Índice

1. Referencia

Algorítmo	Parámetros	Función
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 $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [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] $\forall i$ $\in$[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: 2^K > N. Usar [ ] para llenar
// el arreglo y luego build().
struct RMQ {
    const static int K = ;
    tipo vec[K][1 << K];
    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(n log n)
        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).
typedef int node; // Tipo de los nodos
#define MAXN 100000
#define operacion(x, y) max(x, y)
const int neutro = 0;
struct RMQ {
  int sz;
  node t[4*MAXN];
  node &operator [](int p){ return t[sz + p]; }
  void init(int n){ // O(n lg n)
    sz = 1 << (32 - __builtin_clz(n));
    forn(i, 2*sz) t[i] = neutro;
  }
    void updall(){//O(n)
        dforsn(i,0, sz){
            t[i] = operacion(t[2*i], t[2*i + 1]);
```

```
        }
    }
  node get(int i, int j){ return get(i, j, 1, 0, sz); }
  node get(int i, int j, int n, int a, int b){ // O(lg n)
    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, node val){ // O(lg n)
    for(p += sz; p > 0 && t[p] != val;){
      t[p] = val;
      p /= 2;
      val = operacion(t[p*2], t[p*2 + 1]);
    }
  }
} rmq;
// Uso:
cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3. RMQ (lazy)

```
// TODO: Las funciones pueden pasarse a traves de template. Quedara
    mejor sacar el struct tipo y reemplazar por todo en template?
// Tipo de ejemplo:
struct Tipo {
    const static int neutro = 0;
    int val;

    Tipo(int _val=0) : val(_val) {}

    Tipo operator + (const Tipo &o) const { return val + o.val; }
    Tipo& operator += (const Tipo &o) { val += o.val; return *this; }
};
// Dado un arreglo y una operacion asociativa con neutro:
// get(i, j) opera sobre el rango [i, j).
template <int N, class TNodo, class TAlt>
struct RMQ {
  int sz;
  TNodo t[4*N];
  TAlt dirty[4*N];
  TNodo &operator [](int p){ return t[sz + p]; }
    void init(int n) { // O(n lg n)
        sz = 1 << (32 - __builtin_clz(n));
        forn(i, 2*sz) {
            t[i] = TNodo::neutro;
            dirty[i] = TAlt::neutro;
        }
    }
  void push(int n, int a, int b){ // Propaga el dirty a sus hijos
    if (dirty[n].val != TAlt::neutro){
      t[n] += dirty[n].val*(b - a); // Altera el nodo
      if (n < sz){
        dirty[2*n] += dirty[n];
        dirty[2*n + 1] += dirty[n];
      }
      dirty[n] = TAlt::neutro;
    }
  }
  TNodo get(int i, int j, int n, int a, int b){ // O(lg n)
    if (j <= a || i >= b) return TNodo::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 get(i, j, 2*n, a, c) + get(i, j, 2*n + 1, c, b);
  }
  TNodo get(int i, int j){ return get(i, j, 1, 0, sz); }
  // Altera los valores en [i, j) con una alteracion de val
  void modify(TAlt val, int i, int j, int n, int a, int b){ // O(lg n)
    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;
    modify(val, i, j, 2*n, a, c); modify(val, i, j, 2*n + 1, c, b);
    t[n] = t[2*n] + t[2*n + 1];
  }
  void modify(TAlt val, int i, int j){ modify(val, i, j, 1, 0, sz); }
};
```

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 tomar 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 *upd(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(upd(pos, new_val, t->l, tl, tm), t->r);
  else return new node(t->l, upd(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. Sliding window RMQ

```
// Para max pasar less y -INF
template <class T, class Compare, T INF>
struct RMQ {
    deque<T> d; queue<T> q;
    void push(T v) {
        while (!d.empty() && Compare()(d.back(), v)) d.pop_back();
        d.pb(v);
        q.push(v);
    }

    void pop() {
        if (!d.empty() && d.front()==q.front()) d.pop_front();
        q.pop();
    }

    T getMax() {
        return d.empty() ? INF : d.front();
    }

    int size() {
        return si(q);
    }
};
RMQ<ll, less<ll>, -INF> rmq;
```

2.6. Fenwick Tree

```
// Para 2D: tratar cada columna como un Fenwick Tree,
// agregando un for anidado en cada operacion.
// Trucazo para 2D: si los elementos no se repiten,
// se puede usar un ordered set para memoria O(n*log^2(n))
typedef ll tipo;
struct Fenwick {
    static const int sz = (1 << 18) + 1;
    tipo t[sz];
    void adjust(int p, tipo v){ // p en [1, sz), O(lg n)
        for(int i = p; i < sz; i += (i & -i)) t[i] += v;
    }
    tipo sum(int p){ // Suma acumulada en [1, p], O(lg n)
        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); }
    int lower_bound(tipo v) { // Menor x con suma acumulada >= v, O(lg n
        )
        int x = 0, d = sz-1;
        if(v > t[d]) return sz;
        for(; d; d >>= 1) if(t[x|d] < v) v -= t[x |= d];
        return x+1;
    }
};
```

