Índice

1. algorithm

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i ∈[f,l) tq. i=elem, pred(i), i∈[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) ∈ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, . . .) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, . . .)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i$ ∈[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Estructuras

2.1. RMQ (static)

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion: LVL $\geq$ ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
struct RMQ{
  #define LVL 10
  tipo vec[LVL][1<<(LVL+1)];
  tipo &operator[](int p){return vec[0][p];}
  tipo get(int i, int j) {//intervalo [i,j)
    int p = 31-__builtin_clz(j-i);
    return min(vec[p][i],vec[p][j-(1<<p)]);
  }
  void build(int n) {//O(nlogn)
    int mp = 31-__builtin_clz(n);
    forn(p, mp) forn(x, n-(1<<p))
      vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
  }};
```

2.2. RMQ (dynamic)

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
     sobre el rango [i, j).
#define MAXN 100000
#define operacion(x, y) max(x, y)
const int neutro=0;
struct RMQ{
  int sz;
  tipo t[4*MAXN];
  tipo &operator[](int p){return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i]=neutro;
  }
  void updall(){//O(n)
    dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
  tipo get(int i, int j){return get(i,j,1,0,sz);}
  tipo get(int i, int j, int n, int a, int b){//O(lgn)
    if(j<=a || i>=b) return neutro;
    if(i<=a && b<=j) return t[n];
    int c=(a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  void set(int p, tipo val){//O(lgn)
    for(p+=sz; p>0 && t[p]!=val;){
      t[p]=val;
      p/=2;
      val=operacion(t[p*2], t[p*2+1]);
    }
  }
}rmq;
//Usage:
cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3. RMQ (lazy)

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
     sobre el rango [i, j).
typedef int Elem;//Elem de los elementos del arreglo
typedef int Alt;//Elem de la alteracion
#define operacion(x,y) x+y
const Elem neutro=0; const Alt neutro2=0;
#define MAXN 100000
struct RMQ{
  int sz;
  Elem t[4*MAXN];
  Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem
  Elem &operator[](int p){return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i]=neutro;
    forn(i, 2*sz) dirty[i]=neutro2;
  }
  void push(int n, int a, int b){//propaga el dirty a sus hijos
    if(dirty[n]!=0){
      t[n]+=dirty[n]*(b-a);//altera el nodo
      if(n<sz){
        dirty[2*n]+=dirty[n];
        dirty[2*n+1]+=dirty[n];
      }
      dirty[n]=0;
    }
  }
  Elem get(int i, int j, int n, int a, int b){//O(lgn)
```

```
    if(j<=a || i>=b) return neutro;
    push(n, a, b);//corrige el valor antes de usarlo
    if(i<=a && b<=j) return t[n];
    int c=(a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  Elem get(int i, int j){return get(i,j,1,0,sz);}
  //altera los valores en [i, j) con una alteracion de val
  void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
    push(n, a, b);
    if(j<=a || i>=b) return;
    if(i<=a && b<=j){
      dirty[n]+=val;
      push(n, a, b);
      return;
    }
    int c=(a+b)/2;
    alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
    t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
  }
  void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
}rmq;
```

2.4. RMQ (persistente)

```
typedef int tipo;
tipo oper(const tipo &a, const tipo &b){
    return a+b;
}
struct node{
  tipo v; node *l,*r;
  node(tipo v):v(v), l(NULL), r(NULL) {}
    node(node *l, node *r) : l(l), r(r){
        if(!l) v=r->v;
        else if(!r) v=l->v;
        else v=oper(l->v, r->v);
    }
};
node *build (tipo *a, int tl, int tr) {//modificar para que tome tipo a
  if (tl+1==tr) return new node(a[tl]);
  int tm=(tl + tr)>>1;
  return new node(build(a, tl, tm), build(a, tm, tr));
}
node *update(int pos, int new_val, node *t, int tl, int tr){
  if (tl+1==tr) return new node(new_val);
  int tm=(tl+tr)>>1;
  if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r)
    ;
  else return new node(t->l, update(pos, new_val, t->r, tm, tr));
}
tipo get(int l, int r, node *t, int tl, int tr){
    if(l==tl && tr==r) return t->v;
  int tm=(tl + tr)>>1;
    if(r<=tm) return get(l, r, t->l, tl, tm);
    else if(l>=tm) return get(l, r, t->r, tm, tr);
  return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
}
```

2.5. Fenwick Tree

```
//For 2D threat each column as a Fenwick tree, by adding a nested for in
     each operation
struct Fenwick{
  static const int sz=1000001;
  tipo t[sz];
  void adjust(int p, tipo v){//valid with p in [1, sz), O(lgn)
    for(int i=p; i<sz; i+=(i&-i)) t[i]+=v; }
  tipo sum(int p){//cumulative sum in [1, p], O(lgn)
    tipo s=0;
    for(int i=p; i; i-=(i&-i)) s+=t[i];
    return s;
  }
  tipo sum(int a, int b){return sum(b)-sum(a-1);}
  //get largest value with cumulative sum less than or equal to x;
  //for smallest, pass x-1 and add 1 to result
  int getind(tipo x) {//O(lgn)
      int idx = 0, mask = N;
      while(mask && idx < N) {
        int t = idx + mask;
      if(x >= tree[t])
          idx = t, x -= tree[t];
        mask >>= 1;
      }
      return idx;
  }};
```

2.6. Disjoint Set Union

```
// Fast version, almost O(1). join by size.
class DSUFast {
    public:

        vector <int> parent;
        vector <int> size;

        DSUFast(int sz){
            size = vector<int>(sz, 1);
            parent.resize(sz);
            forn(i, sz)
                parent[i] = i;
        }

        int find(int v){
            if(v == parent[v])
                return v;
            return parent[v] = find(parent[v]);
        }

        void join(int u, int v){
            int oldComp = find(u);
            int newComp = find(v);

            if(oldComp != newComp){
                if(size[oldComp] > size[newComp])
                    swap(oldComp, newComp);

                parent[oldComp] = newComp;
                size[newComp] += size[oldComp];
            }
        }
};

// Slower version, O(log n) average join. Stores each component.

class DSUComp {
    public:

        vector <vector <int>> component;
        vector <int> myComp;

        DSUComp(int size){
            forn(i, size){
                myComp.pb(i);
                component.pb({i});
            }
        }

        int find(int v){
            return myComp[v];
        }

        void join(int u, int v){
            int oldComp = find(u);
            int newComp = find(v);

            if(component[oldComp].size() > component[newComp].size())
                swap(oldComp, newComp);

            for(int w : component[oldComp]){
                component[newComp].pb(w);
                myComp[w] = newComp;
            }

            component[oldComp].clear();
        }

        // Returns a reference!
        vector <int> &getComponent(int v){
            return component[find(v)];
        }
};
```

2.7. Disjoint Intervals

```
// AC - https://codeforces.com/contest/27/submission/47281082
struct disjoint_intervals{ // intervals [first, second)
  set<pll> s;
  void insert(pll v){
    if(v.fst>=v.snd) return;
    auto at=s.lower_bound(v), it=at;
    if(at!=s.begin()&&(--at)->snd>=v.fst) v.fst=at->fst, --it;
    for(;it!=s.end()&&it->fst<=v.snd;s.erase(it++))
```

```
      v.snd=max(v.snd,it->snd);
    s.insert(v);
  }
};
```

2.8. RMQ (2D)

```
struct RMQ2D{//n filas x m columnas
  int sz;
  RMQ t[4*MAXN];
  RMQ &operator[](int p){return t[sz/2+p];}//t[i][j]=i fila, j col
  void init(int n, int m){//O(n*m)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i].init(m); }
  void set(int i, int j, tipo val){//O(lgm.lgn)
    for(i+=sz; i>0;){
      t[i].set(j, val);
      i/=2;
      val=operacion(t[i*2][j], t[i*2+1][j]);
    } }
  tipo get(int i1, int j1, int i2, int j2){return get(i1,j1,i2,j2,1,0,
      sz);}
  //O(lgm.lgn), rangos cerrado abierto
  int get(int i1, int j1, int i2, int j2, int n, int a, int b){
    if(i2<=a || i1>=b) return 0;
    if(i1<=a && b<=i2) return t[n].get(j1, j2);
    int c=(a+b)/2;
    return operacion(get(i1, j1, i2, j2, 2*n, a, c),
         get(i1, j1, i2, j2, 2*n+1, c, b));
  }
} rmq;
//Example to initialize a grid of M rows and N columns:
RMQ2D rmq; rmq.init(n,m);
forn(i, n) forn(j, m){
  int v; cin >> v; rmq.set(i, j, v);}
```

2.9. Big Int

```
#define BASEXP 6
#define BASE 1000000
#define LMAX 1000
struct bint{
    int l;
    ll n[LMAX];
    bint(ll x=0){
        l=1;
        forn(i, LMAX){
            if (x) l=i+1;
            n[i]=x%BASE;
            x/=BASE;

        }
    }
    bint(string x){
    l=(x.size()-1)/BASEXP+1;
        fill(n, n+LMAX, 0);
        ll r=1;
        forn(i, sz(x)){
            n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
            r*=10; if(r==BASE)r=1;
        }
    }
    void out(){
    cout << n[l-1];
    dforn(i, l-1) printf("%6.6llu", n[i]);//6=BASEXP!
  }
  void invar(){
    fill(n+l, n+LMAX, 0);
    while(l>1 && !n[l-1]) l--;
  }
};
bint operator+(const bint&a, const bint&b){
  bint c;
    c.l = max(a.l, b.l);
    ll q = 0;
    forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
    if(q) c.n[c.l++] = q;
    c.invar();
    return c;
}
pair<bint, bool> lresta(const bint& a, const bint& b)   // c = a - b
{
  bint c;
    c.l = max(a.l, b.l);
    ll q = 0;
    forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/
        BASE-1;
    c.invar();
    return make_pair(c, !q);
}
bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).first;}
bint operator- (const bint&a, const bint&b){return lresta(a, b).first;}
bool operator< (const bint&a, const bint&b){return !lresta(a, b).second;}
bool operator<= (const bint&a, const bint&b){return lresta(b, a).second;}
bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
bint operator*(const bint&a, ll b){
    bint c;
    ll q = 0;
    forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
    c.l = a.l;
    while(q) c.n[c.l++] = q %BASE, q/=BASE;
    c.invar();
    return c;
}
bint operator*(const bint&a, const bint&b){
    bint c;
    c.l = a.l+b.l;
    fill(c.n, c.n+b.l, 0);
    forn(i, a.l){
        ll q = 0;
        forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q/=BASE;
        c.n[i+b.l] = q;
    }
    c.invar();
    return c;
}
pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
  bint c;
  ll rm = 0;
  dforn(i, a.l){
            rm = rm * BASE + a.n[i];
            c.n[i] = rm / b;
            rm %= b;
    }
    c.l = a.l;
    c.invar();
    return make_pair(c, rm);
}
bint operator/(const bint&a, ll b){return ldiv(a, b).first;}
ll operator%(const bint&a, ll b){return ldiv(a, b).second;}
pair<bint, bint> ldiv(const bint& a, const bint& b){
  bint c;
    bint rm = 0;
    dforn(i, a.l){
        if (rm.l==1 && !rm.n[0])
            rm.n[0] = a.n[i];
        else{
            dforn(j, rm.l) rm.n[j+1] = rm.n[j];
            rm.n[0] = a.n[i];
            rm.l++;
        }
        ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
        ll u = q / (b.n[b.l-1] + 1);
        ll v = q /  b.n[b.l-1] + 1;
        while (u < v-1){
            ll m = (u+v)/2;
            if (b*m <= rm) u = m;
            else v = m;
        }
        c.n[i]=u;
        rm-=b*u;
    }
  c.l=a.l;
    c.invar();
    return make_pair(c, rm);
}
bint operator/(const bint&a, const bint&b){return ldiv(a, b).first;}
bint operator%(const bint&a, const bint&b){return ldiv(a, b).second;}
```

