        double x,y;
        point(){x=y=0;}
        point(double X, double Y): x(X), y(Y) {}
        point operator+(point a) const
        {
                a.x+=x;
                a.y+=y;
                return a;
        }
        bool operator<(point a) const
```

```
	{
		return (a.x == x? a.y < y : a.x < x);
	}
};
double dist(point a, point b)
{
	return hypot(a.x-b.x, a.y-b.y);
}
point tovec(point a, point b)
{
	return point(b.x-a.x,b.y-a.y);
}
double norm(point a)
{
	return hypot(a.x,a.y);
}
double dot(point a, point b)
{
	return a.x*b.x + a.y*b.y;
}
double cross(point a, point b)
{
	return a.x*b.y - a.y*b.x;
}
bool ccw(point a, point b, point c)
{
	return cross(tovec(a,b),tovec(a,c)) >= 0; //depende si se acepta
	    colinear o no
}
double an(point a, point b, point c)
{
	a = tovec(b,a), b = tovec(b,c);
	return acos(dot(a,b)/(norm(a)*norm(b)));
}
double perimeter(const vector<point> &p)
{
	double result = 0.0;
	for(int i = 0; i<p.size()-1; i++)
	{
		result += dist(p[i],p[i+1]);
	}
	return result;
}
double area(const vector<point> &p)
{
	double result = 0.0;
	for(int i=0;i<p.size()-1;i++)
	{
		result += p[i].x*p[i+1].y - p[i].y*p[i+1].x;;
	}
	return fabs(result)/2.0;
}
point lineIntersectSeg(point p, point q, point A, point B)
{
	double a = B.y - A.y;
	double b = A.x - B.x;
	double c = B.x * A.y - A.x * B.y;
	double u = fabs(a * p.x + b * p.y + c);
	double v = fabs(a * q.x + b * q.y + c);
	return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) /
	    (u+v));
}
vector<point> cutPolygon(point a, point b, const vector<point> &Q)
{
	vector<point> P;
	for (int i = 0; i < (int)Q.size(); i++) {
		double left1 = cross(tovec(a, b), tovec(a, Q[i])), left2 =
		    0;
		if (i != (int)Q.size()-1) left2 = cross(tovec(a, b),
		    tovec(a, Q[i+1]));
		if (left1 > -EPS) P.push_back(Q[i]); // Q[i] is on the
		    left of ab ; left1 < EPS para la derecha
		if (left1 * left2 < -EPS) // edge (Q[i], Q[i+1]) crosses
		    line ab
			P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
	}
	if (!P.empty() && (P.back().x != P.front().x || P.back().y !=
	    P.front().y))
		P.push_back(P.front()); // make Ps first point = Ps last
		    point
	return P;
}
bool isConvex(const vector<point> &p)
{
	int sz = p.size();
	if(sz<=3) return false;
	bool left = ccw(p[0],p[1],p[2]);
	cout<<left<<endl;
	for(int i = 1; i < sz - 1; i++)
	{
```

```
                cout<<i<<' '<<ccw(p[i],p[i+1],p[((i+2)==sz)? 1:i+2])<<endl;
                if(ccw(p[i],p[i+1],p[((i+2)==sz)? 1:i+2])!=left)
                        return false;
        }
        return true;
}
bool isIn(const vector<point> &p, point a)
{
        double ang = 0;
        int sz = p.size();
        if(sz == 0) return false;
        for(int i = 0; i<sz-1;i++)
        {
                if(ccw(a,p[i],p[i+1]))
                        ang += an(p[i],a,p[i+1]);
                else
                        ang -= an(p[i],a,p[i+1]);
        }
        cout<<ang<<endl;
        return fabs(ang - 2.0*PI) < EPS;
}
```

5.7 Triangles and Circles

```
double rInCircle(double ab, double bc, double ca) {
        double s = (ab+bc+ca)/2;
        return sqrt(s*(s-ab)*(s-bc)*(s-ca));
}
double areaTri1(double a, double b, double c) {//heron
        double s = (a+b+c)/2;
        return sqrt(s*(s-a)*(s-b)*(s-c));
}
line perp(line a, point p) {//perpendicular
        line res;
        if(a.b=0)
                res.a = 0, res.b = 1, res.c = -p.y;
        else
                if(fabs(a.a)<EPS)
                        res.a = 1, res.b = 0, res.c = -p.y;
                else
                        res.a = -1.0/a.a, res.b = 1, res.c = -res.a*p.x-p.y;

}
```

```
bool circumCircle(point a, point b, point c, point &ctr, double &r)
    {//circuncentro completo
        double area = areaTri(a,b,c);
        if(fabs(area)<EPS) return 0;
        line l1, l2;
        pointsToLine(a,b,l2);
        pointsToLine(a,c,l2);
        point p1 = point((a.x+b.x)/2.0,(a.y+b.y)/2.0), p2 =
            point((a.x+c.x)/2.0,(a.y+c.y)/2.0);
        l1 = perp(l1,p1), l2 = perp(l2,p2);
        areIntersect(l1,l2,ctr);
        r = dist(a,b)*dist(b,c)*dist(a,c)/(4.0*areaTri(a,b,c));
        return true;
}
bool isInCircum(point a, point b, point c, point p) {//si esta dentro del
    circulo circunscrito
        double r;
        point ctr;
        if(!circumCircle(a,b,c,ctr,r)) return false;
        return dist(ctr,p) <= r ;
}
bool inCircle(point a, point b, point c, point &ctr) {//incentro
        double r = rIncircle(a,b,c);
        if(r< EPS) return false;
        line l1,l2;
        point p1;
        double ratio = dist(a,b) / dist(a,c);
        p1 = translate(b, scale(tovec(b,c),ratio/(1+ratio)));
        pointsToLine(a,p1,l1);
        ratio = dist(b,a) / dist(b,c);
        p1 = translate(a, scale(tovec(a,c),ratio/(1+ratio)));
        pointsToLine(b,p1,l2);
        areIntersect(l1,l2,ctr);
        return true;
}
line toLinep(point a, point b, point c) {//para mediatriz
        line l;
        if(b.x == c.x)
                l.a = 0, l.b = 1 , l.c = -a.y;
        else if(b.y == c.y)
                l.a = 1, l.b = 0, l.c = -a.x;
        else
                l.a = 1/((b.y-a.y)/(b.x-a.x)), l.b = 1, l.c = -l.a*a.x-a.y;
        return l;
}
```

```
point circun(point a, point b, point c) {//circuncentro
       line l1, l2;
       l1 = toLinep(point((a.x+b.x)/2,(a.y+b.y)/2),a,b);
       l2 = toLinep(point((a.x+c.x)/2,(a.y+c.y)/2),a,c);
       areIntersect(l1,l2,a);
       return a;
}
bool circle2PtsRad(point a, point b, double r, point &c) {//dados 2
    puntos y un radio
       double det = (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
       det = r * r / det - 0.25;
       if(det < 0.0) return false;
       det = sqrt(det);
       c.x = (a.x + b.x) * 0.5 + (b.y-a.y) * det;
       c.y = (a.y + b.y) * 0.5 + (a.x-b.x) * det;
       return true;
}
```

5.8 basics

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
```

```
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }
int sign(type x) { return ge(x, 0) - le(x, 0); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y;
        }
    bool operator !=(const point &p) const{ return x != p.x or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x
        and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
```

```
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y,
            min(p.y, q.y)) and le(y, max(p.y, q.y));
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this -
        x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) : min(dist(x),
            dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x +
        cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.y / p.abs(), p.x / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p) { return point(-p.y,p.x); }
point rotate_cw90(point p) { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions
    below:
type dot(point p, point q)  { return p.x*q.x + p.y*q.y; }
type cross(point p, point q) { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) +
    cross(c,a); }

int angle_less(const point& a1, const point& b1, const point& a2, const
    point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot( a1, b1), abs(cross( a1, b1)));
    point p2(dot( a2, b2), abs(cross( a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}
```

5.9 centroide

```
// calcula el centro de masa de un poligono antihorario
point cen(vector<point> p) {
        double x = 0, y = 0, area = 0, ax;
        int n = p.size()-1;
        fore(i, 0, n) {
                ax = (p[i] ^ p[i+1]) / 2;
                area += ax;
                x += ax * (p[i].x + p[i+1].x) / 3;
                y += ax * (p[i].y + p[i+1].y) / 3;
        }
        return point(x / area, y / area);
}
```

5.10 circle

```
#include "basics.cpp"
#include "lines.cpp"
```

```
struct circle {
    point c;
    ld r;
    circle() { c = point(); r = 0; }
    circle(point _c, ld _r) : c(_c), r(_r) {}
    ld area() { return acos(-1.0)*r*r; }
    ld chord(ld rad) { return 2*r*sin(rad/2.0); }
    ld sector(ld rad) { return 0.5*rad*area()/acos(-1.0); }
    bool intersects(circle other) {
        return le(c.dist(other.c), r + other.r);
    }
    bool contains(point p) { return le(c.dist(p), r); }
    pair<point, point> getTangentPoint(point p) {
        ld d1 = c.dist(p), theta = asin(r/d1);
        point p1 = (c - p).rotate(-theta);
        point p2 = (c - p).rotate(theta);
        p1 = p1*(sqrt(d1*d1 - r*r)/d1) + p;
        p2 = p2*(sqrt(d1*d1 - r*r)/d1) + p;
        return make_pair(p1,p2);
    }
};

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b - a).y, -(b - a).x);
    point v = point((c - a).y, -(c - a).x);
    point n = (c - b)*0.5;
    ld t = cross(u,n)/cross(v,u);
    ans.c = ((a + c)*0.5) + (v*t);
    ans.r = ans.c.dist(a);
    return ans;
}

point compute_circle_center(point a, point b, point c) {
    //circumcenter
    b = (a + b)/2;
    c = (a + c)/2;
    return compute_line_intersection(b, b + rotate_cw90(a - b), c, c +
        rotate_cw90(a - c));
}

int inside_circle(point p, circle c) {
    if (fabs(p.dist(c.c) - c.r)<EPS) return 1;
    else if (p.dist(c.c) < c.r) return 0;
    else return 2;
} //0 = inside/1 = border/2 = outside

circle incircle( point p1, point p2, point p3 ) {
    ld m1 = p2.dist(p3);
    ld m2 = p1.dist(p3);
    ld m3 = p1.dist(p2);
    point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1 + m2 + m3));
    ld s = 0.5*(m1 + m2 + m3);
    ld r = sqrt(s*(s - m1)*(s - m2)*(s - m3))/s;
    return circle(c, r);
}

circle minimum_circle(vector<point> p) {
    random_shuffle(p.begin(), p.end());
    circle C = circle(p[0], 0.0);
    for(int i = 0; i < (int)p.size(); i++) {
        if (C.contains(p[i])) continue;
        C = circle(p[i], 0.0);
        for(int j = 0; j < i; j++) {
            if (C.contains(p[j])) continue;
            C = circle((p[j] + p[i])*0.5, 0.5*p[j].dist(p[i]));
            for(int k = 0; k < j; k++) {
                if (C.contains(p[k])) continue;
                C = circumcircle(p[j], p[i], p[k]);
            }
        }
    }
    return C;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<point> circle_line_intersection(point a, point b, point c, ld r) {
    vector<point> ret;
    b = b - a;
    a = a - c;
    ld A = dot(b, b);
    ld B = dot(a, b);
    ld C = dot(a, a) - r*r;
    ld D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(c + a + b*(sqrt(D + EPS) - B)/A);
    if (D > EPS)
        ret.push_back(c + a + b*(-B - sqrt(D))/A);
    return ret;
```

```
}

vector<point> circle_circle_intersection(point a, point b, ld r, ld R) {
    vector<point> ret;
    ld d = sqrt(a.dist2(b));
    if (d > r + R || d + min(r, R) < max(r, R)) return ret;
    ld x = (d*d - R*R + r*r)/(2*d);
    ld y = sqrt(r*r - x*x);
    point v = (b - a)/d;
    ret.push_back(a + v*x + rotate_ccw90(v)*y);
    if (y > 0)
        ret.push_back(a + v*x - rotate_ccw90(v)*y);
    return ret;
}

//GREAT CIRCLE

double gcTheta(double pLat, double pLong, double qLat, double qLong) {
        pLat *= acos(-1.0) / 180.0; pLong *= acos(-1.0) / 180.0; //
            convert degree to radian
        qLat *= acos(-1.0) / 180.0; qLong *= acos(-1.0) / 180.0;
        return acos(cos(pLat)*cos(pLong)*cos(qLat)*cos(qLong) +
                cos(pLat)*sin(pLong)*cos(qLat)*sin(qLong) +
                sin(pLat)*sin(qLat));
}

double gcDistance(double pLat, double pLong, double qLat, double qLong,
    double radius) {
        return radius*gcTheta(pLat, pLong, qLat, qLong);
}


/*
 * Codeforces 101707B
 */
/*
point A, B;
circle C;

double getd2(point a, point b) {
        double h = dist(a, b);
        double r = C.r;
        double alpha = asin(h/(2*r));
        while (alpha < 0) alpha += 2*acos(-1.0);
        return dist(a, A) + dist(b, B) + r*2*min(alpha, 2*acos(-1.0) -
            alpha);
}

int main() {
        scanf("%lf %lf", &A.x, &A.y);
        scanf("%lf %lf", &B.x, &B.y);
        scanf("%lf %lf %lf", &C.c.x, &C.c.y, &C.r);
        double ans;
        if (distToLineSegment(C.c, A, B) >= C.r) {
                ans = dist(A, B);
        }
        else {
                pair<point, point> tan1 = C.getTangentPoint(A);
                pair<point, point> tan2 = C.getTangentPoint(B);
                ans = 1e+30;
                ans = min(ans, getd2(tan1.first, tan2.first));
                ans = min(ans, getd2(tan1.first, tan2.second));
                ans = min(ans, getd2(tan1.second, tan2.first));
                ans = min(ans, getd2(tan1.second, tan2.second));
        }
        printf("%.18f\n", ans);
        return 0;
}*/
```