2.7. Union Find

```
struct UF { // Operations take O(log*(n))
    vi p,s;
    UF(int n){ p.resize(n), iota(all(p), 0), s.assign(n, 1); }
    int find(int i){
      while (p[i] != i) p[i] = p[p[i]], i = p[i];
      return i;
    }
    bool con(int x, int y){ return find(x) == find(y); }
    bool join(int x, int y){
      x = find(x), y = find(y);
        if (con(x, y)) return false;

      if (s[x] < s[y]) p[x] = y, s[y] += s[x];
      else p[y] = x, s[x] += s[y];
        return true;
    }
};
```

2.8. Disjoint Intervals

```
// Guarda intervalos como [first, second]
// En caso de colision, los une en un solo intervalo
bool operator <(const pii &a, const pii &b){ return a.first < b.first; }
struct disjoint_intervals {
  set<pii> segs;
  void insert(pii v){ // O(lg n)
    if(v.second - v.first == 0.0) return; // Cuidado!
    set<pii>::iterator it, at;
    at = it = segs.lower_bound(v);
    if(at != segs.begin() && (--at)->second >= v.first){
      v.first = at->first;
      --it;
    }
    for(; it!=segs.end() && it->first <= v.second; segs.erase(it++))
      v.second = max(v.second, it->second);
    segs.insert(v);
  }
};
```

2.9. RMQ (2D)

```
struct RMQ2D { // n filas, m columnas
  int sz;
  RMQ t[4*MAXN]; // t[i][j] = i fila, j columna
  RMQ &operator [](int p){ return t[sz/2 + p]; }
  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(lg(m)*lg(n))
    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(lg(m)*lg(n)), 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;
// Ejemplo para inicializar una matriz de n filas por m columnas
RMQ2D rmq; rmq.init(n, m);
forn(i, n) forn(j, m){
  int v; cin >> v; rmq.set(i, j, v);
}
```

2.10. Big Int

```
#define BASE 10
#define LMAX 1000
int pad(int x){
    x--; int c = 0;
    while(x) x /= 10, c++;
    return c;
}
const int PAD = pad(BASE);
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){
        int sz = si(x);
        l = (sz-1)/PAD + 1;
        fill(n, n+LMAX, 0);
        ll r = 1;
        forn(i,sz){
            if(i % PAD == 0) r = 1;
            n[i/PAD] += r*(x[sz-1-i]-'0');
            r *= 10;
        }
    }
    void out() const {
        cout << n[l-1] << setfill('0');
        dforn(i,l-1) cout << setw(PAD) << n[i];
    }
    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 {c,!q};
}
bint &operator -=(bint &a, const bint &b){ return a = lresta(a, b).fst;
    }
bint operator -(const bint &a, const bint &b){ return lresta(a, b).fst;
    }
bool operator <(const bint &a, const bint &b){ return !lresta(a, b).snd;
     }
bool operator <=(const bint &a, const bint &b){ return lresta(b, a).snd;
     }
bool operator ==(const bint &a, const bint &b){ return a <= b && b <= a;
     }
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 {c,rm};
}
bint operator /(const bint &a, ll b){ return ldiv(a, b).fst; }
ll operator %(const bint &a, ll b){ return ldiv(a, b).snd; }
pair<bint,bint> ldiv(const bint &a, const bint &b){
    bint c, 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 {c,rm};
}
bint operator /(const bint &a, const bint &b){ return ldiv(a, b).fst; }
bint operator %(const bint &a, const bint &b){ return ldiv(a, b).snd; }
bint gcd(bint a, bint b){
    while(b != bint(0)){
        bint r = a % b;
        a = b, b = r;
    }
    return a;
}
```

2.11. Hash

```
mt19937 rng;
struct hashing {
    int mod, mul;

    bool prime(int n) {
        for (int d = 2; d*d <= n; d++) if (n%d == 0) return false;
        return true;
    }

    void setValues(int mod, int mul) {
        this->mod = mod;
        this->mul = mul;
    }

    void randomize() {
        rng.seed(time(0));
        mod = uniform_int_distribution<>(0, (int) 5e8)(rng) + 1e9;
        while (!prime(mod)) mod++;
        mul = uniform_int_distribution<>(2,mod-2)(rng);
    }

    vi h, pot;
    void process(const string &s) {
        h.resize(si(s)+1);
        pot.resize(si(s)+1);
        h[0] = 0; forn(i,si(s)) h[i+1] = (((ll)h[i] * mul) + s[i]) % mod
```

```
            ;
        pot[0] = 1; forn(i,si(s)) pot[i+1] = (ll) pot[i] * mul % mod;
    }

    int hash(int i, int j) {
        int res = h[j] - (ll) h[i] * pot[j-i] % mod;
        if (res < 0) res += mod;
        return res;
    }

    int hash(const string &s) {
        int res = 0;
        for (char c : s) res = (res * (ll) mul + c) % mod;
        return res;
    }

};

hashing h1,h2;
```