2.10. MinQueue

```
// AC ARRAYSUB - https://www.spoj.com/status/ns=22910678#
#define op(x,y) min(x,y)
template<class T>
struct min_queue{
    stack<pair<T,T>> s1, s2;
    int size(){ return s1.size() + s2.size(); }
    T get(){
        if(s1.empty() or s2.empty())
            return s1.empty() ? s2.top().snd : s1.top().snd;
        return op(s1.top().snd, s2.top().snd);
    }
    void push(T elem){
        T m = s1.empty() ? elem : op(elem, s1.top().snd);
        s1.push({elem, m});
    }
    T pop(){
        if(s2.empty())
        while(!s1.empty()){
            T elem = s1.top().fst;
            s1.pop();
            T m = s2.empty() ? elem : op(elem, s2.top().snd);
            s2.push({elem, m});
        }
        T res = s2.top().fst;
        s2.pop();
        return res;
    }
};
```

2.11. HashTables

```
//Compilar: g++ --std=c++11
struct Hash{
  size_t operator()(const ii &a)const{
    size_t s=hash<int>()(a.fst);
    return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
  }
  size_t operator()(const vector<int> &v)const{
    size_t s=0;
    for(auto &e : v)
      s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
    return s;
  }
};
unordered_set<ii, Hash> s;
unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

2.12. Modnum

```
struct mnum{
  static const tipo mod=12582917;
  tipo v;
  mnum(tipo v=0): v(v%mod) {}
  mnum operator+(mnum b){return v+b.v;}
  mnum operator-(mnum b){return v>=b.v? v-b.v : mod-b.v+v;}
  mnum operator*(mnum b){return v*b.v;}
  mnum operator^(int n){
    if(!n) return 1;
    return n%2? (*this)^(n/2)*(this) : (*this)^(n/2);}
};
```

2.13. Treap para set

```
typedef int Key;
typedef struct node *pnode;
struct node{
    Key key;
    int prior, size;
    pnode l,r;
    node(Key key=0): key(key), prior(rand()), size(1), l(0), r(0) {}
};
static int size(pnode p) { return p ? p->size : 0; }
void push(pnode p) {
  // modificar y propagar el dirty a los hijos aca(para lazy)
}
// Update function and size from children's Value
void pull(pnode p) {//recalcular valor del nodo aca (para rmq)
  p->size = 1 + size(p->l) + size(p->r);
}
//junta dos arreglos
pnode merge(pnode l, pnode r) {
  if (!l || !r) return l ? l : r;
  push(l), push(r);
  pnode t;
  if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
  else r->l=merge(l, r->l), t = r;
  pull(t);
  return t;
}
//parte el arreglo en dos, l<key<=r
void split(pnode t, Key key, pnode &l, pnode &r) {
    if (!t) return void(l = r = 0);
    push(t);
    if (key <= t->key) split(t->l, key, l, t->l), r = t;
    else split(t->r, key, t->r, r), l = t;
    pull(t);
}

void erase(pnode &t, Key key) {
    if (!t) return;
    push(t);
    if (key == t->key) t=merge(t->l, t->r);
    else if (key < t->key) erase(t->l, key);
    else erase(t->r, key);
    if(t) pull(t);
}

ostream& operator<<(ostream &out, const pnode &t) {
  if(!t) return out;
    return out << t->l << t->key << ' ' << t->r;
}
pnode find(pnode t, Key key) {
    if (!t) return 0;
    if (key == t->key) return t;
    if (key < t->key) return find(t->l, key);
    return find(t->r, key);
}
struct treap {
    pnode root;
    treap(pnode root=0): root(root) {}
    int size() { return ::size(root); }
    void insert(Key key) {
        pnode t1, t2; split(root, key, t1, t2);
        t1=::merge(t1,new node(key));
        root=::merge(t1,t2);
    }
    void erase(Key key1, Key key2) {
        pnode t1,t2,t3;
        split(root,key1,t1,t2);
        split(t2,key2, t2, t3);
        root=merge(t1,t3);
    }
    void erase(Key key) {::erase(root, key);}
    pnode find(Key key) { return ::find(root, key); }
    Key &operator[](int pos){return find(pos)->key;}//ojito
};
treap merge(treap a, treap b) {return treap(merge(a.root, b.root));}
```

2.14. Treap para arreglo

```
typedef struct node *pnode;
struct node{
    Value val, mini;
    int dirty;
    int prior, size;
    pnode l,r,parent;
    node(Value val): val(val), mini(val), dirty(0), prior(rand()), size(1), l(0), r(0), parent(0) {}
};
static int size(pnode p) { return p ? p->size : 0; }
void push(pnode p) {//propagar dirty a los hijos(aca para lazy)
  p->val.fst+=p->dirty;
  p->mini.fst+=p->dirty;
  if(p->l) p->l->dirty+=p->dirty;
  if(p->r) p->r->dirty+=p->dirty;
  p->dirty=0;
}
static Value mini(pnode p) { return p ? push(p), p->mini : ii(1e9, -1); }
// Update function and size from children's Value
void pull(pnode p) {//recalcular valor del nodo aca (para rmq)
  p->size = 1 + size(p->l) + size(p->r);
  p->mini = min(min(p->val, mini(p->l)), mini(p->r));//operacion del rmq!
  p->parent=0;
  if(p->l) p->l->parent=p;
  if(p->r) p->r->parent=p;
}
//junta dos arreglos
pnode merge(pnode l, pnode r) {
  if (!l || !r) return l ? l : r;
  push(l), push(r);
  pnode t;
  if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
  else r->l=merge(l, r->l), t = r;
  pull(t);
  return t;
}
//parte el arreglo en dos, sz(l)==tam
void split(pnode t, int tam, pnode &l, pnode &r) {
  if (!t) return void(l = r = 0);
  push(t);
  if (tam <= size(t->l)) split(t->l, tam, l, t->l), r = t;
  else split(t->r, tam - 1 - size(t->l), t->r, r), l = t;
  pull(t);
}
pnode at(pnode t, int pos) {
  if(!t) exit(1);
  push(t);
  if(pos == size(t->l)) return t;
  if(pos < size(t->l)) return at(t->l, pos);
  return at(t->r, pos - 1 - size(t->l));
}
int getpos(pnode t){//inversa de at
  if(!t->parent) return size(t->l);
  if(t==t->parent->l) return getpos(t->parent)-size(t->r)-1;
  return getpos(t->parent)+size(t->l)+1;
}
void split(pnode t, int i, int j, pnode &l, pnode &m, pnode &r) {
  split(t, i, l, t), split(t, j-i, m, r);}
Value get(pnode &p, int i, int j){//like rmq
  pnode l,m,r;
    split(p, i, j, l, m, r);
    Value ret=mini(m);
    p=merge(l, merge(m, r));
    return ret;
}
void print(const pnode &t) {//for debugging
  if(!t) return;
    push(t);
    print(t->l);
    cout << t->val.fst << '␣';
    print(t->r);
}
```

2.15. Convex Hull Trick

```
struct Line{tipo m,h;};
tipo inter(Line a, Line b){
    tipo x=b.h-a.h, y=a.m-b.m;
    return x/y+(x%y?!((x>0)^(y>0)):0);//==ceil(x/y)
}
struct CHT {
  vector<Line> c;
  bool mx;
  int pos;
  CHT(bool mx=0):mx(mx),pos(0){}//mx=1 si las query devuelven el max
  inline Line acc(int i){return c[c[0].m>c.back().m? i : sz(c)-1-i];}
  inline bool irre(Line x, Line y, Line z){
    return c[0].m>z.m? inter(y, z) <= inter(x, y)
                         : inter(y, z) >= inter(x, y);
  }
  void add(tipo m, tipo h) {//O(1), los m tienen que entrar ordenados
        if(mx) m*=-1, h*=-1;
    Line l=(Line){m, h};
        if(sz(c) && m==c.back().m) { l.h=min(h, c.back().h), c.pop_back(); if(pos) pos--; }
        while(sz(c)>=2 && irre(c[sz(c)-2], c[sz(c)-1], l)) { c.pop_back(); if(pos) pos--; }
        c.pb(l);
  }
  inline bool fbin(tipo x, int m) {return inter(acc(m), acc(m+1))>x;}
  tipo eval(tipo x){
    int n = sz(c);
    //query con x no ordenados O(lgn)
    int a=-1, b=n-1;
    while(b-a>1) { int m = (a+b)/2;
      if(fbin(x, m)) b=m;
      else a=m;
    }
    return (acc(b).m*x+acc(b).h)*(mx?-1:1);
        //query O(1)
    while(pos>0 && fbin(x, pos-1)) pos--;
    while(pos<n-1 && !fbin(x, pos)) pos++;
    return (acc(pos).m*x+acc(pos).h)*(mx?-1:1);
  }
} ch;
```

2.16. Convex Hull Trick (Dynamic)

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable multiset<Line>::iterator it;
    const Line *succ(multiset<Line>::iterator it) const;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
```

```cpp
        const Line *s=succ(it);
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line>{ // will maintain upper hull for maximum
    bool bad(iterator y) {
        iterator z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        iterator x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    iterator next(iterator y){return ++y;}
    iterator prev(iterator y){return --y;}
    void insert_line(ll m, ll b) {
        iterator y = insert((Line) { m, b });
        y->it=y;
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        Line l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
}h;
const Line *Line::succ(multiset<Line>::iterator it) const{
    return (++it==h.end()? NULL : &*it);}
```

2.17. Gain-Cost Set

```cpp
//esta estructura mantiene pairs(beneficio, costo)
//de tal manera que en el set quedan ordenados
//por beneficio Y COSTO creciente. (va borrando los que no son optimos)
struct V{
  int gain, cost;
  bool operator<(const V &b)const{return gain<b.gain;}
};
set<V> s;
void add(V x){
  set<V>::iterator p=s.lower_bound(x);//primer elemento mayor o igual
  if(p!=s.end() && p->cost <= x.cost) return;//ya hay uno mejor
  p=s.upper_bound(x);//primer elemento mayor
  if(p!=s.begin()){//borro todos los peores (<=beneficio  y >=costo)
    --p;//ahora es ultimo elemento menor o igual
    while(p->cost >= x.cost){
      if(p==s.begin()){s.erase(p); break;}
      s.erase(p--);
    }
  }
  s.insert(x);
}
int get(int gain){//minimo costo de obtener tal ganancia
  set<V>::iterator p=s.lower_bound((V){gain, 0});
  return p==s.end()? INF : p->cost;}
```

2.18. Set con busq binaria

```cpp
#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>,//key,mapped type, comparator
    rb_tree_tag,tree_order_statistics_node_update> set_t;
//find_by_order(i) devuelve iterador al i-esimo elemento
//order_of_key(k): devuelve la pos del lower bound de k
//Ej: 12, 100, 505, 1000, 10000.
//order_of_key(10) == 0, order_of_key(100) == 1,
//order_of_key(707) == 3, order_of_key(9999999) == 5
```

2.19. Fast Hash Table

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
```

3. Algos

3.1. Longest Increasing Subsecuence