5.11 circle2ptsrad

```
bool circle2PtsRad(point a, point b, double r, point &c) {//dados 2
    puntos y un radio
  double det = (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
  det = r * r / det - 0.25;
  if(det < 0.0) return false;
  det = sqrt(det);
  c.x = (a.x + b.x) * 0.5 + (b.y-a.y) * det;
  c.y = (a.y + b.y) * 0.5 + (a.x-b.x) * det;
  return true;
}
```

5.12 closest pair

```
#include "basics.cpp"
```

```
//DIVIDE AND CONQUER METHOD
//Warning: include variable id into the struct point

struct cmp_y {
    bool operator()(const point & a, const point & b) const {
        return a.y < b.y;
    }
};

ld min_dist = LINF;
pair<int, int> best_pair;
vector<point> pts, stripe;
int n;

void upd_ans(const point & a, const point & b) {
    ld dist = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
    if (dist < min_dist) {
        min_dist = dist;
        // best_pair = {a.id, b.id};
    }
}

void closest_pair(int l, int r) {
    if (r - l <= 3) {
        for (int i = l; i < r; ++i) {
            for (int j = i + 1; j < r; ++j) {
                upd_ans(pts[i], pts[j]);
            }
        }
        sort(pts.begin() + l, pts.begin() + r, cmp_y());
        return;
    }

    int m = (l + r) >> 1;
    type midx = pts[m].x;
    closest_pair(l, m);
    closest_pair(m, r);

    merge(pts.begin() + l, pts.begin() + m, pts.begin() + m, pts.begin()
        + r, stripe.begin(), cmp_y());
    copy(stripe.begin(), stripe.begin() + r - l, pts.begin() + l);

    int stripe_sz = 0;
    for (int i = l; i < r; ++i) {
        if (abs(pts[i].x - midx) < min_dist) {
            for (int j = stripe_sz - 1; j >= 0 && pts[i].y - stripe[j].y <
                min_dist; --j)
                upd_ans(pts[i], stripe[j]);
            stripe[stripe_sz++] = pts[i];
        }
    }
}

int main(){
    //read and save in vector pts
    min_dist = LINF;
    stripe.resize(n);
    sort(pts.begin(), pts.end());
    closest_pair(0, n);
}


//LINE SWEEP
int n; //amount of points
point pnt[N];

struct cmp_y {
    bool operator()(const point & a, const point & b) const {
        if(a.y == b.y) return a.x < b.x;
        return a.y < b.y;
    }
};

ld closest_pair() {
  sort(pnt, pnt+n);
  ld best = numeric_limits<double>::infinity();
  set<point, cmp_y> box;

  box.insert(pnt[0]);
  int l = 0;

  for (int i = 1; i < n; i++){
    while(l < i and pnt[i].x - pnt[l].x > best)
      box.erase(pnt[l++]);
    for(auto it = box.lower_bound({0, pnt[i].y - best}); it != box.end()
        and pnt[i].y + best >= it->y; it++)
      best = min(best, hypot(pnt[i].x - it->x, pnt[i].y - it->y));
    box.insert(pnt[i]);
  }
  return best;
```

```
}
```

5.13 halfplane

```
const double DINF=1e100;
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);
int sgn2(double x){return x<0?-1:1;}
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(auto a){return ln(p,p+pq.rot(a));} // 2D
};
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 (2D)
  return ln((p+q)*.5,p).rot(ccw90);
}
// polygon intersecting left side of halfplanes
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}};
  fore(i,0,4) b.pb(halfplane(bx[i],bx[(i+1)%4]));
  sort(all(b));
  int n=sz(b),q=1,h=0;
  vector<halfplane> c(sz(b)+10);
  fore(i,0,n){
    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;
  fore(i,q,h+1) s.pb(c[i]^c[i+1]);
  return s;
}
struct pol {
  int n;vector<pt> p;
  pol(){}
  pol(vector<pt> _p){p=_p;n=p.size();}
  double area(){
    double r=0.;
    fore(i,0,n)r+=p[i]%p[(i+1)%n];
    return abs(r)/2; // negative if CW, positive if CCW
  }
};
```

5.14 latlong

```
/*
Converts from rectangular coordinates to latitude/longitude and vice
versa. Uses degrees (not radians).
*/

#include <iostream>
#include <cmath>

using namespace std;

struct ll
{
  double r, lat, lon;
};

struct rect
{
  double x, y, z;
};

ll convert(rect& P)
{
  ll Q;
  Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
  Q.lat = 180/M_PI*asin(P.z/Q.r);
  Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

  return Q;
}

rect convert(ll& Q)
{
  rect P;
  P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.z = Q.r*sin(Q.lat*M_PI/180);

  return P;
}

int main()
{
  rect A;
  ll B;

  A.x = -1.0; A.y = 2.0; A.z = -3.0;

  B = convert(A);
  cout << B.r << " " << B.lat << " " << B.lon << endl;

  A = convert(B);
  cout << A.x << " " << A.y << " " << A.z << endl;
}
```

5.15 linea

```
struct line {
  double a, b, c;
  line(point p, point q) {
    a = p.y - q.y;
    b = q.x - p.x;
```

```
    c = -a * p.x - b * p.y;
  };
  void setOrigin(point p) { c += a * p.x + b * p.y; } //trasladar linea
      como si p fuera el origen
};
double det(double a, double b, double c, double d) {
       return a * d - b * c;
}
point intersec(line a, line b) { //primero estar seguro si no son
    paralelas
       double d = -det(a.a, a.b, b.a, b.b);
       return point(det(a.c, a.b, b.c, b.b) / d, det(a.a, a.c, b.a, b.c)
           / d);
}
```

5.16 nearest neighbour

```
// Closest Neighbor - O(n * log(n))
const ll N = 1e6+3, INF = 1e18;
ll n, cn[N], x[N], y[N]; // number of points, closes neighbor, x
    coordinates, y coordinates

ll sqr(ll i) { return i*i; }
ll dist(int i, int j) { return sqr(x[i]-x[j]) + sqr(y[i]-y[j]); }
ll dist(int i) { return i == cn[i] ? INF : dist(i, cn[i]); }

bool cpx(int i, int j) { return x[i] < x[j] or (x[i] == x[j] and y[i] <
    y[j]); }
bool cpy(int i, int j) { return y[i] < y[j] or (y[i] == y[j] and x[i] <
    x[j]); }

ll calc(int i, ll x0) {
  ll dlt = dist(i) - sqr(x[i]-x0);
  return dlt >= 0 ? ceil(sqrt(dlt)) : -1;
}

void updt(int i, int j, ll x0, ll &dlt) {
  if (dist(i) > dist(i, j)) cn[i] = j, dlt = calc(i, x0);
}

void cmp(vi &u, vi &v, ll x0) {
  for(int a=0, b=0; a<u.size(); ++a) {
    ll i = u[a], dlt = calc(i, x0);
    while(b < v.size() and y[i] > y[v[b]]) b++;
    for(int j = b-1; j >= 0  and y[i] - dlt <= y[v[j]]; j--) updt(i,
        v[j], x0, dlt);
    for(int j = b; j < v.size() and y[i] + dlt >= y[v[j]]; j++) updt(i,
        v[j], x0, dlt);
  }
}

void slv(vi &ix, vi &iy) {
  int n = ix.size();
  if (n == 1) { cn[ix[0]] = ix[0]; return; }

  int m = ix[n/2];

  vi ix1, ix2, iy1, iy2;
  for(int i=0; i<n; ++i) {
    if (cpx(ix[i], m)) ix1.push_back(ix[i]);
    else ix2.push_back(ix[i]);

    if (cpx(iy[i], m)) iy1.push_back(iy[i]);
    else iy2.push_back(iy[i]);
  }

  slv(ix1, iy1);
  slv(ix2, iy2);

  cmp(iy1, iy2, x[m]);
  cmp(iy2, iy1, x[m]);
}

void slv(int n) {
  vi ix, iy;
  ix.resize(n);
  iy.resize(n);
  for(int i=0; i<n; ++i) ix[i] = iy[i] = i;
  sort(ix.begin(), ix.end(), cpx);
  sort(iy.begin(), iy.end(), cpy);
  slv(ix, iy);
}
```

5.17 polygon

```
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.;
		fore(i,0,n)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;
		fore(i,0,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)
		fore(i,0,n)if(ln(p[i],p[(i+1)%n]).seghas(q))return true;
		int cnt=0;
		fore(i,0,n){
			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);
		fore(i,0,n)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;
		fore(i,1,n)if(v*(p[i]-p[k])>EPS)k=i;
		return p[k];
	}
	if(n==SZ(p))p.pb(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
	fore(i,0,n){
		int
		    d0=sgn(l.pq%(p[i]-l.p)),d1=sgn(l.pq%(p[(i+1)%n]-l.p));
		if(d0>=0)q.pb(p[i]);
		ln m(p[i],p[(i+1)%n]);
		if(d0*d1<0&&!(l/m))q.pb(l^m);
	}
	return pol(q);
}
double intercircle(circle c){ // area of intersection with circle
	double r=0.;
	fore(i,0,n){
		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()&&SZ(w.back().p)<2*SZ(p)){
                for(pt v:w.back().p)p.pb(v);
                w.pop_back();
        }
        w.pb(pol(chull(p)));
}
ll query(pt v){ // max(q*v:q in w), O(log^2(n))
        ll r=-INF;
        for(auto& p:w)r=max(r,p.farthest(v)*v);
        return r;
}
```