```
//Para non-increasing, cambiar comparaciones y revisar busq binaria
//Given an array, paint it in the least number of colors so that each color turns to a non-increasing subsequence.
//Solution:Min number of colors=Length of the longest increasing subsequence
int N, a[MAXN];//secuencia y su longitud
ii d[MAXN+1];//d[i]=ultimo valor de la subsecuencia de tamanio i
int p[MAXN];//padres
vector<int> R;//respuesta
void rec(int i){
 if(i==-1) return;
 R.push_back(a[i]);
 rec(p[i]);
}
int lis(){//O(nlogn)
 d[0] = ii(-INF, -1); forn(i, N) d[i+1]=ii(INF, -1);
 forn(i, N){
  int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
  if (d[j-1].first < a[i]&&a[i] < d[j].first){
   p[i]=d[j-1].second;
   d[j] = ii(a[i], i);
  }
 }
 R.clear();
 dforn(i, N+1) if(d[i].first!=INF){
  rec(d[i].second);//reconstruir
  reverse(R.begin(), R.end());
  return i;//longitud
 }
 return 0;
}
```

3.2. Alpha-Beta prunning

```
ll alphabeta(State &s, bool player = true, int depth = 1e9, ll alpha = -INF, ll beta = INF) { //player = true -> Maximiza
  if(s.isFinal()) return s.score;
 //~ if (!depth) return s.heuristic();
  vector<State> children;
  s.expand(player, children);
  int n = children.size();
  forn(i, n) {
    ll v = alphabeta(children[i], !player, depth-1, alpha, beta);
    if(!player) alpha = max(alpha, v);
    else beta = min(beta, v);
    if(beta <= alpha) break;
  }
  return !player ? alpha : beta;}
```

3.3. Mo's algorithm

```
int n,sq;
struct Qu{//queries [l, r]
  //intervalos cerrado abiertos !!! importante!!
  int l, r, id;
}qs[MAXN];
int ans[MAXN], curans;//ans[i]=ans to ith query
bool bymos(const Qu &a, const Qu &b){
  if(a.l/sq!=b.l/sq) return a.l<b.l;
  return (a.l/sq)&1? a.r<b.r : a.r>b.r;
}
void mos(){
  forn(i, t) qs[i].id=i;
  sort(qs, qs+t, bymos);
  int cl=0, cr=0;
  sq=sqrt(n);
  curans=0;
  forn(i, t){ //intervalos cerrado abiertos !!! importante!!
    Qu &q=qs[i];
    while(cl>q.l) add(--cl);
    while(cr<q.r) add(cr++);
    while(cl<q.l) remove(cl++);
    while(cr>q.r) remove(--cr);
    ans[q.id]=curans;
  }
}
```

4. Strings

4.1. Manacher

```
int d1[MAXN];//d1[i]=long del maximo palindromo impar con centro en i
int d2[MAXN];//d2[i]=analogo pero para longitud par
//0 1 2 3 4
//a a b c c <--d1[2]=3
//a a b b <--d2[2]=2 (estan uno antes)
void manacher(){
  int l=0, r=-1, n=sz(s);
  forn(i, 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;
  forn(i, n){
    int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
    while(i+k-1<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;
  }
```

4.2. KMP

```
string T;//cadena donde buscar(where)
string P;//cadena a buscar(what)
int b[MAXLEN];//back table b[i] maximo borde de [0..i)
void kmppre(){//by gabina with love
    int i =0, j=-1; b[0]=-1;
    while(i<sz(P)){
        while(j>=0 && P[i] != P[j]) j=b[j];
        i++, j++, b[i] = j;
    }
}
void kmp(){
    int i=0, j=0;
    while(i<sz(T)){
        while(j>=0 && T[i]!=P[j]) j=b[j];
        i++, j++;
        if(j==sz(P)) printf("P␣is␣found␣at␣index␣%d␣in␣T\n", i-j), j=b[j];
    }
}

int main(){
    cout << "T=";
    cin >> T;
    cout << "P=";
```

4.3. Trie

```
struct trie{
  map<char, trie> m;
  void add(const string &s, int p=0){
    if(s[p]) m[s[p]].add(s, p+1);
  }
  void dfs(){
    //Do stuff
    forall(it, m)
      it->second.dfs();
  }
};
```

4.4. Hash

```
struct Hash {
  int p = 1777771, mod[2], pInv[2];
  //~ h[k][i] = hash[k]( s[0, i) )
  //~ p**i * potInv[k][i] = 1 modulo mod[k]
  vector<int> h[2],potInv[2];
  Hash(string& s){
    mod[0] = 999727999; mod[1] = 1070777777;
    //~ Modular inverse of p
    pInv[0] = 325255434; pInv[1] = 10018302;
    forn(k,2){
      h[k].resize(s.size()+1);
      potInv[k].resize(s.size()+1);
    }
    forn(k,2){
      h[k][0] = 0;
      potInv[k][0] = 1;
      ll pAcum = 1;
      forr(i, 1, s.size()+1){
        h[k][i] = (h[k][i-1] + pAcum * s[i-1]) % mod[k];
        potInv[k][i] = (1LL * potInv[k][i-1] * pInv[k]) % mod[k];
        pAcum = (pAcum * p) % mod[k];
      }
    }
  }
  //~ get(i, j) = hash( s[i,j) )
  ll get(int s, int e){
    ll hashes[2];
    forn(k, 2){
      hashes[k] = (h[k][e] - h[k][s] + mod[k]) % mod[k];
      hashes[k] = (1LL * hashes[k] * potInv[k][s]) % mod[k];
    }
    return (hashes[0]<<32)|hashes[1];
  }
};
```

4.5. Suffix Array (largo, nlogn)

```
#define MAX_N 1000
#define rBOUND(x) (x<n? r[x] : 0)
//sa will hold the suffixes in order.
int sa[MAX_N], r[MAX_N], n;
string s; //input string, n=sz(s)

int f[MAX_N], tmpsa[MAX_N];
void countingSort(int k){
  zero(f);
  forn(i, n) f[rBOUND(i+k)]++;
  int sum=0;
  forn(i, max(255, n)){
    int t=f[i]; f[i]=sum; sum+=t;}
  forn(i, n)
    tmpsa[f[rBOUND(sa[i]+k)]++]=sa[i];
  memcpy(sa, tmpsa, sizeof(sa));
}
void constructsa(){//O(n log n)
  n=sz(s);
  forn(i, n) sa[i]=i, r[i]=s[i];
  for(int k=1; k<n; k<<=1){
    countingSort(k), countingSort(0);
    int rank, tmpr[MAX_N];
    tmpr[sa[0]]=rank=0;
    forr(i, 1, n)
      tmpr[sa[i]]=(r[sa[i]]==r[sa[i-1]] && r[sa[i]+k]==r[sa[i-1]+k] )? rank : ++rank;
    memcpy(r, tmpr, sizeof(r));
    if(r[sa[n-1]]==n-1) break;
  }
}
void print(){//for debug
  forn(i, n)
    cout << i << '␣' <<
    s.substr(sa[i], s.find( '$', sa[i])-sa[i]) << endl;}
```

4.6. String Matching With Suffix Array

```
//returns (lowerbound, upperbound) of the search
ii stringMatching(string P){ //O(sz(P)lgn)
  int lo=0, hi=n-1, mid=lo;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(sa[mid], sz(P), P);
    if(res>=0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(sa[lo], sz(P), P)!=0) return ii(-1, -1);
  ii ans; ans.fst=lo;
  lo=0, hi=n-1, mid;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(sa[mid], sz(P), P);
    if(res>0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(sa[hi], sz(P), P)!=0) hi--;
  ans.snd=hi;
  return ans;
}
```

4.7. LCP (Longest Common Prefix)

```
//Calculates the LCP between consecutives suffixes in the Suffix Array.
//LCP[i] is the length of the LCP between sa[i] and sa[i-1]
int LCP[MAX_N], phi[MAX_N], PLCP[MAX_N];
void computeLCP(){//O(n)
  phi[sa[0]]=-1;
  forr(i, 1, n) phi[sa[i]]=sa[i-1];
  int L=0;
  forn(i, n){
    if(phi[i]==-1) {PLCP[i]=0; continue;}
    while(s[i+L]==s[phi[i]+L]) L++;
    PLCP[i]=L;
    L=max(L-1, 0);
  }
  forn(i, n) LCP[i]=PLCP[sa[i]];
}
```

4.8. Corasick

```

struct trie{
  map<char, trie> next;
  trie* tran[256];//transiciones del automata
  int idhoja, szhoja;//id de la hoja o 0 si no lo es
  //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que es hoja
  trie *padre, *link, *nxthoja;
  char pch;//caracter que conecta con padre
  trie(): tran(),  idhoja(), padre(), link() {}
  void insert(const string &s, int id=1, int p=0){//id>0!!!
    if(p<sz(s)){
      trie &ch=next[s[p]];
      tran[(int)s[p]]=&ch;
      ch.padre=this, ch.pch=s[p];
      ch.insert(s, id, p+1);
    }
    else idhoja=id, szhoja=sz(s);
  }
  trie* get_link() {
    if(!link){
      if(!padre) link=this;//es la raiz
      else if(!padre->padre) link=padre;//hijo de la raiz
      else link=padre->get_link()->get_tran(pch);
    }
    return link; }
  trie* get_tran(int c) {
    if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
    return tran[c]; }
  trie *get_nxthoja(){
    if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
    return nxthoja; }
  void print(int p){
    if(idhoja) cout << "found␣" << idhoja << "␣␣at␣position␣" << p-szhoja << endl;
    if(get_nxthoja()) get_nxthoja()->print(p); }
  void matching(const string &s, int p=0){
    print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1); }
}tri;


int main(){
  tri=trie();//clear
  tri.insert("ho", 1);
  tri.insert("hoho", 2);
```

4.9. Suffix Automaton

```
struct state {
  int len, link;
  map<char,int> next;
  state() { }
};
const int MAXLEN = 10010;
state st[MAXLEN*2];
int sz, last;
void sa_init() {
  forn(i,sz) st[i].next.clear();
  sz = last = 0;
  st[0].len = 0;
  st[0].link = -1;
  ++sz;
}
// Es un DAG de una sola fuente y una sola hoja
// cantidad de endpos = cantidad de apariciones = cantidad de caminos de la clase al nodo terminal
// cantidad de miembros de la clase = st[v].len-st[st[v].link].len (v>0) = caminos del inicio a la clase
// El arbol de los suffix links es el suffix tree de la cadena invertida. La string de la arista link(v)->v son los caracteres que difieren
void sa_extend (char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  // en cur agregamos la posicion que estamos extendiendo
  //podria agregar tambien un identificador de las cadenas a las cuales pertenece (si hay varias)
  int p;
  for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link) // modificar esta linea para hacer separadores unicos entre varias cadenas (c
   ==’$’)
  st[p].next[c] = cur;
 if (p == -1)
  st[cur].link = 0;
 else {
  int q = st[p].next[c];
  if (st[p].len + 1 == st[q].len)
   st[cur].link = q;
  else {
   int clone = sz++;
   // no le ponemos la posicion actual a clone sino indirectamente por el link de cur
   st[clone].len = st[p].len + 1;
   st[clone].next = st[q].next;
   st[clone].link = st[q].link;
   for (; p!=-1 && st[p].next.count(c) && st[p].next[c]==q; p=st[p].link)
    st[p].next[c] = clone;
   st[q].link = st[cur].link = clone;
  }
 }
 last = cur;
}
```

4.10. Z Function

```
char s[MAXN];
int z[MAXN]; // z[i] = i==0 ? 0 : max k tq s[0,k) match with s[i,i+k)
void z_function(char s[],int z[]) {
  int n = strlen(s);
  forn(i, n) z[i]=0;
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min (r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])  ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
}

int main() {
  ios::sync_with_stdio(0);
```

5. Geometria

5.1. Punto

```
struct pto{
 double x, y;
 pto(double x=0, double y=0):x(x),y(y){}
 pto operator+(pto a){return pto(x+a.x, y+a.y);}
 pto operator-(pto a){return pto(x-a.x, y-a.y);}
 pto operator+(double a){return pto(x+a, y+a);}
 pto operator*(double a){return pto(x*a, y*a);}
 pto operator/(double a){return pto(x/a, y/a);}
 //dot product, producto interno:
 double operator*(pto a){return x*a.x+y*a.y;}
 //module of the cross product or vectorial product:
 //if a is less than 180 clockwise from b, a^b>0
 double operator^(pto a){return x*a.y-y*a.x;}
 //returns true if this is at the left side of line qr
 bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
 bool operator<(const pto &a) const{return x<a.x-EPS || (abs(x-a.x)<EPS && y<a.y-EPS);}
bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
 double norm(){return sqrt(x*x+y*y);}
 double norm_sq(){return x*x+y*y;}
};
double dist(pto a, pto b){return (b-a).norm();}
typedef pto vec;

double angle(pto a, pto o, pto b){
 pto oa=a-o, ob=b-o;
 return atan2(oa^ob, oa*ob);}

//rotate p by theta rads CCW w.r.t. origin (0,0)
pto rotate(pto p, double theta){
 return pto(p.x*cos(theta)-p.y*sin(theta),
   p.x*sin(theta)+p.y*cos(theta));
}
```

5.2. Orden radial de puntos

```
struct Cmp{//orden total de puntos alrededor de un punto r
 pto r;
 Cmp(pto r):r(r) {}
```

```
  int cuad(const pto &a) const{
    if(a.x > 0 && a.y >= 0)return 0;
    if(a.x <= 0 && a.y > 0)return 1;
    if(a.x < 0 && a.y <= 0)return 2;
    if(a.x >= 0 && a.y < 0)return 3;
    assert(a.x ==0 && a.y==0);
    return -1;
  }
  bool cmp(const pto&p1, const pto&p2)const{
    int c1 = cuad(p1), c2 = cuad(p2);
    if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
        else return c1 < c2;
  }
    bool operator()(const pto&p1, const pto&p2) const{
    return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
    }
};
```

5.3. Line

```
int sgn(ll x){return x<0? -1 : !!x;}
struct line{
  line() {}
  double a,b,c;//Ax+By=C
//pto MUST store float coordinates!
  line(double a, double b, double c):a(a),b(b),c(c){}
  line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
  int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
};
bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
pto inter(line l1, line l2){//intersection
  double det=l1.a*l2.b-l2.a*l1.b;
  if(abs(det)<EPS) return pto(INF, INF);//parallels
  return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
}
```

5.4. Segment

```
struct segm{
  pto s,f;
  segm(pto s, pto f):s(s), f(f) {}
  pto closest(pto p) {//use for dist to point
     double l2 = dist_sq(s, f);
     if(l2==0.) return s;
     double t =((p-s)*(f-s))/l2;
     if (t<0.) return s;//not write if is a line
     else if(t>1.)return f;//not write if is a line
     return s+((f-s)*t);
  }
    bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS;}
};

//NOTA: Si los segmentos son coolineales solo devuelve un punto de interseccion
pto inter(segm s1, segm s2){
    if(s1.inside(s2.s)) return s2.s; //Fix cuando son colineales
    if(s1.inside(s2.f)) return s2.f; //Fix cuando son colineales
  pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
    if(s1.inside(r) && s2.inside(r)) return r;
  return pto(INF, INF);
}
```

5.5. Rectangle

```
struct rect{
  //lower-left and upper-right corners
  pto lw, up;
};
//returns if there's an intersection and stores it in r
bool inter(rect a, rect b, rect &r){
  r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
  r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
//check case when only a edge is common
  return r.lw.x<r.up.x && r.lw.y<r.up.y;
}
```

5.6. Polygon Area

```
double area(vector<pto> &p){//O(sz(p))
  double area=0;
  forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
  //if points are in clockwise order then area is negative
  return abs(area)/2;
}
//Area ellipse = M_PI*a*b where a and b are the semi axis lengths
//Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

5.7. Circle

```
vec perp(vec v){return vec(-v.y, v.x);}
line bisector(pto x, pto y){
  line l=line(x, y); pto m=(x+y)/2;
  return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
}
struct Circle{
  pto o;
  double r;
  Circle(pto x, pto y, pto z){
    o=inter(bisector(x, y), bisector(y, z));
    r=dist(o, x);
  }
  pair<pto, pto> ptosTang(pto p){
    pto m=(p+o)/2;
    tipo d=dist(o, m);
    tipo a=r*r/(2*d);
    tipo h=sqrt(r*r-a*a);
    pto m2=o+(m-o)*a/d;
    vec per=perp(m-o)/d;
    return make_pair(m2-per*h, m2+per*h);
  }
};
//finds the center of the circle containing p1 and p2 with radius r
//as there may be two solutions swap p1, p2 to get the other
bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
        double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
        if(det<0) return false;
        c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
        return true;
}
#define sqr(a) ((a)*(a))
#define feq(a,b) (fabs((a)-(b))<EPS)
pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
  tipo dx = sqrt(b*b-4.0*a*c);
  return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
}
pair<pto, pto> interCL(Circle c, line l){
  bool sw=false;
  if((sw=feq(0,l.b))){
  swap(l.a, l.b);
  swap(c.o.x, c.o.y);
  }
  pair<tipo, tipo> rc = ecCuad(
  sqr(l.a)+sqr(l.b),
  2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
  sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.o.y
  );
  pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
            pto(rc.second, (l.c - l.a * rc.second) / l.b) );
  if(sw){
  swap(p.first.x, p.first.y);
  swap(p.second.x, p.second.y);
  }
  return p;
}
pair<pto, pto> interCC(Circle c1, Circle c2){
  line l;
  l.a = c1.o.x-c2.o.x;
  l.b = c1.o.y-c2.o.y;
  l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
  -sqr(c2.o.y))/2.0;
  return interCL(c1, l);
}
```

5.8. Point in Poly

```
//checks if v is inside of P, using ray casting
//works with convex and concave.
//excludes boundaries, handle it separately using segment.inside()
bool inPolygon(pto v, vector<pto>& P) {
  bool c = false;
  forn(i, sz(P)){
    int j=(i+1)%sz(P);
    if((P[j].y>v.y) != (P[i].y > v.y) &&
  (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
      c = !c;
  }
  return c;
}
```

5.9. Point in Convex Poly log(n)

```
void normalize(vector<pto> &pt){//delete collinear points first!
  //this makes it clockwise:
    if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
  int n=sz(pt), pi=0;
  forn(i, n)
    if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
      pi=i;
  vector<pto> shift(n);//puts pi as first point
    forn(i, n) shift[i]=pt[(pi+i)%n];
    pt.swap(shift);
}
bool inPolygon(pto p, const vector<pto> &pt){
  //call normalize first!
  if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
  int a=1, b=sz(pt)-1;
  while(b-a>1){
    int c=(a+b)/2;
    if(!p.left(pt[0], pt[c])) a=c;
    else b=c;
  }
  return !p.left(pt[a], pt[a+1]);
}
```

5.10. Convex Check CHECK

```
bool isConvex(vector<int> &p){//O(N), delete collinear points!
  int N=sz(p);
  if(N<3) return false;
  bool isLeft=p[0].left(p[1], p[2]);
  forr(i, 1, N)
    if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
      return false;
  return true; }
```

5.11. Convex Hull

```
//stores convex hull of P in S, CCW order
//left must return >=0 to delete collinear points!
void CH(vector<pto>& P, vector<pto> &S){
  S.clear();
  sort(P.begin(), P.end());//first x, then y
  forn(i, sz(P)){//lower hull
    while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
    S.pb(P[i]);
  }
  S.pop_back();
  int k=sz(S);
  dforn(i, sz(P)){//upper hull
    while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
    S.pb(P[i]);
  }
  S.pop_back();
}
```

5.12. Cut Polygon

```
//cuts polygon Q along the line ab
//stores the left side (swap a, b for the right one) in P
void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
  P.clear();
  forn(i, sz(Q)){
    double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
    if(left1>=0) P.pb(Q[i]);
    if(left1*left2<0)
      P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
  }
}
```

5.13. Bresenham

```
//plot a line approximation in a 2d map
void bresenham(pto a, pto b){
  pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
  pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
  int err=d.x-d.y;
  while(1){
    m[a.x][a.y]=1;//plot
    if(a==b) break;
    int e2=err;
    if(e2 >= 0) err-=2*d.y, a.x+=s.x;
    if(e2 <= 0) err+= 2*d.x, a.y+= s.y;
  }
}
```

5.14. Rotate Matrix

```
//rotates matrix t 90 degrees clockwise
//using auxiliary matrix t2(faster)
void rotate(){
  forn(x, n) forn(y, n)
```

```
    t2[n-y-1][x]=t[x][y];
  memcpy(t, t2, sizeof(t));
}
```

5.15. Interseccion de Circulos en n3log(n)

```
struct event {
    double x; int t;
    event(double xx, int tt) : x(xx), t(tt) {}
    bool operator <(const event &o) const { return x < o.x; }
};
typedef vector<Circle> VC;
typedef vector<event> VE;
int n;
double cuenta(VE &v, double A,double B) {
    sort(v.begin(), v.end());
    double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
    int contador = 0;
    forn(i,sz(v)) {
        //interseccion de todos (contador == n), union de todos (contador > 0)
        //conjunto de puntos cubierto por exacta k Circulos (contador == k)
        if (contador == n) res += v[i].x - lx;
        contador += v[i].t, lx = v[i].x;
    }
    return res;
}
// Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
inline double primitiva(double x,double r) {
    if (x >= r) return r*r*M_PI/4.0;
    if (x <= -r) return -r*r*M_PI/4.0;
    double raiz = sqrt(r*r-x*x);
    return 0.5 * (x * raiz + r*r*atan(x/raiz));
}
double interCircle(VC &v) {
    vector<double> p; p.reserve(v.size() * (v.size() + 2));
    forn(i,sz(v))  p.push_back(v[i].c.x + v[i].r), p.push_back(v[i].c.x - v[i].r);
    forn(i,sz(v)) forn(j,i) {
        Circle &a = v[i], b = v[j];
        double d = (a.c - b.c).norm();
        if (fabs(a.r - b.r) < d && d < a.r + b.r) {
            double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r));
            pto vec = (b.c - a.c) * (a.r / d);
            p.pb((a.c + rotate(vec, alfa)).x), p.pb((a.c + rotate(vec, -alfa)).x);
        }
    }
    sort(p.begin(), p.end());
    double res = 0.0;
    forn(i,sz(p)-1) {
        const double A = p[i], B = p[i+1];
        VE ve; ve.reserve(2 * v.size());
        forn(j,sz(v)) {
            const Circle &c = v[j];
            double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
            double base = c.c.y * (B-A);
            ve.push_back(event(base + arco,-1));
            ve.push_back(event(base - arco, 1));
        }
        res += cuenta(ve,A,B);
    }
    return res;
}
```

6. Math

6.1. Identidades

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

$$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$$

$$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

$$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par}$$

$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^{n} i\right]^2$$

$$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$

$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$$

$$r = e - v + k + 1$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde)

$$A = I + \frac{B}{2} - 1$$

6.2. Ec. Caracteristica

$$a_0T(n)+a_1T(n-1)+...+a_kT(n-k)=0$$

$$p(x)=a_0x^k+a_1x^{k-1}+...+a_k$$

Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$

$$T(n)=\sum_{i=1}^{q}\sum_{j=0}^{m_i-1} c_{ij}n^jr_i^n$$

Las constantes c_{ij} se determinan por los casos base.