5.18 punto

```
/*typedef double T;
typedef complex<T> pt;
#define x real()
#define y imag()*/

//typedef long long ll;
//typedef long double ll;

struct point
{
        ll x, y;
        point() {}
        point(ll x, ll y): x(x), y(y) {}
        point operator -(point p) {return point(x - p.x, y - p.y);}
        point operator +(point p) {return point(x + p.x, y + p.y);}
        ll sq() {return x * x + y * y;}
        double abs() {return sqrt(sq());}
        ll operator ^(point p) {return x * p.y - y * p.x;}
        ll operator *(point p) {return x * p.x + y * p.y;}
        point operator *(ll a) {return point(x * a, y * a);}
        bool operator <(const point& p) const {return x == p.x ? y < p.y :
            x < p.x;}
        bool left(point a, point b) {return ((b - a) ^ (*this - a)) >= 0;}
        ostream& operator<<(ostream& os) {
                return os << "("<< x << "," << y << ")";
        }

};

void polarSort(vector<point>& v) {
  sort(v.begin(), v.end(), [] (point a, point b) {
    const point origin{0, 0};
    bool ba = a < origin, bb = b < origin;
    if (ba != bb) { return ba < bb; }
    return (a^b) > 0;
  });
}
```

5.19 radial sort

```
#include "basics.cpp"
point origin;

/*
    below < above
    order: [pi, 2 * pi)
*/

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(point p, point q){
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin,q) > 0;
    //Be Careful: p.dir(origin,q) == 0
}
```

5.20 stableSum

```
struct stableSum {
  int cnt = 0;
  vector<double> v, prefs{0};
  void operator += (double a) {
    assert(a >= 0);
    int s = ++cnt;
    while(s % 2 == 0) {
      a += v.back();
      v.pop_back(), pref.pop_back();
      s /= 2;
    }
    v.pb(a);
    pref.pb(pref.back() + a);
  }
  double val() {return pref.back();}
};
```

5.21 stanford delaunay

```
// Slow but simple Delaunay triangulation. Does not handle
// degenerate cases (from O'Rourke, Computational Geometry in C)
//
// Running time: O(n^4)
//
// INPUT:   x[] = x-coordinates
//          y[] = y-coordinates
//
// OUTPUT:  triples = a vector containing m triples of indices
//                    corresponding to triangle vertices

#include<vector>
using namespace std;

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {
        int n = x.size();
        vector<T> z(n);
        vector<triple> ret;

        for (int i = 0; i < n; i++)
            z[i] = x[i] * x[i] + y[i] * y[i];

        for (int i = 0; i < n-2; i++) {
            for (int j = i+1; j < n; j++) {
                for (int k = i+1; k < n; k++) {
                    if (j == k) continue;
                    double xn = (y[j]-y[i])*(z[k]-z[i]) -
                        (y[k]-y[i])*(z[j]-z[i]);
                    double yn = (x[k]-x[i])*(z[j]-z[i]) -
                        (x[j]-x[i])*(z[k]-z[i]);
                    double zn = (x[j]-x[i])*(y[k]-y[i]) -
                        (x[k]-x[i])*(y[j]-y[i]);
                    bool flag = zn < 0;
                    for (int m = 0; flag && m < n; m++)
                        flag = flag && ((x[m]-x[i])*xn +
                                        (y[m]-y[i])*yn +
                                        (z[m]-z[i])*zn <= 0);
                    if (flag) ret.push_back(triple(i, j, k));
                }
            }
        }
        return ret;
}

int main()
{
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

5.22 voronoy

```
/*
Complexity: O(nlogn)
Code by Monogon: https://codeforces.com/blog/entry/85638
This code doesn't work when two points have the same x coordinate.
This is handled simply by rotating all input points by 1 radian and
    praying to the geometry gods.

The definition of the Voronoi diagram immediately shows signs of
    applications.
*   Given a set S of n points and m query points p1,...,pm, we can answer
    for each query point, its nearest neighbor in S.
    This can be done in O((n+q)log(n+q)) offline by sweeping the Voronoi
        diagram and query points.
    Or it can be done online with persistent data structures.

*   For each Delaunay triangle, its circumcircle does not strictly
    contain any points in S. (In fact, you can also consider this the
    defining property of Delaunay triangulation)

*   The number of Delaunay edges is at most 3n - 6, so there is hope for
    an efficient construction.

*   Each point p belongs to S is adjacent to its nearest neighbor with a
    Delaunay edge.

*   The Delaunay triangulation maximizes the minimum angle in the
    triangles among all possible triangulations.

*   The Euclidean minimum spanning tree is a subset of Delaunay edges.

*/

#include <bits/stdc++.h>

#define ll long long
#define sz(x) ((int) (x).size())
#define all(x) (x).begin(), (x).end()
#define vi vector<int>
#define pii pair<int, int>
#define rep(i, a, b) for(int i = (a); i < (b); i++)
using namespace std;
template<typename T>
using minpq = priority_queue<T, vector<T>, greater<T>>;

using ftype = long double;
const ftype EPS = 1e-12, INF = 1e100;

struct pt {
    ftype x, y;
    pt(ftype x = 0, ftype y = 0) : x(x), y(y) {}

    // vector addition, subtraction, scalar multiplication
    pt operator+(const pt &o) const {
        return pt(x + o.x, y + o.y);
    }
    pt operator-(const pt &o) const {
        return pt(x - o.x, y - o.y);
    }
    pt operator*(const ftype &f) const {
        return pt(x * f, y * f);
    }

    // rotate 90 degrees counter-clockwise
    pt rot() const {
        return pt(-y, x);
    }

    // dot and cross products
    ftype dot(const pt &o) const {
        return x * o.x + y * o.y;
    }
    ftype cross(const pt &o) const {
        return x * o.y - y * o.x;
    }

    // length
    ftype len() const {
        return hypotl(x, y);
    }

    // compare points lexicographically
    bool operator<(const pt &o) const {
        return make_pair(x, y) < make_pair(o.x, o.y);
    }
};

// check if two vectors are collinear. It might make sense to use a
// different EPS here, especially if points have integer coordinates
```

```
bool collinear(pt a, pt b) {
    return abs(a.cross(b)) < EPS;
}


// intersection point of lines ab and cd. Precondition is that they
    aren't collinear
pt lineline(pt a, pt b, pt c, pt d) {
    return a + (b - a) * ((c - a).cross(d - c) / (b - a).cross(d - c));
}

// circumcircle of points a, b, c. Precondition is that abc is a
    non-degenerate triangle.
pt circumcenter(pt a, pt b, pt c) {
    b = (a + b) * 0.5;
    c = (a + c) * 0.5;
    return lineline(b, b + (b - a).rot(), c, c + (c - a).rot());
}

// x coordinate of sweep-line
ftype sweepx;

// an arc on the beacah line is given implicitly by the focus p,
// the focus q of the following arc, and the position of the sweep-line.
struct arc {
    mutable pt p, q;
    mutable int id = 0, i;
    arc(pt p, pt q, int i) : p(p), q(q), i(i) {}

    // get y coordinate of intersection with following arc.
    // don't question my magic formulas
    ftype gety(ftype x) const {
        if(q.y == INF) return INF;
        x += EPS;
        pt med = (p + q) * 0.5;
        pt dir = (p - med).rot();
        ftype D = (x - p.x) * (x - q.x);
        return med.y + ((med.x - x) * dir.x + sqrtl(D) * dir.len()) /
            dir.y;
    }
    bool operator<(const ftype &y) const {
        return gety(sweepx) < y;
    }
    bool operator<(const arc &o) const {
        return gety(sweepx) < o.gety(sweepx);
    }
};

// the beach line will be stored as a multiset of arc objects
using beach = multiset<arc, less<>>;

// an event is given by
//    x: the time of the event
//    id: If >= 0, it's a point event for index id.
//        If < 0, it's an ID for a vertex event
//    it: if a vertex event, the iterator for the arc to be deleted
struct event {
    ftype x;
    int id;
    beach::iterator it;
    event(ftype x, int id, beach::iterator it) : x(x), id(id), it(it) {}
    bool operator<(const event &e) const {
        return x > e.x;
    }
};

struct fortune {
    beach line; // self explanatory
    vector<pair<pt, int>> v; // (point, original index)
    priority_queue<event> Q; // priority queue of point and vertex events
    vector<pii> edges; // delaunay edges
    vector<bool> valid; // valid[-id] == true if the vertex event with
        corresponding id is valid
    int n, ti; // number of points, next available vertex ID
    fortune(vector<pt> p) {
        n = sz(p);
        v.resize(n);
        rep(i, 0, n) v[i] = {p[i], i};
        sort(all(v)); // sort points by coordinate, remember original
            indices for the delaunay edges
    }
    // update the remove event for the arc at position it
    void upd(beach::iterator it) {
        if(it->i == -1) return; // doesn't correspond to a real point
        valid[-it->id] = false; // mark existing remove event as invalid
        auto a = prev(it);
        if(collinear(it->q - it->p, a->p - it->p)) return; // doesn't
            generate a vertex event
        it->id = --ti; // new vertex event ID
        valid.push_back(true); // label this ID true
```

```
    pt c = circumcenter(it->p, it->q, a->p);
    ftype x = c.x + (c - it->p).len();
    // event is generated at time x.
    // make sure it passes the sweep-line, and that the arc truly
        shrinks to 0
    if(x > sweepx - EPS && a->gety(x) + EPS > it->gety(x)) {
        Q.push(event(x, it->id, it));
    }
}
// add Delaunay edge
void add_edge(int i, int j) {
    if(i == -1 || j == -1) return;
    edges.push_back({v[i].second, v[j].second});
}
// handle a point event
void add(int i) {
    pt p = v[i].first;
    // find arc to split
    auto c = line.lower_bound(p.y);
    // insert new arcs. passing the following iterator gives a slight
        speed-up
    auto b = line.insert(c, arc(p, c->p, i));
    auto a = line.insert(b, arc(c->p, p, c->i));
    add_edge(i, c->i);
    upd(a); upd(b); upd(c);
}
// handle a vertex event
void remove(beach::iterator it) {
    auto a = prev(it);
    auto b = next(it);
    line.erase(it);
    a->q = b->p;
    add_edge(a->i, b->i);
    upd(a); upd(b);
}
// X is a value exceeding all coordinates
void solve(ftype X = 1e9) {
    // insert two points that will always be in the beach line,
    // to avoid handling edge cases of an arc being first or last
    X *= 3;
    line.insert(arc(pt(-X, -X), pt(-X, X), -1));
    line.insert(arc(pt(-X, X), pt(INF, INF), -1));
    // create all point events
    rep(i, 0, n) {
        Q.push(event(v[i].first.x, i, line.end()));
    }
    ti = 0;
    valid.assign(1, false);
    while(!Q.empty()) {
        event e = Q.top(); Q.pop();
        sweepx = e.x;
        if(e.id >= 0) {
            add(e.id);
        }else if(valid[-e.id]) {
            remove(e.it);
        }
    }
}
};
```

6 graphs

6.1 2SAT

```
vector<int> g[tam],ginv[tam];

int n,m;
int neg(int num)
{
    if (num>n)
        return num-n;
    return num+n;
}

void add(int iz,int der)
{
    g[iz].pb(der);
    g[neg(der)].pb(neg(iz));

    ginv[der].pb(iz);
    ginv[neg(iz)].pb(neg(der));
}
bool vis[tam],valor[tam];vector<int> orden;
void dfs0(int num)
{
    vis[num]=1;
    for(int hijo:g[num])
```