6.3. Combinatorio

```
forn(i, MAXN+1){//comb[i][k]=i tomados de a k
  comb[i][0]=comb[i][i]=1;
  forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
}
ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p teniendo comb[p][p] precalculado.
  ll aux = 1;
  while (n + k) aux = (aux * comb[n%p][k%p]) %p, n/=p, k/=p;
  return aux;
}
```

6.4. Exp. de Numeros Mod.

```
ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=(q*q)%m;
  return e%2? (b * q)%m : q;
}
```

6.5. Exp. de Matrices y Fibonacci en log(n)

```
#define SIZE 350
int NN;
double tmp[SIZE][SIZE];
void mul(double a[SIZE][SIZE], double b[SIZE][SIZE]){ zero(tmp);
    forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
    forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
}
void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE]){
    forn(i, NN) forn(j, NN) res[i][j]=(i==j);
    while(n){
        if(n&1) mul(res, a), n--;
        else mul(a, a), n/=2;
    } }
```

6.6. Matrices y determinante $O(n^3)$

```
struct Mat {
    vector<vector<double> > vec;
    Mat(int n): vec(n, vector<double>(n) ) {}
    Mat(int n, int m): vec(n, vector<double>(m) ) {}
    vector<double> &operator[](int f){return vec[f];}
    const vector<double> &operator[](int f) const {return vec[f];}
    int size() const {return sz(vec);}
    Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
        Mat m(sz(b),sz(b[0]));
        forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j];
        return m;    }
    Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
        int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
        Mat mat(n,t);
        forn(i,n) forn(j,t) forn(k,m) mat[i][j] += vec[i][k] * b[k][j];
        return mat;    }
    double determinant(){//sacado de e maxx ru
        double det = 1;
        int n = sz(vec);
        Mat m(*this);
        forn(i, n){//para cada columna
            int k = i;
            forr(j, i+1, n)//busco la fila con mayor val abs
                if(abs(m[j][i])>abs(m[k][i])) k = j;
            if(abs(m[k][i])<1e-9) return 0;
            m[i].swap(m[k]);//la swapeo
            if(i!=k) det = -det;
            det *= m[i][i];
            forr(j, i+1, n) m[i][j] /= m[i][i];
            //hago 0 todas las otras filas
            forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
                forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
        }
        return det;
    }
};

int n;
int main() {
//DETERMINANTE:
```

```
//https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=625
  freopen("input.in", "r", stdin);
    ios::sync_with_stdio(0);
    while(cin >> n && n){
        Mat m(n);
        forn(i, n) forn(j, n) cin >> m[i][j];
        cout << (ll)round(m.determinant()) << endl;
    }
    cout  << "*" << endl;
  return 0;
}
```

6.7. Teorema Chino del Resto

$$y = \sum_{j=1}^{n}(x_j * (\prod_{i=1,i\neq j}^{n} m_i)^{-1}_{m_j} * \prod_{i=1,i\neq j}^{n} m_i)$$

6.8. Criba

```
#define MAXP 100000 //no necesariamente primo
int criba[MAXP+1];
void crearcriba(){
  int w[] = {4,2,4,2,4,6,2,6};
  for(int p=25;p<=MAXP;p+=10) criba[p]=5;
  for(int p=9;p<=MAXP;p+=6) criba[p]=3;
  for(int p=4;p<=MAXP;p+=2) criba[p]=2;
  for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
    for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
}
vector<int> primos;
void buscarprimos(){
  crearcriba();
  forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
}
//~ Useful for bit trick: #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) ), #define INDEX(i) ( (criba[i>>5]>>((i)&31))&1 ), unsigned int criba[MAXP/32+1];


int main() {
  freopen("primos", "w", stdout);
  buscarprimos();
```

6.9. Funciones de primos

Sea $n = \prod p_i^{k_i}$, fact(n) genera un map donde a cada p_i le asocia su k_i

```
//factoriza bien numeros hasta MAXP^2
map<ll,ll> fact(ll n){ //O (cant primos)
  map<ll,ll> ret;
  forall(p, primos){
    while(!(n%*p)){
      ret[*p]++;//divisor found
      n/=*p;
    }
  }
  if(n>1) ret[n]++;
  return ret;
}
//factoriza bien numeros hasta MAXP
map<ll,ll> fact2(ll n){ //O (lg n)
  map<ll,ll> ret;
  while (criba[n]){
    ret[criba[n]]++;
    n/=criba[n];
  }
  if(n>1) ret[n]++;
  return ret;
}
//Usar asi:  divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
void divisores(const map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::iterator it, ll n=1){
    if(it==f.begin()) divs.clear();
    if(it==f.end()) { divs.pb(n);  return; }
    ll p=it->fst, k=it->snd; ++it;
    forn(_, k+1) divisores(f, divs, it, n), n*=p;
}
ll sumDiv (ll n){
  ll rta = 1;
  map<ll,ll> f=fact(n);
  forall(it, f) {
  ll pot = 1, aux = 0;
  forn(i, it->snd+1) aux += pot, pot *= it->fst;
  rta*=aux;
  }
  return rta;
}
ll eulerPhi (ll n){ // con criba: O(lg n)
  ll rta = n;
  map<ll,ll> f=fact(n);
  forall(it, f) rta -= rta / it->first;
  return rta;
}
ll eulerPhi2 (ll n){ // O (sqrt n)
  ll r = n;
  forr (i,2,n+1){
    if ((ll)i*i > n) break;
    if (n % i == 0){
      while (n%i == 0) n/=i;
      r -= r/i; }
  }
  if (n != 1) r-= r/n;
  return r;
}

int main() {
  buscarprimos();
  forr (x,1, 500000){
    cout << "x␣=␣" << x << endl;
    cout << "Numero␣de␣factores␣primos:␣" << numPrimeFactors(x) << endl;
    cout << "Numero␣de␣distintos␣factores␣primos:␣" <<
        numDiffPrimeFactors(x) << endl;
    cout << "Suma␣de␣factores␣primos:␣" << sumPrimeFactors(x) << endl;
    cout << "Numero␣de␣divisores:␣" << numDiv(x) << endl;
    cout << "Suma␣de␣divisores:␣" << sumDiv(x) << endl;
    cout << "Phi␣de␣Euler:␣" << eulerPhi(x) << endl;
  }
  return 0;
}
```

6.10. Phollard's Rho (rolando)

```
ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}

ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c, and minimize overfloor
  ll x = 0, y = a%c;
  while (b > 0){
    if (b % 2 == 1) x = (x+y) % c;
    y = (y*2) % c;
    b /= 2;
  }
  return x % c;
}

ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
  return e%2? mulmod(b,q,m) : q;
}

bool es_primo_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;

  forn (i, s-1){
    x = mulmod(x, x, n);
    if (x == 1) return false;
    if (x+1 == n) return true;
  }
  return false;
}

bool rabin (ll n){ //devuelve true si n es primo
  if (n == 1) return false;
  const int ar[] = {2,3,5,7,11,13,17,19,23};
  forn (j,9)
    if (!es_primo_prob(n,ar[j]))
      return false;
  return true;
}

ll rho(ll n){
    if( (n & 1) == 0 ) return 2;
    ll x = 2 , y = 2 , d = 1;
    ll c = rand() % n + 1;
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = gcd( x - y , n );
        else d = gcd( y - x , n );
    }
    return d==n? rho(n):d;
}

map<ll,ll> prim;
void factRho (ll n){ //O (lg n)^3. un solo numero
  if (n == 1) return;
  if (rabin(n)){
    prim[n]++;
    return;
  }
  ll factor = rho(n);
  factRho(factor);
  factRho(n/factor);
}
```

6.11. GCD

```
tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}
```

6.12. Extended Euclid

```
void extendedEuclid (ll a, ll b){ //a * x + b * y = d
  if (!b) { x = 1; y = 0; d = a; return;}
  extendedEuclid (b, a%b);
  ll x1 = y;
  ll y1 = x - (a/b) * y;
  x = x1; y = y1;
}
```

6.13. LCM

```
tipo lcm(tipo a, tipo b){return a / gcd(a,b) * b;}
```

6.14. Inversos

```
#define MAXMOD 15485867
ll inv[MAXMOD];//inv[i]*i=1 mod MOD
void calc(int p){//O(p)
  inv[1]=1;
  forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
}
int inverso(int x){//O(log x)
  return expmod(x, eulerphi(MOD)-2);//si mod no es primo(sacar a mano)
  return expmod(x, MOD-2);//si mod es primo
}
```

6.15. Simpson

```
double integral(double a, double b, int n=10000) {//O(n), n=cantdiv
  double area=0, h=(b-a)/n, fa=f(a), fb;
  forn(i, n){
    fb=f(a+h*(i+1));
    area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
  }
  return area*h/6.;}
```

6.16. Fraction

```
tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
struct frac{
  tipo p,q;
  frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
  void norm(){
    tipo a = mcd(p,q);
    if(a) p/=a, q/=a;
    else q=1;
    if (q<0) q=-q, p=-p;}
  frac operator+(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
  frac operator-(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
  frac operator*(frac o){
    tipo a = mcd(q,o.p), b = mcd(o.q,p);
    return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
  frac operator/(frac o){
    tipo a = mcd(q,o.q), b = mcd(o.p,p);
    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
  bool operator<(const frac &o) const{return p*o.q < o.p*q;}
```

```
 bool operator==(frac o){return p==o.p&&q==o.q;}
};
```

6.17. Polinomio

```
    int m = sz(c), n = sz(o.c);
    vector<tipo> res(max(m,n));
    forn(i, m) res[i] += c[i];
    forn(i, n) res[i] += o.c[i];
    return poly(res);    }
  poly operator*(const tipo cons) const {
  vector<tipo> res(sz(c));
    forn(i, sz(c)) res[i]=c[i]*cons;
    return poly(res);    }
  poly operator*(const poly &o) const {
    int m = sz(c), n = sz(o.c);
    vector<tipo> res(m+n-1);
    forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[j];
    return poly(res);    }
 tipo eval(tipo v) {
  tipo sum = 0;
  dforn(i, sz(c)) sum=sum*v + c[i];
  return sum; }
  //poly contains only a vector<int> c (the coeficients)
 //the following function generates the roots of the polynomial
//it can be easily modified to return float roots
 set<tipo> roots(){
  set<tipo> roots;
  tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
  vector<tipo> ps,qs;
  forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
  forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
  forall(pt,ps)
   forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
    tipo root = abs((*pt) / (*qt));
    if (eval(root)==0) roots.insert(root);
   }
  return roots; }
};
pair<poly,tipo> ruffini(const poly p, tipo r) {
 int n = sz(p.c) - 1 ;
 vector<tipo> b(n);
 b[n-1] = p.c[n];
 dforn(k,n-1) b[k] = p.c[k+1] + r*b[k+1];
 tipo resto = p.c[0] + r*b[0];
 poly result(b);
 return make_pair(result,resto);
}
poly interpolate(const vector<tipo>& x,const vector<tipo>& y) {
  poly A; A.c.pb(1);
  forn(i,sz(x)) { poly aux; aux.c.pb(-x[i]), aux.c.pb(1), A = A * aux; }
 poly S; S.c.pb(0);
 forn(i,sz(x)) { poly Li;
  Li = ruffini(A,x[i]).fst;
  Li = Li * (1.0 / Li.eval(x[i])); // here put a multiple of the coefficients instead of 1.0 to avoid using double
  S = S + Li * y[i];  }
 return S;
}

int  main(){
 return 0;
}
```

6.18. Ec. Lineales

```
bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
 int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
 vector<int> p; forn(i,m) p.push_back(i);
 forn(i, rw) {
  int uc=i, uf=i;
  forr(f, i, n) forr(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;uc=c;}
  if (feq(a[uf][uc], 0)) { rw = i; break; }
  forn(j, n) swap(a[j][i], a[j][uc]);
  swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
  tipo inv = 1 / a[i][i]; //aca divide
  forr(j, i+1, n) {
   tipo v = a[j][i] * inv;
   forr(k, i, m) a[j][k]-=v * a[i][k];
   y[j] -= v*y[i];
  }
 } // rw = rango(a), aca la matriz esta triangulada
 forr(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de compatibilidad
 x = vector<tipo>(m, 0);
 dforn(i, rw){
  tipo s = y[i];
  forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
  x[p[i]] = s / a[i][i]; //aca divide
 }
 ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
 forn(k, m-rw) {
  ev[k][p[k+rw]] = 1;
  dforn(i, rw){
   tipo s = -a[i][k+rw];
   forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
   ev[k][p[i]] = s / a[i][i]; //aca divide
  }
 }
 return true;
}
```

6.19. FFT

```
//~ typedef complex<double> base; //menos codigo, pero mas lento
//elegir si usar complejos de c (lento) o estos
struct base{
  double r,i;
  base(double r=0, double i=0):r(r), i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
base operator*(const base &a, const base &b){
  return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
base operator+(const base &a, const base &b){
  return base(a.r+b.r, a.i+b.i);}
base operator-(const base &a, const base &b){
  return base(a.r-b.r, a.i-b.i);}
vector<int> rev; vector<base> wlen_pw;
inline static void fft(base a[], int n, bool invert) {
  forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
 for (int len=2; len<=n; len<<=1) {
  double ang = 2*M_PI/len * (invert?-1:+1);
  int len2 = len>>1;
  base wlen (cos(ang), sin(ang));
  wlen_pw[0] = base (1, 0);
    forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
  for (int i=0; i<n; i+=len) {
   base t, *pu = a+i, *pv = a+i+len2,  *pu_end = a+i+len2, *pw = &wlen_pw[0];
   for (; pu!=pu_end; ++pu, ++pv, ++pw)
    t = *pv * *pw, *pv = *pu - t,*pu = *pu + t;
  }
 }
 if (invert) forn(i, n) a[i]/= n;}
inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
  wlen_pw.resize(n), rev.resize(n);
  int lg=31-__builtin_clz(n);
  forn(i, n){
  rev[i] = 0;
    forn(k, lg) if(i&(1<<k)) rev[i]|=1<<(lg-1-k);
  }}
inline static void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res) {
 vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
  int n=1; while(n < max(sz(a), sz(b))) n <<= 1; n <<= 1;
  calc_rev(n);
 fa.resize (n),  fb.resize (n);
 fft (&fa[0], n, false),  fft (&fb[0], n, false);
 forn(i, n) fa[i] = fa[i] * fb[i];
 fft (&fa[0], n, true);
 res.resize(n);
  forn(i, n) res[i] = int (fa[i].real() + 0.5); }
void toPoly(const string &s, vector<int> &P){//convierte un numero a polinomio
  P.clear();
  dforn(i, sz(s)) P.pb(s[i]-'0');}
```

6.20. Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

$0! = 1$	$11! = 39.916.800$
$1! = 1$	$12! = 479.001.600$ ($\in$ `int`)
$2! = 2$	$13! = 6.227.020.800$
$3! = 6$	$14! = 87.178.291.200$
$4! = 24$	$15! = 1.307.674.368.000$
$5! = 120$	$16! = 20.922.789.888.000$
$6! = 720$	$17! = 355.687.428.096.000$
$7! = 5.040$	$18! = 6.402.373.705.728.000$
$8! = 40.320$	$19! = 121.645.100.408.832.000$
$9! = 362.880$	$20! = 2.432.902.008.176.640.000$ ($\in$ `tint`)
$10! = 3.628.800$	$21! = 51.090.942.171.709.400.000$

max signed tint = 9.223.372.036.854.775.807

max unsigned tint = 18.446.744.073.709.551.615

Primos

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 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052,511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

Divisores

Cantidad de divisores (σ_0) para *algunos* $n/\neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$

$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$

Suma de divisores (σ_1) para *algunos* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$

$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

7. Grafos

7.1. Dijkstra

```
// AC - https://codeforces.com/contest/20/submission/47280560
vector<pll> G[MAXN];  // u->[(v,cost)]
ll dist[MAXN], dad[MAXN];
void dijkstra(ll x){
  memset(dist,-1,sizeof(dist));
  memset(dad,-1,sizeof(dad));
  priority_queue<pll> Q;
  dist[x]=0; Q.push({0,x});
  while(!Q.empty()){
    x=Q.top().snd; ll c=-Q.top().fst; Q.pop();
    if(dist[x]!=c)continue;
    forn(i,G[x].size()){
      ll y=G[x][i].fst, c=G[x][i].snd;
      if(dist[y]<0 || dist[x]+c<dist[y])
    dist[y]=dist[x]+c, Q.push({-dist[y],y}), dad[y]=x;
  }
 }
}
```

7.2. Bellman-Ford

```
vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
int dist[MAX_N];
void bford(int src){//O(VE)
 dist[src]=0;
 forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
  dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
}

bool hasNegCycle(){
 forn(j, N) if(dist[j]!=INF) forall(it, G[j])
  if(dist[it->snd]>dist[j]+it->fst) return true;
 //inside if: all points reachable from it->snd will have -INF distance (do bfs)
 return false;
}
```

7.3. Floyd-Warshall

```
//G[i][j] contains weight of edge (i, j) or INF
//G[i][i]=0
int G[MAX_N][MAX_N];
void floyd(){//O(N^3)
forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
 G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
}
bool inNegCycle(int v){
 return G[v][v]<0;}
//checks if there's a neg. cycle in path from a to b
bool hasNegCycle(int a, int b){
 forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
  return true;
 return false;
}
```

7.4. Spanning Tree (Kruskal)

```
// Requires disjointSetjoin.cpp
struct Edge{
  int u, v;
  ll weight;
  bool operator < (Edge const& other) const{
    return weight < other.weight;
  }
};

// Constructs MST from vector<Edge> e and stores it in result.
// Returns the sum of the edges used.
ll kruskal(int nodeAmount, vector<Edge> &ed, vector<vector<int>> &result){
  ll cost = 0;
  DSUFast dsu(nodeAmount);
  sort(ed.begin(), ed.end());
  result.assign(nodeAmount, vector<int>());

  for(Edge e : ed)
    if(dsu.find(e.u) != dsu.find(e.v)){
      cost += e.weight;
      result[e.u].pb(e.v);
      result[e.v].pb(e.u);
      dsu.join(e.u, e.v);
    }

  return cost;
}
```

7.5. Prim

```
bool taken[MAXN];
priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
void process(int v){
  taken[v]=true;
  forall(e, G[v])
    if(!taken[e->second]) pq.push(*e);
}

ll prim(){
  zero(taken);
  process(0);
  ll cost=0;
  while(sz(pq)){
    ii e=pq.top(); pq.pop();
    if(!taken[e.second]) cost+=e.first, process(e.second);
  }
  return cost;
}
```

7.6. 2-SAT + Tarjan SCC

```
//We have a vertex representing a var and other for his negation.
//Every edge stored in G represents an implication. To add an equation of the form a||b, use addor(a, b)
//MAX=max cant var, n=cant var
#define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
vector<int> G[MAX*2];
//idx[i]=index assigned in the dfs
//lw[i]=lowest index(closer from the root) reachable from i
int lw[MAX*2], idx[MAX*2], qidx;
stack<int> q;
int qcmp, cmp[MAX*2];
//verdad[cmp[i]]=valor de la variable i
bool verdad[MAX*2+1];

int neg(int x) { return x>=n? x-n : x+n;}
void tjn(int v){
 lw[v]=idx[v]=++qidx;
 q.push(v), cmp[v]=-2;
 forall(it, G[v]){
  if(!idx[*it] || cmp[*it]==-2){
   if(!idx[*it]) tjn(*it);
   lw[v]=min(lw[v], lw[*it]);
  }
 }
 if(lw[v]==idx[v]){
  int x;
  do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
  verdad[qcmp]=(cmp[neg(v)]<0);
  qcmp++;
 }
}
//remember to CLEAR G!!!
bool satisf(){//O(n)
 memset(idx, 0, sizeof(idx)), qidx=0;
 memset(cmp, -1, sizeof(cmp)), qcmp=0;
 forn(i, n){
  if(!idx[i]) tjn(i);
  if(!idx[neg(i)]) tjn(neg(i));
 }
 forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
 return true;
}
```

7.7. Articulation Points

```
int N;
vector<int> G[1000000];
//V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
int qV, V[1000000], L[1000000], P[1000000];
void dfs(int v, int f){
 L[v]=V[v]=++qV;
 forall(it, G[v])
  if(!V[*it]){
   dfs(*it, v);
   L[v] = min(L[v], L[*it]);
   P[v]+= L[*it]>=V[v];
  }
  else if(*it!=f)
   L[v]=min(L[v], V[*it]);
}
int cantart(){ //O(n)
 qV=0;
 zero(V), zero(P);
 dfs(1, 0); P[1]--;
 int q=0;
 forn(i, N) if(P[i]) q++;
return q;
}
```

7.8. Comp. Biconexas y Puentes

```
struct edge {
 int u,v, comp;
 bool bridge;
};
vector<edge> e;
void addEdge(int u, int v) {
 G[u].pb(sz(e)), G[v].pb(sz(e));
 e.pb((edge){u,v,-1,false});
}
//d[i]=id de la dfs
//b[i]=lowest id reachable from i
int d[MAXN], b[MAXN], t;
int nbc;//cant componentes
int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
void initDfs(int n) {
  zero(G), zero(comp);
  e.clear();
  forn(i,n) d[i]=-1;
  nbc = t = 0;
}
stack<int> st;
void dfs(int u, int pe) {//O(n + m)
  b[u] = d[u] = t++;
  comp[u] = (pe != -1);
  forall(ne, G[u]) if (*ne != pe){
    int v = e[*ne].u ^ e[*ne].v ^ u;
    if (d[v] == -1) {
      st.push(*ne);
      dfs(v,*ne);
      if (b[v] > d[u]){
        e[*ne].bridge = true; // bridge
      }
      if (b[v] >= d[u]){ // art
        int last;
        do {
          last = st.top(); st.pop();
          e[last].comp = nbc;
        } while (last != *ne);
        nbc++;
        comp[u]++;
      }
      b[u] = min(b[u], b[v]);
    }
    else if (d[v] < d[u]) { // back edge
      st.push(*ne);
      b[u] = min(b[u], d[v]);
    }
  }
}
```

7.9. LCA (binary lifting)

```
// Lowest Common Ancestor with Binary Lifting

// Given a global graph vector<int> G[N] with N nodes;
class LCA{
  public:
    int timer;
    int logN;
    vector<int> tIn, tOut;
    // up[v][k] stores the 2**k ancestor of v
    vector<vector<int>> up;

    LCA(int root = 0){
      timer = 0;
      tIn.resize(N);
      tOut.resize(N);
      logN = ceil(log2(N));
      up.assign(N, vector<int>(logN + 1));
      dfs(root, root);
    }

    void dfs(int v, int p){
      tIn[v] = timer++;
      up[v][0] = p;
      forn(i, logN)
        up[v][i+1] = up[ up[v][i] ][i];

      for(int w : G[v])
        if(w != p)
          dfs(w, v);

      tOut[v] = timer++;
    }

    // Is u an ancestor of v?
    bool isAncestor(int u, int v){
      return tIn[u] <= tIn[v] and tOut[u] >= tOut[v];
    }

    int lca(int u, int v){
      if(isAncestor(u, v))
        return u;
      if(isAncestor(v, u))
        return v;

      dforn(i, logN+1)
        if(not isAncestor(up[u][i], v))
          u = up[u][i];

      return up[u][0];
    }
};
```