```
    {
        if (!vis[hijo])
            dfs0(hijo);
    }
    orden.pb(num);
}
int componente[tam],scc;
void dfs1(int num)
{
    vis[num]=1;
    componente[num]=scc;
    for(int hijo: ginv[num])
    {
        if (!vis[hijo])
        dfs1(hijo);
    }
}
int main()
{
    ios::sync_with_stdio(0); cin.tie(0);

    cin>>n>>m;
    int iz,der,peso;
    forr(i,0,m)
    {
        cin>>iz>>der>>peso;
        if (peso==1)
        {
            add(iz,der);
            add(der,iz);
        }
        else
        {
            add(neg(iz),der);
            add(iz,neg(der));
        }
    }
    memset(vis,0,sizeof vis);
    forr(i,1,2*n+1)
    {
        if (!vis[i])
            dfs0(i);
    }
    memset(vis,0,sizeof vis);
    scc=0;
    forr(i,0,orden.size())
    {
        iz=orden[orden.size()-1-i];
        if (!vis[iz]){
            scc++;
            dfs1(iz);
        }
    }
    bool invalid=false;
    vector<int> respuesta;
    for(int i=1;i<=n;i++)
    {
        //cout<<componente[i]<<' '<<componente[i+n]<<endl;
        if (componente[i]==componente[i+n])
            invalid=1;
        if (componente[i]>componente[i+n])
        {
            respuesta.pb(i);
            valor[i]=1;
        }
        else valor[i]=0;
    }
    if (invalid==false)
    {
        cout<<respuesta.size()<<endl;
        for(int num:respuesta)
            cout<<num<<" ";
    }
    else
    {
        cout<<"Impossible\n";
    }
}
// read the question correctly (is y a vowel? what are the exact
    constraints?)
// look out for SPECIAL CASES (n=1?) and overflow (ll vs int?) ARRAY
    OUT OF BOUNDSS2
```

6.2 2sat

```
namespace sat2{
  set<int> G[tam], Ginv[tam];
```

```
  int N, mark[tam], mark_comp[tam], valor[tam];
  int neg(const int& x) { return (x>=N)? x - N : x + N;}
  void add_(const int& x,const int& y) {G[x].insert(y);Ginv[y].insert(x);}
  void addor(const int x,const int y) {add_(neg(x),y);add_(neg(y),x);}
  void dfs0(int u, vector<int>& orden) { mark[u] = 1;
    for(auto& v: G[u]) {
      if (!mark[v]) dfs0(v,orden);
    }  orden.push_back(u);
  }
  void dfs1(int u, const int& cmp) { mark_comp[u] = cmp;
    for(auto& v: Ginv[u]) {
      if (!mark_comp[v]) dfs1(v,cmp);
    }
  }
  bool check() { bool impos = false;
    for(int i = 0; i < N; i++) {
      impos |= (mark_comp[i] == mark_comp[neg(i)]);
       valor[i] = (mark_comp[i] > mark_comp[neg(i)]) ;}
    return !impos;
  }
}
```

6.3 LCA DP

```
#include <bits/stdc++.h>
#define clr(a,h)        memset(a, (h), sizeof(a))

using namespace std;

const int tam = 1010; // Max Nodos
const int Log2Tam = (log(tam)/log(2)) + 3;

vector< vi > g;
int dp[tam][Log2Tam], depth[tam];
int n, m, root; // nodos, aristas, raiz del arbol

void initDFS(int v, int p, int d)
{
        dp[v][0] = p; // padre inmediato
        depth[v] = d;
        for (int u : g[v])
        {
                if (u == p) continue;
                initDFS(u, v, d+1);
        }
}

void initLCA()
{
        clr(dp, -1);
        initDFS(root, -1, 0);
        for (int pot = 1; pot < Log2Tam; pot++)
        {
                for (int v = 0; v < n; v++)
                {
                        if (dp[v][pot-1] == -1) continue;
                        dp[v][pot] = dp[ dp[v][pot-1] ][pot-1];
                }
        }
}

int LCA(int a, int b)
{
        // b siempre debajo o al mismo nivel que a
        if (depth[a] > depth[b]) swap(a, b);

        int diff = depth[b] - depth[a];
        for (int pot = Log2Tam-1; pot >= 0; pot--)
        {
                if ( ( diff & (1 << pot) ) )
                {
                        b = dp[b][pot];
                }
        }
        if (a == b) return a;
        for (int pot = Log2Tam-1; pot >= 0; pot--)
        {
                if (dp[a][pot] != dp[b][pot])
                {
                        a = dp[a][pot];
                        b = dp[b][pot];
                }
        }
        int lca = dp[a][0];
        return lca;
}

int main()
```

```
{
    /*
        Iniciar el arbol en el grafo g
        Asignar la raiz en la variable root

        Llamar initLCA() luego de crear el arbol
        LCA(a, b) devuelve el LCA de los nodos a y b
    */
    return 0;
}
// PLUS ULTRA!
```

6.4 Tarjan y BlockCutTree

```
int n;
vector<vector<int>> adj;

vector<bool> visited;
vector<int> tin, low;
int timer;
vector<vectotr<int>> comps; // componentes biconexos
stack<int> stk;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    stk.push(v);
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v])
            {
                if (p != -1) IS_CUTPOINT(v);

                comps.push_back({v});
                while (comps.back().back() != to)
                {
                        comps.back().push_back(stk.top());
                        stk.pop();
                }

            }
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}

vector<int> id;
int curBCTNode;
vector<vector<int> > tree;
vector<bool> isAP;

void buildTree() {
    curBCTNode = 0;
    id.assign(n, -1);
    tree.clear();
    isAP.clear();
    fore(v, 0, n) {
        if (cutpoint[v]) {
            id[v] = tree.size();
            tree.pb({});
            isAP.pb(true);
        }
    }
    for (auto comp : comps) {
        int v = tree.size();
        tree.pb({});
        isAP.pb(false);
```

```
        for (int x : comp) {
            if (cutpoint[x]) {
                tree[v].pb(id[x]);
                tree[id[x]].pb(v);
            }
            else {
                id[x] = v;
            }
        }
    }
}
```

6.5 articulation-bridges-biconnected

```
namespace art_bic {
  vector<int> g[tam];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[tam],B[tam],T;
  int nbc; // number of biconnected components
  int art[tam]; // 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]--;
  }
}
```

6.6 bellman ford

```
/*****************************************************************************
* BELLMAN-FORD ALGORITHM (SHORTEST PATH TO A VERTEX - WITH NEGATIVE COST)
    *
* Time complexity: O(VE)
    *
* Usage: dist[node]
    *
* Notation: m:         number of edges
    *
*          n:         number of vertices
    *
*          (a, b, w):  edge between a and b with weight w
    *
*          s:         starting node
    *
*****************************************************************************
const int N = 1e4+10; // Maximum number of nodes
vector<int> adj[N], adjw[N];
int dist[N], v, w;

memset(dist, 63, sizeof(dist));
dist[0] = 0;
for (int i = 0; i < n-1; ++i)
  for (int u = 0; u < n; ++u)
    for (int j = 0; j < adj[u].size(); ++j)
      v = adj[u][j], w = adjw[u][j],
      dist[v] = min(dist[v], dist[u]+w);
```

6.7 centroid descomposition

```
vi g[tam];
vector<vi> par(tam), dispa(tam);
vi siz(tam), cintra(tam);
void sisi(int node, int pa, int dis, int papazo)
{
    if(papazo != -1)
        par[node].pb(papazo), dispa[node].pb(dis);
    siz[node] = 1;
    for(int x : g[node])
        if(x != pa && !cintra[x])
            sisi(x, node, dis + 1, papazo), siz[node] += siz[x];
}
int centruida(int node, int pa, int simp)
{
    for(int x : g[node])
        if(x != pa && !cintra[x])
            if(siz[x] > simp / 2)
                return centruida(x, node, simp);
    return node;
}
void centauros(int node, int pa)
{
    sisi(node, -1, 1, pa);
    int centris = centruida(node, -1, siz[node]);
    cintra[centris] = 1;
    for(int x : g[centris])
        if(!cintra[x])
            centauros(x, centris);
}
```

6.8 centroid

```
namespace cent_{
  vector<int> g[tam];int n;
  bool tk[tam];
  int fat[tam]; // father in centroid decomposition
  int szt[tam]; // 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; // next is ops
    for(auto y:g[x])if(!tk[y])cdfs(y,x);
  }
  void centroid(){memset(tk,false,sizeof(tk));cdfs();}
}
```

6.9 dijkstra

```
vi dijkstra(int n, vector<vii> &g, int s)
{
    vi dis(n, MOD), vis(n);
    dis[s] = 0;
    priority_queue<ii> que;
    que.push({0, s});
    while(!que.empty())
    {
        int node = que.top().s;
        que.pop();
        if(vis[node]) continue;
        vis[node] = 1;
        for(ii go : g[node])
            if(dis[go.f] > dis[node] + go.s)
            {
                dis[go.f] = dis[node] + go.s;
                que.push({-dis[go.f], go.f});
            }
    }

    return dis;
}
```

6.10 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&&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;
}
```

6.11 dsu

```
#define tam 1<<21
#define offset 200056

const ll MOD=1e9+7;
vector<int> g[tam];
int
    viz[tam],vder[tam],vtree[tam],depth[tam],csize[tam],cant[tam],vpa[tam];
int cuniv;
void dfs0(int num)
{
    vtree[cuniv]=depth[num]; //.-.
    viz[num]=cuniv++;
    csize[num]=1;
    for(int v:g[num])
    {
        if (depth[v]==-1){
            vpa[v]=num;
            depth[v]=depth[num]+1,dfs0(v),csize[num]+=csize[v];
        }
    }
    vder[num]=cuniv;
}
int amax,apos;
int answer[tam];
void dsu(int num,bool keep)
{
    int maxx,bigchild;
    maxx=bigchild=-1;
    for(int v:g[num])
    {
        if (v==vpa[num])continue;
        if (csize[v]>maxx)
        {
            maxx=csize[v];
            bigchild=v;
        }
    }

    for(int v:g[num])
    {
        if (v==vpa[num]|| v==bigchild)continue;
        dsu(v,0);
    }
    if (bigchild!=-1)
        dsu(bigchild,1);
```

```
    int posini=viz[num];
    cant[vtree[posini]]++;

    if (cant[vtree[posini]]>=amax)
        amax=cant[vtree[posini]],apos=vtree[posini];
    for(int v:g[num])
    {
        if (v==vpa[num]|| v==bigchild)continue;
        forr(i,viz[v],vder[v])
        {
            cant[vtree[i]]++;
            if (cant[vtree[i]]>amax)
                amax=cant[vtree[i]],apos=vtree[i];
            else if (cant[vtree[i]]==amax)
                apos=min(vtree[i],apos);
        }
    }
    answer[num]=apos;
    if (keep==0)
    {
        forr(i,viz[num],vder[num])
        {
            cant[vtree[i]]=0;
        }
        apos=amax=0;
    }
}
```

6.12 dynamic-connectivity

```
namespace dyn_con {
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
        printf("%d\n",dsu.comp); // can ask current state UnionFind
      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);
  }  };
}
```