7.10. Heavy Light Decomposition

```
int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
int dad[MAXN];//dad[v]=padre del nodo v
void dfs1(int v, int p=-1){//pre-dfs
  dad[v]=p;
  treesz[v]=1;
  forall(it, G[v]) if(*it!=p){
    dfs1(*it, v);
    treesz[v]+=treesz[*it];
  }
}
//PONER Q EN 0  !!!!!
int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
//Las cadenas aparecen continuas en el recorrido!
int cantcad;
int homecad[MAXN];//dada una cadena devuelve su nodo inicial
int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
void heavylight(int v, int cur=-1){
  if(cur==-1) homecad[cur=cantcad++]=v;
  pos[v]=q++;
  cad[v]=cur;
  int mx=-1;
  forn(i, sz(G[v])) if(G[v][i]!=dad[v])
    if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
  if(mx!=-1) heavylight(G[v][mx], cur);
  forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
    heavylight(G[v][i], -1);
}
//ejemplo de obtener el maximo numero en el camino entre dos nodos
//RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
//esta funcion va trepando por las cadenas
int query(int an, int v){//O(logn)
  //si estan en la misma cadena:
  if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
  return max(query(an, dad[homecad[cad[v]]]),
         rmq.get(pos[homecad[cad[v]]], pos[v]+1));
}
```

7.11. Centroid Decomposition

```
int n;
vector<int> G[MAXN];
bool taken[MAXN];//poner todos en FALSE al principio!!
int padre[MAXN];//padre de cada nodo en el centroid tree

int szt[MAXN];
void calcsz(int v, int p) {
  szt[v] = 1;
  forall(it,G[v]) if (*it!=p && !taken[*it])
    calcsz(*it,v), szt[v]+=szt[*it];
}
void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
  if(tam==-1) calcsz(v, -1), tam=szt[v];
  forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
    {szt[v]=0; centroid(*it, f, lvl, tam); return;}
  taken[v]=true;
  padre[v]=f;
  forall(it, G[v]) if(!taken[*it])
    centroid(*it, v, lvl+1, -1);
}
```

7.12. Euler Cycle

```
int n,m,ars[MAXE], eq;
vector<int> G[MAXN];//fill G,n,m,ars,eq
list<int> path;
int used[MAXN];
bool usede[MAXE];
queue<list<int>::iterator> q;
int get(int v){
  while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
  return used[v];
}
void explore(int v, int r, list<int>::iterator it){
  int ar=G[v][get(v)]; int u=v^ars[ar];
  usede[ar]=true;
  list<int>::iterator it2=path.insert(it, u);
  if(u!=r) explore(u, r, it2);
  if(get(v)<sz(G[v])) q.push(it);
}
void euler(){
  zero(used), zero(usede);
  path.clear();
  q=queue<list<int>::iterator>();
  path.push_back(0); q.push(path.begin());
  while(sz(q)){
    list<int>::iterator it=q.front(); q.pop();
    if(used[*it]<sz(G[*it])) explore(*it, *it, it);
  }
  reverse(path.begin(), path.end());
}
void addEdge(int u, int v){
  G[u].pb(eq), G[v].pb(eq);
  ars[eq++]=u^v;
}
```

7.13. Diametro árbol

```
  fill(df.begin(), df.end(), -1);
  fill(ds.begin(), ds.end(), -1);
  diams.clear(), centers.clear();
  int v, c;
  forn(i, N) if(df[i] == -1){
    c = v = bfs(bfs(i, ds), df);
    forn(_, df[v]/2) c = p[c];
    diams.pb(df[v]);
    centers.pb({c, df[v]&1 ? p[c] : c});
  }
}
```

7.14. Chu-liu

```
void visit(graph &h, int v, int s, int r,
  vector<int> &no, vector< vector<int> > &comp,
  vector<int> &prev, vector< vector<int> > &next, vector<weight> &mcost,
  vector<int> &mark, weight &cost, bool &found) {
  if (mark[v]) {
    vector<int> temp = no;
    found = true;
    do {
      cost += mcost[v];
      v = prev[v];
      if (v != s) {
        while (comp[v].size() > 0) {
          no[comp[v].back()] = s;
          comp[s].push_back(comp[v].back());
          comp[v].pop_back();
        }
      }
    } while (v != s);
    forall(j,comp[s]) if (*j != r) forall(e,h[*j])
      if (no[e->src] != s) e->w -= mcost[ temp[*j] ];
  }
  mark[v] = true;
  forall(i,next[v]) if (no[*i] != no[v] && prev[no[*i]] == v)
    if (!mark[no[*i]] || *i == s)
      visit(h, *i, s, r, no, comp, prev, next, mcost, mark, cost, found);
}
weight minimumSpanningArborescence(const graph &g, int r) {
    const int n=sz(g);
  graph h(n);
  forn(u,n) forall(e,g[u]) h[e->dst].pb(*e);
  vector<int> no(n);
  vector<vector<int> > comp(n);
  forn(u, n) comp[u].pb(no[u] = u);
  for (weight cost = 0; ;) {
    vector<int> prev(n, -1);
    vector<weight> mcost(n, INF);
    forn(j,n) if (j != r) forall(e,h[j])
      if (no[e->src] != no[j])
        if (e->w < mcost[ no[j] ])
          mcost[ no[j] ] = e->w, prev[ no[j] ] = no[e->src];
    vector< vector<int> > next(n);
    forn(u,n) if (prev[u] >= 0)
      next[ prev[u] ].push_back(u);
    bool stop = true;
    vector<int> mark(n);
    forn(u,n) if (u != r && !mark[u] && !comp[u].empty()) {
      bool found = false;
      visit(h, u, u, r, no, comp, prev, next, mcost, mark, cost, found);
      if (found) stop = false;
    }
    if (stop) {
      forn(u,n) if (prev[u] >= 0) cost += mcost[u];
      return cost;
    }
  }
}
```

7.15. Hungarian

```
//Dado un grafo bipartito completo con costos no negativos, encuentra el matching perfecto de minimo costo.
tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar: cost=matriz de adyacencia
int n, max_match, xy[N], yx[N], slackx[N],prev2[N];//n=cantidad de nodos
bool S[N], T[N]; //sets S and T in algorithm
void add_to_tree(int x, int prevx) {
  S[x] = true, prev2[x] = prevx;
  forn(y, n) if (lx[x] + ly[y] - cost[x][y] < slack[y] - EPS)
    slack[y] = lx[x] + ly[y] - cost[x][y], slackx[y] = x;
}
void update_labels(){
  tipo delta = INF;
  forn (y, n) if (!T[y]) delta = min(delta, slack[y]);
  forn (x, n) if (S[x]) lx[x] -= delta;
  forn (y, n) if (T[y]) ly[y] += delta; else slack[y] -= delta;
}
void init_labels(){
  zero(lx), zero(ly);
  forn (x,n) forn(y,n) lx[x] = max(lx[x], cost[x][y]);
}
void augment() {
  if (max_match == n) return;
  int x, y, root, q[N], wr = 0, rd = 0;
  memset(S, false, sizeof(S)), memset(T, false, sizeof(T));
  memset(prev2, -1, sizeof(prev2));
  forn (x, n) if (xy[x] == -1){
    q[wr++] = root = x, prev2[x] = -2;
    S[x] = true; break; }
  forn (y, n) slack[y] = lx[root] + ly[y] - cost[root][y], slackx[y] = root;
  while (true){
    while (rd < wr){
      x = q[rd++];
      for (y = 0; y < n; y++) if (cost[x][y] == lx[x] + ly[y] && !T[y]){
        if (yx[y] == -1) break; T[y] = true;
        q[wr++] = yx[y], add_to_tree(yx[y], x); }
      if (y < n) break; }
    if (y < n) break;
    update_labels(), wr = rd = 0;
    for (y = 0; y < n; y++) if (!T[y] && slack[y] == 0){
      if (yx[y] == -1){x = slackx[y]; break;}
      else{
        T[y] = true;
        if (!S[yx[y]]) q[wr++] = yx[y], add_to_tree(yx[y], slackx[y]);
      }}
    if (y < n) break; }
  if (y < n){
    max_match++;
    for (int cx = x, cy = y, ty; cx != -2; cx = prev2[cx], cy = ty)
      ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
    augment(); }
}
tipo hungarian(){
  tipo ret = 0; max_match = 0, memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx)), init_labels(), augment(); //steps 1-3
  forn (x,n) ret += cost[x][xy[x]]; return ret;
}
```

7.16. Dynamic Conectivity

```
struct UnionFind {
    int n, comp;
    vector<int> pre,si,c;
    UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
        forn(i,n) pre[i] = i; }
    int find(int u){return u==pre[u]?u:find(pre[u]);}
    bool merge(int u, int v) {
        if((u=find(u))==(v=find(v))) return false;
        if(si[u]<si[v]) swap(u, v);
        si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
        return true;
    }
    int snap(){return sz(c);}
    void rollback(int snap){
        while(sz(c)>snap){
            int v = c.back(); c.pop_back();
            si[pre[v]] -= si[v], pre[v] = v, comp++;
        }
    }
};
enum {ADD,DEL,QUERY};
struct Query {int type,u,v;};
struct DynCon {
    vector<Query> q;
    UnionFind dsu;
    vector<int> match,res;
    map<ii,int> last;//se puede no usar cuando hay identificador para cada arista (mejora poco)
    DynCon(int n=0):dsu(n){}
    void add(int u, int v) {
        if(u>v) swap(u,v);
        q.pb((Query){ADD, u, v}), match.pb(-1);
        last[ii(u,v)] = sz(q)-1;
    }
    void remove(int u, int v) {
        if(u>v) swap(u,v);
        q.pb((Query){DEL, u, v});
        int prev = last[ii(u,v)];
        match[prev] = sz(q)-1;
        match.pb(prev);
    }
    void query() {//podria pasarle un puntero donde guardar la respuesta
        q.pb((Query){QUERY, -1, -1}), match.pb(-1);}
    void process() {
        forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz(q);
        go(0,sz(q));
    }
    void go(int l, int r) {
        if(l+1==r){
            if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
                res.pb(dsu.comp);//aqui query=cantidad de componentes conexas
            return;
        }
        int s=dsu.snap(), m = (l+r) / 2;
        forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v);
        go(l,m);
        dsu.rollback(s);
        s = dsu.snap();
        forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].v);
        go(m,r);
        dsu.rollback(s);
    }
}dc;
```

8. Network Flow

8.1. MaxFlow mal robados

```
// Push relabel in O(V^2 E^0.5) with gap heuristic
// It's quite fast
template<typename flow_t = long long>
struct PushRelabel {
    struct Edge {
        int to, rev;
        flow_t f, c;
    };
    vector<vector<Edge> > G;
    vector<flow_t> ec;
    vector<Edge*> cur;
    vector<vector<int> > hs;
    vector<int> H;
    PushRelabel(int n) : G(n), ec(n), cur(n), hs(2*n), H(n) {}
    void add_edge(int s, int t, flow_t cap, flow_t rcap=0) {
        if (s == t) return;
        Edge a = {t, (int)G[t].size(), 0, cap};
        Edge b = {s, (int)G[s].size(), 0, rcap};
        G[s].pb(a);
        G[t].pb(b);
    }
    void add_flow(Edge& e, flow_t f) {
        Edge &back = G[e.to][e.rev];
        if (not ec[e.to] and f)
            hs[H[e.to]].pb(e.to);
        e.f += f; e.c -= f;
        ec[e.to] += f;
        back.f -= f; back.c += f;
        ec[back.to] -= f;
    }
    flow_t max_flow(int s, int t) {
        int v = G.size();
        H[s] = v;
        ec[t] = 1;
        vector<int> co(2*v);
        co[0] = v-1;
        for(int i=0;i<v;++i) cur[i] = G[i].data();
        for(auto &e:G[s]) add_flow(e, e.c);
        if(hs[0].size())
        for (int hi = 0;hi>=0;) {
            int u = hs[hi].back();
            hs[hi].pop_back();
            while (ec[u] > 0) // discharge u
                if (cur[u] == G[u].data() + G[u].size()) {
                    H[u] = 1e9;
                    for(auto &e:G[u])
                        if (e.c and H[u] > H[e.to]+1)
                            H[u] = H[e.to]+1, cur[u] = &e;
                    if (++co[H[u]], !--co[hi] and hi < v)
                        for(int i=0;i<v;++i)
                            if (hi < H[i] and H[i] < v){
                                --co[H[i]];
                                H[i] = v + 1;
                            }
                    hi = H[u];
                } else if (cur[u]->c and H[u] == H[cur[u]->to]+1)
                    add_flow(*cur[u], min(ec[u], cur[u]->c));
                else ++cur[u];
            while (hi>=0 and hs[hi].empty()) --hi;
        }
        return -ec[s];
    }
};

// Chinese HLPP
// VERY fast
template <int MAXN, class T = int> struct HLPP {
    const T INF = numeric_limits<T>::max();
    struct edge {
        int to, rev;
        T f;
    };
    int s = MAXN - 1, t = MAXN - 2;
    vector<edge> adj[MAXN];
    vector<int> lst[MAXN], gap[MAXN];
    T excess[MAXN];
    int highest, height[MAXN], cnt[MAXN], work;
    void addEdge(int from, int to, int f, bool isDirected = true) {
        adj[from].push_back({to, adj[to].size(), f});
        adj[to].push_back({from, adj[from].size() - 1, isDirected ? 0 : f});
    }
    void updHeight(int v, int nh) {
        work++;
        if (height[v] != MAXN)
            cnt[height[v]]--;
        height[v] = nh;
        if (nh == MAXN)
            return;
        cnt[nh]++, highest = nh;
        gap[nh].push_back(v);
        if (excess[v] > 0)
            lst[nh].push_back(v);
    }
    void globalRelabel() {
        work = 0;
        for(int &h : height) h = MAXN;
        memset(cnt, 0, sizeof cnt);
        for (int i = 0; i < highest; i++)
            lst[i].clear(), gap[i].clear();
        height[t] = 0;
        queue<int> q({t});
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (auto &e : adj[v])
                if (height[e.to] == MAXN && adj[e.to][e.rev].f > 0)
                    q.push(e.to), updHeight(e.to, height[v] + 1);
            highest = height[v];
        }
    }
    void push(int v, edge &e) {
        if (excess[e.to] == 0)
            lst[height[e.to]].push_back(e.to);
        T df = min(excess[v], e.f);
        e.f -= df, adj[e.to][e.rev].f += df;
        excess[v] -= df, excess[e.to] += df;
    }
    void discharge(int v) {
        int nh = MAXN;
        for (auto &e : adj[v]) {
            if (e.f > 0) {
                if (height[v] == height[e.to] + 1) {
                    push(v, e);
                    if (excess[v] <= 0)
                        return;
                } else
                    nh = min(nh, height[e.to] + 1);
            }
        }
        if (cnt[height[v]] > 1)
            updHeight(v, nh);
        else {
            for (int i = height[v]; i <= highest; i++) {
                for (auto j : gap[i])
                    updHeight(j, MAXN);
                gap[i].clear();
            }
        }
    }
    T calc(int heur_n = MAXN) {
        memset(excess, 0, sizeof excess);
        excess[s] = INF, excess[t] = -INF;
        globalRelabel();
        for (auto &e : adj[s])
            push(s, e);
        for (; highest >= 0; highest--) {
            while (!lst[highest].empty()) {
                int v = lst[highest].back();
                lst[highest].pop_back();
                discharge(v);
                if (work > 4 * heur_n)
                    globalRelabel();
            }
        }
        return excess[t] + INF;
    }
};
```

8.2. Dinic

```

const int MAX = 300;
// Corte minimo: vertices con dist[v]>=0 (del lado de src) VS.  dist[v]==-1 (del lado del dst)
// Para el caso de la red de Bipartite Matching (Sean V1 y V2 los conjuntos mas proximos a src y dst respectivamente):
// Reconstruir matching: para todo v1 en V1 ver las aristas a vertices de V2 con it->f>0, es arista del Matching
// Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con dist[v]>0
// Max Independent Set: tomar los vertices NO tomados por el Min Vertex Cover
// Max Clique: construir la red de G complemento (debe ser bipartito!) y encontrar un Max Independet Set
// Min Edge Cover: tomar las aristas del matching + para todo vertices no cubierto hasta el momento, tomar cualquier arista de el
int nodes, src, dst;
int dist[MAX], q[MAX], work[MAX];
struct Edge {
    int to, rev;
    ll f, cap;
    Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(cap) {}
};
vector<Edge> G[MAX];
void addEdge(int s, int t, ll cap){
    G[s].pb(Edge(t, sz(G[t]), 0, cap)), G[t].pb(Edge(s, sz(G[s])-1, 0, 0));}
bool dinic_bfs(){
    fill(dist, dist+nodes, -1), dist[src]=0;
    int qt=0; q[qt++]=src;
    for(int qh=0; qh<qt; qh++){
        int u =q[qh];
        forall(e, G[u]){
            int v=e->to;
            if(dist[v]<0 && e->f < e->cap)
                dist[v]=dist[u]+1, q[qt++]=v;
        }
    }
    return dist[dst]>=0;
}
ll dinic_dfs(int u, ll f){
    if(u==dst) return f;
    for(int &i=work[u]; i<sz(G[u]); i++){
        Edge &e = G[u][i];
        if(e.cap<=e.f) continue;
        int v=e.to;
        if(dist[v]==dist[u]+1){
                ll df=dinic_dfs(v, min(f, e.cap-e.f));
                if(df>0){
                        e.f+=df, G[v][e.rev].f-= df;
                        return df;  }
        }
    }
    return 0;
}
ll maxFlow(int _src, int _dst){
    src=_src, dst=_dst;
    ll result=0;
    while(dinic_bfs()){
        fill(work, work+nodes, 0);
        while(ll delta=dinic_dfs(src,INF))
            result+=delta;
    }
    // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1 forman el min-cut
    return result; }
```

8.3. Konig

```
// asume que el dinic YA ESTA tirado
// asume que nodes-1 y nodes-2 son la fuente y destino
int match[maxnodes]; // match[v]=u si u-v esta en el matching, -1 si v no esta matcheado
int s[maxnodes]; // numero de la bfs del koning
queue<int> kq;
// s[e]%2==1  o si e esta en V1 y s[e]==-1-> lo agarras
void koning() {//O(n)
  forn(v,nodes-2) s[v] = match[v] = -1;
  forn(v,nodes-2) forall(it,g[v]) if (it->to < nodes-2 && it->f>0)
    { match[v]=it->to; match[it->to]=v;}
  forn(v,nodes-2) if (match[v]==-1) {s[v]=0;kq.push(v);}
  while(!kq.empty()) {
    int e = kq.front(); kq.pop();
    if (s[e]%2==1) {
      s[match[e]] = s[e]+1;
      kq.push(match[e]);
    } else {

      forall(it,g[e]) if (it->to < nodes-2 && s[it->to]==-1) {
        s[it->to] = s[e]+1;
        kq.push(it->to);
      }
    }
  }
}
```

8.4. Min-cost Max-flow

```
const int MAXN=10000;
typedef ll tf;
typedef ll tc;
const tf INFFLUJO = 1e14;
const tc INFCOSTO = 1e14;
struct edge {
  int u, v;
  tf cap, flow;
  tc cost;
  tf rem() { return cap - flow; }
};
int nodes; //numero de nodos
vector<int> G[MAXN]; // limpiar!
vector<edge> e;  // limpiar!
void addEdge(int u, int v, tf cap, tc cost) {
  G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
  G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
}
tc dist[MAXN], mnCost;
int pre[MAXN];
tf cap[MAXN], mxFlow;
bool in_queue[MAXN];
void flow(int s, int t) {
  zero(in_queue);
  mxFlow=mnCost=0;
  while(1){
    fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
    memset(pre, -1, sizeof(pre)); pre[s]=0;
    zero(cap); cap[s] = INFFLUJO;
    queue<int> q; q.push(s); in_queue[s]=1;
    while(sz(q)){
      int u=q.front(); q.pop(); in_queue[u]=0;
      for(auto it:G[u]) {
        edge &E = e[it];
        if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9){ // ojo EPS
          dist[E.v]=dist[u]+E.cost;
          pre[E.v] = it;
          cap[E.v] = min(cap[u], E.rem());
          if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
        }
      }
    }
    if (pre[t] == -1) break;
    mxFlow +=cap[t];
    mnCost +=cap[t]*dist[t];
    for (int v = t; v != s; v = e[pre[v]].u) {
      e[pre[v]].flow += cap[t];
      e[pre[v]^1].flow -= cap[t];
    }
  }
}
```

9. Template

```
#include <bits/stdc++.h>
using namespace std;
#define forr(i,a,b) for(int i=(a); i<int(b); ++i)
#define forn(i,n) forr(i,0,n)
#define dforn(i,n) for(int i=int(n)-1; i>=0; --i)
#define dprint(v) cout << #v"=" << v << endl //;)
#define zero(v) memset(v, 0, sizeof(v))
#define forall(it,v) for(auto it=v.begin();it!=v.end();++it)
#define pb push_back
#define fst first
#define snd second
typedef long long ll;
typedef long double real;
typedef pair<int,int> pii;
typedef pair<ll, ll> pll;

const int MAXN=42069;

int main() {
    // freopen("input.in", "r", stdin);
    ios::sync_with_stdio(0); cin.tie(nullptr); cout.tie(nullptr);

    return 0;
}
```

10. Ayudamemoria

Cant. decimales

```
#include <iomanip>
cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```
#include <iomanip>
cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de linea

```
#include <sstream>
//hacer cin.ignore() antes de getline()
while(getline(cin, line)){
     istringstream is(line);
     while(is >> X)
       cout << X << "␣";
     cout << endl;
}
```

Aleatorios

```
#define RAND(a, b) (rand()%(b-a+1)+a)
srand(time(NULL));
```

Doubles Comp.

```
const double EPS = 1e-9;
x == y  <=> fabs(x-y) < EPS
x >  y  <=> x > y + EPS
x >= y  <=> x > y - EPS
```

Muahaha

```
#include <signal.h>
void divzero(int p){
  while(true);}
void segm(int p){
  exit(0);}
//in main
signal(SIGFPE, divzero);
signal(SIGSEGV, segm);
```

Mejorar velocidad

```
ios::sync_with_stdio(false);
```

Mejorar velocidad 2

```
//Solo para enteros positivos
inline void Scanf(int& a){
  char c = 0;
  while(c<33) c = getc(stdin);
  a = 0;
  while(c>33) a = a*10 + c - '0', c = getc(stdin);
}
```

Expandir pila

```
#include <sys/resource.h>
rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur=1024L*1024L*256L;//256mb
setrlimit(RLIMIT_STACK, &rl);
```

Leer del teclado

```
freopen("/dev/tty", "a", stdin);
```

Iterar subconjunto

```
for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```