6.13 edmonds-blossom

```
namespace ed_bls{ // undirected G
  vector<int> g[tam];
  int n,m,mt[tam],qh,qt,q[tam],ft[tam],bs[tam];
  bool inq[tam],inb[tam],inp[tam];
  int lca(int root, int x, int y){
    memset(inp,0,sizeof(inp));
    while(1){
      inp[x=bs[x]]=true;
      if(x==root)break;
      x=ft[mt[x]];
    }
    while(1){
      if(inp[y=bs[y]])return y;
      else y=ft[mt[y]];
    }
  }
  void mark(int z, int x){
    while(bs[x]!=z){
      int y=mt[x];
      inb[bs[x]]=inb[bs[y]]=true;
      x=ft[y];
      if(bs[x]!=z)ft[x]=y;
    }
  }
  void contr(int s, int x, int y){
    int z=lca(s,x,y);
    memset(inb,0,sizeof(inb));
    mark(z,x);mark(z,y);
    if(bs[x]!=z)ft[x]=y;
    if(bs[y]!=z)ft[y]=x;
    fore(x,0,n)if(inb[bs[x]]){
      bs[x]=z;
      if(!inq[x])inq[q[++qt]=x]=true;
    }
  }
  int findp(int s){
    memset(inq,0,sizeof(inq));
    memset(ft,-1,sizeof(ft));
    fore(i,0,n)bs[i]=i;
    inq[q[qh=qt=0]=s]=true;
    while(qh<=qt){
      int x=q[qh++];
      for(int y:g[x])if(bs[x]!=bs[y]&&mt[x]!=y){
        if(y==s||mt[y]>=0&&ft[mt[y]]>=0)contr(s,x,y);
        else if(ft[y]<0){
          ft[y]=x;
          if(mt[y]<0)return y;
          else if(!inq[mt[y]])inq[q[++qt]=mt[y]]=true;
        }
      }
    }
    return -1;
  }
  int aug(int s, int t){
    int x=t,y,z;
    while(x>=0){
      y=ft[x];
      z=mt[y];
      mt[y]=x;mt[x]=y;
      x=z;
    }
    return t>=0;
  }
  int edmonds(){ // O(n^2 m)
    int r=0;
    memset(mt,-1,sizeof(mt));
    fore(x,0,n)if(mt[x]<0)r+=aug(x,findp(x));
    return r;
  }
}
```

6.14 erdos gallai

```
// Erdos-Gallai - O(nlogn)
// check if it's possible to create a simple graph (undirected edges) from
// a sequence of vertice's degrees
bool gallai(vector<int> v) {
  vector<ll> sum;
  sum.resize(v.size());

  sort(v.begin(), v.end(), greater<int>());
  sum[0] = v[0];
  for (int i = 1; i < v.size(); i++) sum[i] = sum[i-1] + v[i];
  if (sum.back() % 2) return 0;

  for (int k = 1; k < v.size(); k++) {
```

```
    int p = lower_bound(v.begin(), v.end(), k, greater<int>()) -
        v.begin();
    if (p < k) p = k;
    if (sum[k-1] > 1ll*k*(p-1) + sum.back() - sum[p-1]) return 0;
  }
  return 1;
}
```

6.15 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;
}
```

6.16 lca

```
// Lowest Common Ancestor <O(nlogn), O(logn)>
const int N = 1e6, M = 25;
int anc[M][N], h[N], rt;

// TODO: Calculate h[u] and set anc[0][u] = parent of node u for each u

// build (sparse table)
anc[0][rt] = rt; // set parent of the root to itself
for (int i = 1; i < M; ++i)
  for (int j = 1; j <= n; ++j)
    anc[i][j] = anc[i-1][anc[i-1][j]];

// query
int lca(int u, int v) {
  if (h[u] < h[v]) swap(u, v);
  for (int i = M-1; i >= 0; --i) if (h[u]-(1<<i) >= h[v])
    u = anc[i][u];

  if (u == v) return u;

  for (int i = M-1; i >= 0; --i) if (anc[i][u] != anc[i][v])
    u = anc[i][u], v = anc[i][v];
  return anc[0][u];
}
```

6.17 max weight lca

```
// Using LCA to find max edge weight between (u, v)

const int N = 1e5+5; // Max number of vertices
const int K = 20;   // Each 1e3 requires ~ 10 K
const int M = K+5;
int n;              // Number of vertices
vector <pii> adj[N];
int vis[N], h[N], anc[N][M], mx[N][M];

void dfs (int u) {
    vis[u] = 1;
    for (auto p : adj[u]) {
        int v = p.st;
        int w = p.nd;
        if (!vis[v]) {
            h[v] = h[u]+1;
            anc[v][0] = u;
            mx[v][0] = w;
            dfs(v);
        }
```

```
    }
}

void build () {
    // cl(mn, 63) -- Don't forget to initialize with INF if min edge!
    anc[1][0] = 1;
    dfs(1);
    for (int j = 1; j <= K; j++) for (int i = 1; i <= n; i++) {
        anc[i][j] = anc[anc[i][j-1]][j-1];
        mx[i][j] = max(mx[i][j-1], mx[anc[i][j-1]][j-1]);
    }
}

int mxedge (int u, int v) {
    int ans = 0;

    if (h[u] < h[v]) swap(u, v);
    for (int j = K; j >= 0; j--) if (h[anc[u][j]] >= h[v]) {
        ans = max(ans, mx[u][j]);
        u = anc[u][j];
    }
    if (u == v) return ans;
    for (int j = K; j >= 0; j--) if (anc[u][j] != anc[v][j]) {
        ans = max(ans, mx[u][j]);
        ans = max(ans, mx[v][j]);
        u = anc[u][j];
        v = anc[v][j];
    }
    return max({ans, mx[u][0], mx[v][0]});
}
```

6.18 stoer$_w$agner

```
    // a is a N*N matrix storing the graph we use; a[i][j]=a[j][i]
    memset(use,0,sizeof(use));
    ans=maxlongint;
    for (int i=1;i<N;i++)
    {
        memcpy(visit,use,505*sizeof(int));
        memset(reach,0,sizeof(reach));
        memset(last,0,sizeof(last));
        t=0;
        for (int j=1;j<=N;j++)
            if (use[j]==0) {t=j;break;}
        for (int j=1;j<=N;j++)
            if (use[j]==0) reach[j]=a[t][j],last[j]=t;
        visit[t]=1;
        for (int j=1;j<=N-i;j++)
        {
            maxc=maxk=0;
            for (int k=1;k<=N;k++)
                if ((visit[k]==0)&&(reach[k]>maxc)) maxc=reach[k],maxk=k;
            c2=maxk,visit[maxk]=1;
            for (int k=1;k<=N;k++)
                if (visit[k]==0) reach[k]+=a[maxk][k],last[k]=maxk;
        }
        c1=last[c2];
        sum=0;
        for (int j=1;j<=N;j++)
            if (use[j]==0) sum+=a[j][c2];
        ans=min(ans,sum);
        use[c2]=1;
        for (int j=1;j<=N;j++)
            if ((c1!=j)&&(use[j]==0))
                {a[j][c1]+=a[j][c2];a[c1][j]=a[j][c1];}
    }
```

7 math

7.1 moebius O(n)

```
// multiplicative function calculator
// euler_phi and mobius are multiplicative
// if another f[N] needed just remove comments
// O(N)

bool p[N];
vector<ll> primes;
ll g[N];
// ll f[N];

void mfc(){
    // if g(1) != 1 than it's not multiplicative
    g[1] = 1;
    // f[1] = 1;
```

```
    primes.clear();
    primes.reserve(N / 10);
    for(ll i = 2; i < N; i++){
        if(!p[i]){
            primes.push_back(i);
            for(ll j = i; j < N; j *= i){
                g[j] = // g(p^k) you found
                // f[j] = f(p^k) you found
                p[j] = (j != i);
            }
        }
        for(ll j : primes){
            if(i * j >= N || i % j == 0)
                break;
            for(ll k = j; i * k < N; k *= j){
                g[i * k] = g[i] * g[k];
                // f[i * k] = f[i] * f[k];
                p[i * k] = true;
            }
        }
    }
}
```

7.2 Catalan

```
// Catalan, parentesis balanceados, arboles binarios, triangulacion
    poligono convexto de n + 2 lados, caminos en grilla sin atravesar
    diagonal
// Cat[n] = C(2n, n) / (n + 1)
// C(n, k) es el coeficiente binomial
Cat[0] = 1;
Cat[n+1] = Cat[n] * 2 * (2 * n + 1) / (n + 2);
Cat[n] = Cat[n-1] * 2 * (2 * n - 1) / (n + 1);

// Convolucion Catalan, parntesis balanceados en k cajas
// Catk[n] = C(2n + k - 1, n) * k / (n + k)
Catk[0] = 1;
Catk[n + 1] = Catk[n] * (2 * n + k) * (2 * n + 1 + k) / (n + 1) / (n + k
    + 1);
Catk[n] = Catk[n - 1] * (2 * n + k - 1) * (2 * n + k - 2) / n / (n + k);

int main()
{
        // count the number of plane trees with a given set of leaves,
        // the number of ways of inserting parentheses into a sequence,
        // and the number of ways of dissecting a convex polygon into
            smaller polygons by inserting diagonals
        ll scat[26];//super catalan
        scat[0]=scat[1]=1;
        for(int i=2;i<26;i++)
                scat[i]=(3*(2*i-1)*scat[i-1]-(i-2)*scat[i-2])/(i+1);
        ll cat[26];
        for(int i=0;i<26;i++)
                cat[i]=cata(i);
        return 0;
}
```

7.3 LinearRecurrence

```
/*
Description: Generates the kth term of an n-order linear recurrence
S[i] =    S[i    j    1]tr[j], given S[0 . . .  n    1] and tr[0 . . . n
      1]. Faster
than matrix multiplication. Useful together with BerlekampMassey.
Usage: linearRec({0, 1}, {1, 1}, k) // kth Fibonacci number
Time: O (n^2 log k)*/
typedef vector<ll> Poly;
#define sz(x) (int)(x).size()
  ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);
  auto combine = [&](Poly a, Poly b) {
  Poly res(n * 2 + 1);
  fore(i,0,n+1) fore(j,0,n+1)
    res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
  for (int i = 2 * n; i > n; --i) fore(j,0,n)
    res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
  res.resize(n + 1);
  return res;
  };
  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }
  ll res = 0;
```

```
  fore(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

7.4 basis Z2

```
const int N=905;
struct Gaus
{
    bitset<N> basis[N];
    bool val[N];
    int cmp;
    bool pos;
    Gaus(){
        cmp=0;
        pos=1;
    }
    void ad(bitset<N> nuevo,bool ult)
    {
        for(int i=N-1;i>=0;i--)
        {
            if (nuevo[i])
            {
                if (basis[i][i])
                    nuevo^=basis[i],ult^=val[i];
                else
                {
                    basis[i]=nuevo,val[i]=ult;
                    cmp++;
                    return;
                }
            }
        }
        if (ult)
            pos=0;
    }

};
```

7.5 binofac

```
int pot (int b, int e) {
    int res = 1;
    while(e)
    {
        if(e & 1) res = res * b % MOD;
        b = b * b % MOD;
        e /= 2;
    }
    return res;
}
vi fac(tam), facin(tam);
int bino(int n, int k) {
    return k < 0 ? 0 : n < 0 ? 0 : k > n ? 0 : fac[n] * facin[k] % MOD *
        facin[n - k] % MOD;
};
void precalbino() {
    fac[0] = facin[0] = 1;
    fore(i, 1, tam)
    {
        fac[i] = fac[i - 1] * i % MOD;
        facin[i] = pot(fac[i], MOD - 2);
    }
}
```

7.6 chinese-remainder

```
constexpr long long safe_mod(long long x, long long m) {
  x %= m;
  if (x < 0) x += m;
  return x;
}
constexpr std::pair<long long, long long> inv_gcd(long long a, long long
    b) {
  a = safe_mod(a, b);
  if (a == 0) return {b, 0};
  long long s = b, t = a;
  long long m0 = 0, m1 = 1;
  while (t) {
    long long u = s / t;
    s -= t * u;
    m0 -= m1 * u;
    auto tmp = s;
    s = t;
```

```
    t = tmp;
    tmp = m0;
    m0 = m1;
    m1 = tmp;
  }
  if (m0 < 0) m0 += b / s;
  return {s, m0};
}
std::pair<long long, long long> crt(const std::vector<long long>& r,
                const std::vector<long long>& m) {
  assert(r.size() == m.size());
  int n = int(r.size());
  long long r0 = 0, m0 = 1;
  for (int i = 0; i < n; i++) {
    assert(1 <= m[i]);
    long long r1 = safe_mod(r[i], m[i]), m1 = m[i];
    if (m0 < m1) {
      std::swap(r0, r1);
      std::swap(m0, m1);
    }
    if (m0 % m1 == 0) {
      if (r0 % m1 != r1) return {0, 0};
      continue;
    }
    long long g, im;
    std::tie(g, im) = inv_gcd(m0, m1);
    long long u1 = (m1 / g);
    if ((r1 - r0) % g) return {0, 0};
    long long x = (r1 - r0) / g % u1 * im % u1;
    r0 += x * m0;
    m0 *= u1;
    if (r0 < 0) r0 += m0;
  }
  return {r0, m0};
}
cin>>a>>b>>c>>d;
extendedEuclid(b, d);
mul = b / g * d;
b /= g;
d /= g;
cout<<(mulmod(x, mulmod(b, c, mul), mul) + mulmod(y, mulmod(d, a, mul),
    mul)) % mul<<' '<<mul<<'\n';
```

7.7 count primes 23

```
using ll = long long;
struct _count_primes_struct_t_ {
    vector<int> primes;
    vector<int> mnprimes;
    ll ans;
    ll y;
    vector<pair<pair<ll, int>, char>> queries;

    ll count_primes(ll n) {
        // this y is actually n/y
        // also no logarithms, welcome to reality, this y is the best for
            n=10^12 or n=10^13
        y = pow(n, 0.64);
        if (n < 100) y = n;

        // linear sieve
        primes.clear();
        mnprimes.assign(y + 1, -1);
        ans = 0;
        for (int i = 2; i <= y; ++i) {
            if (mnprimes[i] == -1) {
                mnprimes[i] = primes.size();
                primes.push_back(i);
            }
            for (int k = 0; k < primes.size(); ++k) {
                int j = primes[k];
                if (i * j > y) break;
                mnprimes[i * j] = k;
                if (i % j == 0) break;
            }
        }
        if (n < 100) return primes.size();
        ll s = n / y;

        for (int p : primes) {
            if (p > s) break;
            ans++;
        }
        // pi(n / y)
        int ssz = ans;

        // F with two pointers
        int ptr = primes.size() - 1;
```

```
    for (int i = ssz; i < primes.size(); ++i) {
        while (ptr >= i && (ll)primes[i] * primes[ptr] > n)
            --ptr;
        if (ptr < i) break;
        ans -= ptr - i + 1;
    }

    // phi, store all queries
    phi(n, ssz - 1);

    sort(queries.begin(), queries.end());
    int ind = 2;
    int sz = primes.size();

    // the order in fenwick will be reversed, because prefix sum in a
        fenwick is just one query
    fenwick fw(sz);
    for (auto [na, sign] : queries) {
        auto [n, a] = na;
        while (ind <= n)
            fw.add(sz - 1 - mnprimes[ind++], 1);
        ans += (fw.ask(sz - a - 2) + 1) * sign;
    }
    queries.clear();
    return ans - 1;
}

void phi(ll n, int a, int sign = 1) {
    if (n == 0) return;
    if (a == -1) {
        ans += n * sign;
        return;
    }
    if (n <= y) {
        queries.emplace_back(make_pair(n, a), sign);
        return;
    }
    phi(n, a - 1, sign);
    phi(n / primes[a], a - 1, -sign);
}

struct fenwick {
    vector<int> tree;
    int n;

    fenwick(int n = 0) : n(n) {
        tree.assign(n, 0);
    }

    void add(int i, int k) {
        for (; i < n; i = (i | (i + 1)))
            tree[i] += k;
    }

    int ask(int r) {
        int res = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            res += tree[r];
        return res;
    }
};
} _count_primes_struct_;

ll count_primes(ll n) {
    return _count_primes_struct_.count_primes(n);
}
```

7.8 count primes 34

```
using ll = long long;
ll count_primes(ll n) {
    vector<ll> v;
    for (ll k = 1; k * k <= n; ++k) {
        v.push_back(n / k);
        v.push_back(k);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());

    // return i such that v[i] = x
    // since v[i] = i + 1 for i <= sqrt(n) and v[v.size() - i] = n / i
        for i <= sqrt(n),
    // we can calculate index in O(1)
    ll sq = sqrt(n);
    auto geti = [&](ll x) {
        if (x <= sq) return (int)x - 1;
        else         return (int)(v.size() - (n / x));
    };
```

```
    vector<ll> dp(v.size());

    // S(n, 0) = n
    for (int i = 0; i < v.size(); ++i)
        dp[i] = v[i];

    int a = 0;
    for (ll p = 2; p * p <= n; ++p) {
        // this condition is true for primes
        if (dp[geti(p)] != dp[geti(p - 1)]) {
            ++a;
            for (int i = (int)v.size() - 1; i >= 0; --i) {
                if (v[i] < p * p) break;
                dp[i] -= dp[geti(v[i] / p)] - a;
            }
        }
    }

    return dp[geti(n)] - 1;
}
```

7.9 exteded-euclid

```
int x, y, d;
void extendedEuclid(int a, int b)//ecuacion diofantica ax + by = d
{
  if(b==0) {x=1; y=0; d=a; return;}
  extendedEuclid(b,a%b);
  int x1=y;
  y = x-(a/b)*y;
  x=x1;
}
```

7.10 extended euclid

```
// Extended Euclid:
void euclid(ll a, ll b, ll &x, ll &y) {
  if (b) euclid(b, a%b, y, x), y -= x*(a/b);
  else x = 1, y = 0;
}
```

```
// find (x, y) such that a*x + b*y = c or return false if it's not
    possible
// [x + k*b/gcd(a, b), y - k*a/gcd(a, b)] are also solutions
bool diof(ll a, ll b, ll c, ll &x, ll &y){
  euclid(abs(a), abs(b), x, y);
  ll g = abs(__gcd(a, b));
  if(c % g) return false;
  x *= c / g;
  y *= c / g;
  if(a < 0) x = -x;
  if(b < 0) y = -y;
  return true;
}

// auxiliar to find_all_solutions
void shift_solution (ll &x, ll &y, ll a, ll b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}

// Find the amount of solutions of
// ax + by = c
// in given intervals for x and y
ll find_all_solutions (ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll
    maxy) {
  ll x, y, g = __gcd(a, b);
  if(!diof(a, b, c, x, y)) return 0;
  a /= g; b /= g;

  int sign_a = a>0 ? +1 : -1;
  int sign_b = b>0 ? +1 : -1;

  shift_solution (x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift_solution (x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  int lx1 = x;

  shift_solution (x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift_solution (x, y, a, b, -sign_b);
  int rx1 = x;
```

```
  shift_solution (x, y, a, b, - (miny - y) / a);
  if (y < miny)
    shift_solution (x, y, a, b, -sign_a);
  if (y > maxy)
    return 0;
  int lx2 = x;

  shift_solution (x, y, a, b, - (maxy - y) / a);
  if (y > maxy)
    shift_solution (x, y, a, b, sign_a);
  int rx2 = x;

  if (lx2 > rx2)
    swap (lx2, rx2);
  int lx = max (lx1, lx2);
  int rx = min (rx1, rx2);

  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}

bool crt_auxiliar(ll a, ll b, ll m1, ll m2, ll &ans){
  ll x, y;
  if(!diof(m1, m2, b - a, x, y)) return false;
  ll lcm = m1 / __gcd(m1, m2) * m2;
  ans = ((a + x % (lcm / m1) * m1) % lcm + lcm) % lcm;
  return true;
}

// find ans such that ans = a[i] mod b[i] for all 0 <= i < n or return
    false if not possible
// ans + k * lcm(b[i]) are also solutions
bool crt(int n, ll a[], ll b[], ll &ans){
  if(!b[0]) return false;
  ans = a[0] % b[0];
  ll l = b[0];
  for(int i = 1; i < n; i++){
   if(!b[i]) return false;
    if(!crt_auxiliar(ans, a[i] % b[i], l, b[i], ans)) return false;
    l *= (b[i] / __gcd(b[i], l));
  }
  return true;
}
```

7.11 fast-gcd

```
int gcd(int a, int b) {
  if (!a || !b)
    return a | b;
  unsigned shift = __builtin_ctz(a | b);
  a >>= __builtin_ctz(a);
  do {
    b >>= __builtin_ctz(b);
    if (a > b)
      swap(a, b);
    b -= a;
  } while (b);
  return a << shift;
}
```

7.12 formulas

```
//Stirling number of the second kind is the number of ways to partition a
    set of n objects into k non-empty subsets.
S(n,k)= ks (n-1,k)+S(n-1,k-1),where S(0,0)=1,s(n,0)=s(0,n)=0
S(n,2)=2^(n-1)-1
S(n,k) k !=number of ways to color n nodes using colors from 1 to k such
    that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways
    to partition a set of n objects into k subsets,
with each subset containing at least r elements.It is denoted by Sr(n,k)
    and obeys the recurrence relation.
Sr(n+1,k)=kSr(n,k)+C(n,r-1)Sr(n-r+1,k-1)
//The Stirling numbers of the first kind count permutations according to
    their number of cycles (counting fixed points as cycles of length
    one).
S(n,k) counts the number of permutations of n elements with k disjoint
    cycles.
S(n,k) = (n-1)S(n-1,k)+S(n-1,k-1),where,s(0,0) =1,S(n,0)=s(0,n)=0
Sum(k,0,n) S(n,k) = n!
The unsigned Stirling numbers may also be defined algebraically, as the
    coefficient of the rising factorial:
    x^(~n)=x(x+1)(x+n1)=Sum(k,0,n)s(n,k)x^k
//Bell number count the number of partition of a set
Bn+1 = Sum(k,0,n){C(n,k)*Bk}
Bn = S Sum(k,0,n)Sr(n,k), where Sr is Stirling number of 2kind
```

```
//Formally, for a sequence of numbers {ai}, we define the ordinary
    generating function (OGF) of a to be A(x)=Sum(i,0,inf)aix^i.
1/(1 x ) = 1+ x + x^2+...= Sum(n,0,inf)x^n
 ln (1 x )=x + x^2/2 + x^3/3+...=Sum(n,0,inf)x^n/n
e^x=1+x + x^2/2! + x^3/3!+...=Sum(n,0,inf)x^n/n!
(1 x )^ k =C( k1 ,0)x^0+C(k,1)x^1+C(k+1,2)x^2+...=Sum(n,0,inf)C(n+k-1,n)x^n
For OGF, C(x)=A(x)^k  generates the sequence
    cn=Sum(i1...ik,i1+i2+...+ik=n)(ai1*ai2...*aik)
For EGF, C(x)=A(x)^k  generates the sequence
cn=Sum(i1...ik,i1+i2+...+ik=n)(ai1*ai2...*aik)*n!/(i1!*...ik!)
Suppose want to generate the sequence cn=a0+a1+...+an. Then, we can take
    C(x)=1/(1x)*A(x).
```

7.13 gauss mod prime

```
//ll A[N][M+1], X[M]

for(int j=0; j<m; j++) { //collumn to eliminate
  int l = j;
  for(int i=j+1; i<n; i++) //find nonzero pivot
    if(A[i][j]%p)
      l=i;
  for(int k = 0; k < m+1; k++) { //Swap lines
    swap(A[l][k],A[j][k]);
  }
  for(int i = j+1; i < n; i++) { //eliminate column
    ll t=mulmod(A[i][j],inv(A[j][j],p),p);
    for(int k = j; k < m+1; k++)
      A[i][k]=(A[i][k]-mulmod(t,A[j][k],p)+p)%p;
  }
}

for(int i = m-1; i >= 0; i--) { //solve triangular system
  for(int j = m-1; j > i; j--)
    A[i][m] = (A[i][m] - mulmod(A[i][j],X[j],p)+p)%p;
    X[i] = mulmod(A[i][m],inv(A[i][i],p),p);
}
```

7.14 gauss

```
// resuelve Ax = b, dada la matriz a de n * (m + 1), n ecuaciones y m
    variables, siendo la ultima columna el vector b
// The function returns the number of solutions of the system (0,1,or ).
    if there's at least a solution, it's in ans
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big
    number
int gauss (vector < vector<double> > a, vector<double> & ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;
  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    for (int i=row; i<n; ++i)
      if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
    if (abs (a[sel][col]) < EPS)
      continue;
    for (int i=col; i<=m; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;
    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }
  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }
  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

7.15 interpol o(n) ex

```
// evaluar un "polinomio interpolado" en o(nlogMOD)
// debe cumplir xi+1 -  xi  = xj +1  - xj  for all i,  j  < n
// recibe vector de ys tal que f(i) = y[i]
vi fac(tam), sig(tam), facin(tam);
ll eval(vll &ys, ll x) {
       int n = ys.size();
       if(x < n) return ys[x];
       ll upi, dowi, res = 0;
    vi pre(n), suf(n);
    pre[1] = 1, suf[n - 1] = 1;
    fore(i, 1, n)
    {
              if(i > 1)
       pre[i] = pre[i - 1] * (x - i + 1) % MOD;
       suf[n - i - 1] = suf[n - i] * (x - n + i) % MOD;
    }
       fore(i, 1, n) {
       upi = pre[i] * suf[i] % MOD;
       dowi = facin[n - i - 1] * facin[i - 1] % MOD * sig[n - i - 1] %
           MOD;
              res = (res + ys[i] * upi % MOD * dowi % MOD) % MOD;
       }
       return res;
}
```

7.16 josephus

```
// UFMG
/* Josephus Problem - It returns the position to be, in order to not die.
    O(n)*/
/* With k=2, for instance, the game begins with 2 being killed and then
    n+2, n+4, ... */
ll josephus(ll n, ll k) {
  if(n==1) return 1;
  else return (josephus(n-1, k)+k-1)%n+1;
}

/* Another Way to compute the last position to be killed - O(d * log n) */
ll josephus(ll n, ll d) {
  ll K = 1;
  while (K <= (d - 1)*n) K = (d * K + d - 2) / (d - 1);
  return d * n + 1 - K;
}
```

7.17 linear sieve

```
std::vector <int> prime;
bool is_composite[MAXN];

void sieve (int n) {
       std::fill (is_composite, is_composite + n, false);
       for (int i = 2; i < n; ++i) {
              if (!is_composite[i]) prime.push_back (i);
              for (int j = 0; j < prime.size () && i * prime[j] < n;
                  ++j) {
                     is_composite[i * prime[j]] = true;
                     if (i % prime[j] == 0) break;
              }
       }
}

// As is shown in the code, the statement if (i % prime[j] == 0) break;
// terminates the loop when p divides i.
// From the analysis above, we can see that the inner loop is executed
    only once for each composite.
// Hence, the code above performs in O(n) complexity, resulting in its
    name  the 'linear' sieve.
```

7.18 matrix-determinant

```
const double EPS=1e-4;
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;
}
```

7.19 moebius

```
//f(n)=sum(d|n,g(d))=>g(n)=sum(d|n,f(d)*mu(n/d))
//f(d)=sum(i->inf,g(i*d)*mu(i));f(d)=#func(a)->d;g(d)=#d|func(a)
//sum(d|n,mu(d))=e(n)=[n=1]
int mu[tam], is_prime [tam];
fore(i, 0, tam) mu[i]=is_prime[i]=1;
fore(i, 2, tam) if(is_prime[i]) {
  forg(j, i, tam, i) {
    if(j > i) is_prime[j] = 0;
    if(j / i % i == 0) mu[j]=0;
    mu[j] = -mu[j];
  }
}
```

7.20 pollard-rho-miller-rabil

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

```
  }
  if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
  else if(rabin(n))f[n]++;
  else {ll q=rho(n);fact(q,f);fact(n/q,f);}
}
```

7.21 simpson

```
double simpson(double f(double), double a, double b) {
        int n = 100000;
        double s = f(a) + f(b);
        double h = (b - a) / n;
        fore(i, 1, n)
                s += ((i & 1) ? 4 : 2) * f(a + h * i);
        return s * (h / 3);
}
```

7.22 stirling y bell

```
// stir[n][k] cantidad de formas de paritcionar un conjunto de n
    elementos en k conjuntos
// bell[n] cantidad de formas de particionar un conjunto
ll stir[tam][tam];
ll bell[tam];
void stirBell() {
  fore(i, 1, tam) {
  stir[i][1] = 1;
  fore(j, 2, 1010)
    stir[i][j] = (j * stir[i - 1][j] % MOD + stir[i - 1][j - 1]) % MOD;
  }
  fore(i, 1, tam)
  fore(j, 1, i + 1)
    bell[i] = (bell[i] + stir[i][j]) % MOD;
}
```

7.23 tonelly shanks

```
ll legendre(ll a, ll p){
  if(a%p==0)return 0; if(p==2)return 1;
  return fpow(a,(p-1)/2,p);
}
ll tonelli_shanks(ll n, ll p){ // sqrt(n) mod p (p must be a prime)
  assert(legendre(n,p)==1); if(p==2)return 1;
  ll s=__builtin_ctzll(p-1), q=(p-1LL)>>s, z=rnd(1,p-1);
  if(s==1)return fpow(n,(p+1)/4LL,p);
  while(legendre(z,p)!=p-1)z=rnd(1,p-1);
  ll c=fpow(z,q,p), r=fpow(n,(q+1)/2,p), t=fpow(n,q,p), m=s;
  while(t!=1){
    ll i=1, ts=(t*t)%p;
    while(ts!=1)i++,ts=(ts*ts)%p;
    ll b=c;
    fore(_,0,m-i-1)b=(b*b)%p;
    r=r*b%p;c=b*b%p;t=t*c%p;m=i;
  }
  return r;
}
```

8 other

8.1 discrete log

```
// Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = 1; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur];
            return ans;
```

```
        }
    }
    return -1;
}
```

8.2 dsu

```
int cnt[maxn];
void dfs(int v, int p, bool keep){
  int mx = -1, bigChild = -1;
  for(auto u : g[v])
     if(u != p && sz[u] > mx)
      mx = sz[u], bigChild = u;
  for(auto u : g[v])
    if(u != p && u != bigChild)
      dfs(u, v, 0); // run a dfs on small childs and clear them from cnt
  if(bigChild != -1)
    dfs(bigChild, v, 1); // bigChild marked as big and not cleared from
        cnt
  for(auto u : g[v])
  if(u != p && u != bigChild)
    for(int p = st[u]; p < ft[u]; p++)
    cnt[ col[ ver[p] ] ]++;
  cnt[ col[v] ]++;
  //now cnt[c] is the number of vertices in subtree of vertex v that has
      color c. You can answer the queries easily.
  if(keep == 0)
    for(int p = st[v]; p < ft[v]; p++)
    cnt[ col[ ver[p] ] ]--;
}
```

8.3 merge sort

```
// Merge-sort with inversion count - O(nlog n)

int n, inv;
vector<int> v, ans;

void mergesort(int l, int r, vector<int> &v){
    if(l == r) return;
    int mid = (l+r)/2;
    mergesort(l, mid, v), mergesort(mid+1, r, v);
    int i = l, j = mid + 1, k = l;
    while(i <= mid or j <= r){
        if(i <= mid and (j > r or v[i] <= v[j])) ans[k++] = v[i++];
        else ans[k++] = v[j++], inv += j-k;
    }
    for(int i = l; i <= r; i++) v[i] = ans[i];
}

//in main
ans.resize(v.size());
```

8.4 mo

```
void remove(idx);
void add(idx);
int get_answer();
int block_size;
struct Query {
  int l, r, idx;
  bool operator<(Query other) const
  {
    if (l / block_size != other.l / block_size)
      return mp(l, r) < mp(other.l, other.r);
    return ((l / block_size) & 1) ? (r < other.r) : (r > other.r);
  }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());
  int cur_l = 0;
  int cur_r = -1;
  for (Query q : queries) {
    while (cur_l > q.l) {
      cur_l--;
      add(cur_l);
    }
    while (cur_r < q.r) {
      cur_r++;
      add(cur_r);
    }
    while (cur_l < q.l) {
      remove(cur_l);
```

```
      cur_l++;
    }
    while (cur_r > q.r) {
      remove(cur_r);
      cur_r--;
    }
    answers[q.idx] = get_answer();
  }
  return answers;
}
```

8.5 parallel$_b$*inarysearch*

```
// Parallel Binary Search - O(nlog n * cost to update data structure +
    qlog n * cost for binary search condition)

struct Query { int i, ans; /*+ query related info*/ };
vector<Query> req;

void pbs(vector<Query>& qs, int l /* = min value*/, int r /* = max
    value*/) {
  if (qs.empty()) return;

  if (l == r) {
    for (auto& q : qs) req[q.i].ans = l;
    return;
  }

  int mid = (l + r) / 2;
  // mid = (l + r + 1) / 2 if different from simple upper/lower bound

  for (int i = l; i <= mid; i++) {
    // add value to data structure
  }

  vector<Query> vl, vr;
  for (auto& q : qs) {
    if (/* cond */) vl.push_back(q);
    else vr.push_back(q);
  }

  pbs(vr, mid + 1, r);

  for (int i = l; i <= mid; i++) {
    // remove value from data structure
  }

  pbs(vl, l, mid);
}
```

8.6 ternary search

```
//Ternary Search - O(log(n))
//Max version, for minimum version just change signals

ll ternary_search(ll l, ll r){
    while(r - l > 3) {
        ll m1 = (l+r)/2;
        ll m2 = (l+r)/2 + 1;
        ll f1 = f(m1), f2 = f(m2);
        //if(f1 > f2) l = m1;
        if (f1 < f2) l = m1;
        else r = m2;
    }
    ll ans = 0;
    for(int i = l; i <= r; i++){
        ll tmp = f(i);
        //ans = min(ans, tmp);
        ans = max(ans, tmp);
    }
    return ans;
}

//Faster version - 300 iteratons up to 1e-6 precision
double ternary_search(double l, double r, int No = 300){
    // for(int i = 0; i < No; i++){
    while(r - l > EPS){
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        // if (f(m1) > f(m2))
        if (f(m1) < f(m2))
            l = m1;
        else
            r = m2;
    }
    return f(l);
```

```
}
```

9 shortcuts

9.1 Better random

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// mt19937_64 para 64 bits
cout << rng() << endl;

vi permutation(10);
fore(i, 0, 10) permutation[i] = i;
       shuffle(permutation.begin(), permutation.end(), rng);
fore(i, 0, 10) cout << permutation[i] << ' ';
```

9.2 Policy based data structures

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//As I know, there is no implemented tree multiset in STL.
//However you can use pair<T,int> as a key where the second element in
    pair is the time when item has been added.
typedef
    tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update>
    ordered_set;

int main()
{
  ordered_set X;
  X.insert(1);
  X.insert(2);
  X.insert(4);
  X.insert(8);
  X.insert(16);
  cout<<*X.find_by_order(1)<<endl; // 2
  cout<<*X.find_by_order(2)<<endl; // 4
  cout<<*X.find_by_order(4)<<endl; // 16
  cout<<(end(X)==X.find_by_order(6))<<endl; // true

  cout<<X.order_of_key(-5)<<endl; // 0
  cout<<X.order_of_key(1)<<endl; // 0
  cout<<X.order_of_key(3)<<endl; // 2
  cout<<X.order_of_key(4)<<endl; // 2
  cout<<X.order_of_key(400)<<endl; // 5

  return 0;
}
```

9.3 cpp 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.
__builtin_parity(x):
// the parity (even or odd) of the number of ones in the bit
    representation
int __builtin_popcountll (unsigned long long x)
// Get the value of the least significant bit that is one.
v=(x&(-x))
```

9.4 dates

```
int dateToInt(int y, int m, int d){
	return 1461*(y+4800+(m-14)/12)/4+367*(m-2-(m-14)/12*12)/12-
		3*((y+4900+(m-14)/12)/100)/4+d-32075;
}
void intToDate(int jd, int& y, int& m, int& d){
	int x,n,i,j;x=jd+68569;
	n=4*x/146097;x-=(146097*n+3)/4;
	i=(4000*(x+1))/1461001;x-=1461*i/4-31;
	j=80*x/2447;d=x-2447*j/80;
	x=j/11;m=j+2-12*x;y=100*(n-49)+i+x;
}
int DayOfWeek(int d, int m, int y){ //starting on Sunday
	static int ttt[]={0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
	y-=m<3;
	return (y+y/4-y/100+y/400+ttt[m-1]+d)%7;
}
```

9.5 gp hash table(mapa rapido)

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct my_hash {
    const uint64_t RANDOM =
        chrono::steady_clock::now().time_since_epoch().count();
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        return splitmix64(x + RANDOM);
    }
};
// gp_hash_table<int,int,my_hash> m;
```

9.6 harmonic lemma

```
for (int i = 1, la; i <= n; i = la + 1) {
	la = n / (n / i);
	//n / x yields the same value for i <= x <= la.
}
```

9.7 prime numbers

```
  2   3   5   7  11  13  17  19  23  29
 31  37  41  43  47  53  59  61  67  71
 73  79  83  89  97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733
739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997 1009 1013
1019 1021 1031 1033 1039 1049 1051 1061 1063 1069
1087 1091 1093 1097 1103 1109 1117 1123 1129 1151
1153 1163 1171 1181 1187 1193 1201 1213 1217 1223
1229 1231 1237 1249 1259 1277 1279 1283 1289 1291
1297 1301 1303 1307 1319 1321 1327 1361 1367 1373
1381 1399 1409 1423 1427 1429 1433 1439 1447 1451
1453 1459 1471 1481 1483 1487 1489 1493 1499 1511
1523 1531 1543 1549 1553 1559 1567 1571 1579 1583
1597 1601 1607 1609 1613 1619 1621 1627 1637 1657
1663 1667 1669 1693 1697 1699 1709 1721 1723 1733
1741 1747 1753 1759 1777 1783 1787 1789 1801 1811
1823 1831 1847 1861 1867 1871 1873 1877 1879 1889
1901 1907 1913 1931 1933 1949 1951 1973 1979 1987

      970'997      971'483  921'281'269  999'279'733
1'000'000'009 1'000'000'021 1'000'000'409 1'005'012'527
```

9.8 python fast in

```
import sys
inputs = sys.stdin.read().splitlines()
con = 1
ln = 0
while True:
        a, b = map(int, inputs[ln].split())
        ln += 1
        # a = int(input())
        # b = int(input())
        if a == 0:
                break
        su = 0
        for i in range (a):
                su += int(inputs[ln])
                ln += 1
        print(f"Bill #{con} costs {su}: each friend should pay {su // b}")
        print()
        con += 1
```

9.9 python

```
# reopen
import sys
sys.stdout = open('out','w')
sys.stdin = open('in' ,'r')

//Dummy example
R = lambda: map(int, input().split())
n, k = R(),
v, t = [], [0]*n
for p, c, i in sorted(zip(R(), R(), range(n))):
    t[i] = sum(v)+c
    v += [c]
    v = sorted(v)[::-1]
    if len(v) > k:
        v.pop()
print(' '.join(map(str, t)))
```

9.10 shortcuts

```
// Better random mt19937_64 para 64 bits
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
cout << rng() << endl;
shuffle(permutation.begin(), permutation.end(), rng);
// while TLE
double t = clock(), TLE = 3;
while((clock() - t) / CLOCKS_PER_SEC < TLE);
// ordered_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 kth largest 0 indexed, order_of_key finds how many are
    less than
// Faster map gp_hash_table<int,int,my_hash> m;
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct my_hash {
  const uint64_t RANDOM =
      chrono::steady_clock::now().time_since_epoch().count();
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    return splitmix64(x + RANDOM);
  }
};
```

9.11 theo

```
 ErdsGallai  theorem
A sequence of non-negative integers d1>=...>=d_n can be represented
as the degree sequence of a finite simple graph on n vertices if and only
    if
d1+...+dn is even and sum(i=1->k, di) <= k*(k - 1) + sum(i=k+1->n,
    min(di, k))
holds for every k in 1<=k<=n
teorema de Erds-Szekeres
```

```
Para r, s dados, cualquier sucesin de longitud al menos (r-1)(s-1)+1
    contiene
una subsucesin montonamente creciente de longitud r o una subsucesin
montonamente decreciente de longitud s
The harmonic lemma
(n/1, n/2, ..., n/n) has at most 2*sqrt(n) different elements
greatest x that satisfy d(x) = d(i) is n / (n / i)
(Tutte) A graph, G = (V, E), has a perfect matching if and only if for
    every subset U of V, the subgraph induced by V - U has at most |U|
    connected components with an odd number of vertices.
Petersens Theorem. Every cubic, bridgeless graph contains a perfect
    matching.
(Dilworth) In any finite partially ordered set, the maximum number of
    elements in any antichain equals the minimum number of chains in any
    partition of the set into chains
Pick: A=I+B/2-1 (area of polygon, points inside, points on border)
```

9.12 while TLE

```
double TLE = 3;
double t = clock();
while((clock() - t) / CLOCKS_PER_SEC < TLE)
{

}
```

10 strings

10.1 AlgoritmoZ

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

10.2 KMP

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

const int tam = 1e5 + 10;
int aut[tam][26];

void genAut(string &s) {
    clr(aut, 0);
    vi pi = kmp(s);
    int n = s.size();
    fore(i, 0, n) {
        aut[i][s[i]-'a'] = i+1;
    }
    fore(i, 0, n+1) {
        for (int c = 0; c < 26; c++) {
            if (aut[i][c]) continue;
            if (i < n && s[i] == 'a' + c) aut[i][c] = i+1;
            else if (i > 0) aut[i][c] = aut[pi[i-1]][c];
            else aut[i][c] = 0;
        }
    }
}
```

10.3 aho-corasick

```
struct vertex {
  int go[26], pch, par, link = -1, super = -1, leaf = 0;
  vertex(): link(0), super(0) { mem(go, -1); }
  vertex(int ch, int from): pch(ch), par(from) { mem(go, -1); }
};
vector<vertex> t(1);
void add(string &s, int pos) {
  int node = 0;
  for(char ch : s) {
    ch -= 'a';
    if(t[node].go[ch] == -1)
      t[node].go[ch] = t.size(); t.emplace_back(ch, node);
    node = t[node].go[ch];
  }
  t[node].leaf = 1;
}
int go(int node, char c);
int suff(int node) {
  if(t[node].link == -1)
    t[node].link = t[node].par == 0 ? 0 : go(suff(t[node].par),
        t[node].pch);
  return t[node].link;
}
int go(int node, char ch) {
  if(t[node].go[ch] == -1)
    t[node].go[ch] = node == 0 ? 0 : go(suff(node), ch);
  return t[node].go[ch];
}
int super(int v) {
  if(t[v].super == -1)
    t[v].super = t[suff(v)].leaf ? suff(v) : super(suff(v));
  return t[v].super;
}
```

10.4 hashing

```
ll pot(ll b, ll e , ll m);
struct Hash
{
       int p = 997, m[2], in[2];
       vector<int> h[2], inv[2];
       Hash(string s)
       {
              m[0] = 998244353, m[1] = 1000000009;
              fore(i, 0, 2)
              {
                     in[i] = pot(p, m[i]-2, m[i]);
                     h[i].resize(s.size() + 1);
                     inv[i].resize(s.size() + 1);
                     ll acu = 1;
                     h[i][0] = 0, inv[i][0] = 1;
                     fore(j, 0, s.size())
                     {
                            h[i][j + 1] = (h[i][j] + acu * s[j]) % m[i];
                            inv[i][j + 1] = (1ll * inv[i][j] * in[i]) %
                                m[i];
                            acu = (acu * p) % m[i];
                     }
              }
       }
       ll get(int b, int e)
       {
              ll ha[2];
              fore(i, 0, 2)
                     ha[i] = ((((h[i][e] - h[i][b]) * (ll)inv[i][b]) %
                         m[i]) + m[i]) % m[i];
              return((ha[0] << 32) | ha[1]) ;
       }
};
```

10.5 hsh-128

```
#define bint __int128
struct Hash {
  bint MOD=212345678987654321LL,P=1777771,PI=10695574108965957 1LL;
  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;
    fore(i,1,s.size()+1){
      h[i]=(h[i-1]+p*s[i-1])%MOD;
```

```
      pi[i]=(pi[i-1]*PI)%MOD;
      p=(p*P)%MOD;
    }
  }
  ll get(int s, int e){
    return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
  }
};
```

10.6 manacher

```
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();
  fore(i,0,n){
    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;
  fore(i,0,n){
    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;
  }
}
```

10.7 suffix-array

```
vector<vector<int>> table;
vector<int> suffixa(string &s){
  int n = s.size(), cc, ax;
  vector<int> sa(n), sa1(n), col(n), col1(n), head(n);
  fore(i, 0, n) sa[i] = i;
  auto cmp = [&](int a, int b){ return s[a] < s[b]; };
  stable_sort(sa.begin(), sa.end(), cmp);
  head[0] = col[sa[0]] = cc = 0;
  fore(i, 1, n){
    if(s[sa[i]] != s[sa[i-1]])
      cc++, head[cc] = i;
    col[sa[i]] = cc;
  }
  table.pb(col);
  for(int k = 1; k < n; k *= 2){
    fore(i, 0, n){
      ax = (sa[i] - k + n) % n;
      sa1[head[col[ax]]++] = ax;
    }
    swap(sa, sa1);
    col1[sa[0]] = head[0] = cc = 0;
    fore(i, 1, n){
      if(col[sa[i]] != col[sa[i - 1]] || col[(sa[i] + k) % n] !=
          col[(sa[i - 1] + k) % n])
        cc++, head[cc] = i;
      col1[sa[i]] = cc;
    }
    swap(col, col1); table.pb(col);
    if(col[sa[n - 1]] == n - 1) break;
  }
  return sa;
}
pair<int, int> query(int b, int e){
  int lev = 31 - __builtin_clz(e - b + 1);
  return mp(table[lev][b], table[lev][e - (1 << lev) + 1]);
}
bool comp(int b1, int e1, int b2, int e2){
  int siz = min(e1 - b1, e2 - b2);
  ii le = query(b1, b1 + siz), ri = query(b2, b2 + siz);
  if(le == ri)
    return e1 - b1 < e2 - b2;
  return le < ri;
}
vector<int> lcp(string &s, vector<int> &sa){
  int n = s.size(), k, z = 0;
  vector<int> sa1(n), lcp(n);
  fore(i, 0, n) sa1[sa[i]] = i;
  fore(i, 0, n){
    k = sa1[i];
    if(k < n - 1)
      while(s[i + z] == s[sa[k+1] + z])
```

```
      z++;
    lcp[k] = z; z = max(z-1, 0);
  }
  return lcp;
}
```