2.12. Modnum

```
const int mod = 998244353;
struct num {
    int a;
    num(int b = 0){ a = b; }
    operator int(){ return a; }
    num operator +(num b){ return a+b.a > mod ? a+b.a-mod : a+b.a; }
    num operator -(num b){ return a-b.a < 0 ? a-b.a+mod : a-b.a; }
    num operator *(num b){ return int(ll(a)*b.a % mod); }
    num operator ^(num e){
    if(!e.a) return 1;
        num q = (*this)^num(e.a/2);
    return e.a & 1 ? q*q*(*this) : q*q;
    }
    num operator ++(int x){ return a++; }
};
int norm(ll x){ return x < 0 ? int(x % mod + mod) : int(x % mod); }
num inv(num x){ return x^num(mod-2); } // mod must be prime
num operator /(num a, num b){ return a*inv(b); }
num neg(num x){ return x.a ? -x.a+mod : 0; }
istream& operator >>(istream &i, num &x){ i >> x.a; return i; }
ostream& operator <<(ostream &o, const num &x){ o << x.a; return o; }
// Cast integral values to num in arithmetic expressions!
```

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 int Value; // pii(profundidad, nodo)
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.first += p->dirty;
  p->mini.first += 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 : pii(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, si(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.first << ' ';
  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 : si(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(si(c) && m==c.back().m) { l.h=min(h, c.back().h), c.pop_back
           (); if(pos) pos--; }
       while(si(c)>=2 && irre(c[si(c)-2], c[si(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 = si(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;
struct CHTBruto {
  vector<Line> c;
  bool mx;
  CHTBruto(bool mx=0):mx(mx){}//mx=si las query devuelven el max o el
      min
  void add(tipo m, tipo h) {
    Line l=(Line){m, h};
       c.pb(l);
  }
  tipo eval(tipo x){
       tipo r=c[0].m*x+c[0].h;
       forn(i, si(c)) if(mx) r=max(r, c[i].m*x+c[i].h);
                      else r=min(r, c[i].m*x+c[i].h);
       return r;
  }
```

```
} chb;
```

2.16. Convex Hull Trick (Dynamic)

```
struct Line {
    tint 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;
        const Line *s=succ(it);
        if(!s) return 0;
        tint 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(tint m, tint 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));
    }
    tint eval(tint 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

```
//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 índices

```
#include <cassert>

#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;
//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
```

3. Algoritmos

3.1. Longest Increasing Subsecuence

```
const int MAXN = 1e5+10, INF = 1e8;

//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
pii 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.pb(a[i]);
  rec(p[i]);
}
int lis(){//O(nlogn)
  d[0] = pii(-INF, -1); forn(i, N) d[i+1]=pii(INF, -1);
  forn(i, N){
    int j = upper_bound(d, d+N+1, pii(a[i], INF))-d;
    if (d[j-1].first < a[i]&&a[i] < d[j].first){ // check < por <= en d[j-1]
      p[i]=d[j-1].second;
      d[j] = pii(a[i], i);
    }
  }
  R.clear();
  dforsn(i, 0, 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(){//(n+q)*sqrt(n))*(O(add())+O(remove()))
    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

Definición: permite calcular todas las substrings de una string s que son palíndromos de longitud impar (y par, ver *observación*) . Para ello, mantiene un arreglo *len* tal que $len[i]$ almacena la longitud del palíndromo impar maximal con centro en i.
Explicación algoritmo: muy similar al algoritmo para calcular la *función Z*. Mantiene el palíndromo que termina más a la derecha entre todos los palíndromos ya detectados. Para calcular $len[i]$, utiliza la información ya calculada si i está dentro de $[l, r]$, y luego corre el algoritmo trivial.
Observación: para calcular los palíndromos de longitud par, basta con utilizar el mismo algoritmo con la cadena $s_0\#s_1\#...\#s_{n-1}$.

```
vi pal_array(string s)
{
    int n = si(s);
    s = "@" + s + "$";

    vi len(n + 1);
    int l = 1, r = 1;

    forsn(i, 1, n+1) {
        len[i] = min(r - i, len[l + (r - i)]);

        while (s[i - len[i]] == s[i + len[i]]) len[i]++;

        if (i + len[i] > r) l = i - len[i], r = i + len[i];
    }

    len.erase(begin(len));
    return len;
}
```

4.2. KMP

```
// pref[i] = max borde de s[0..i] = failure function al intentar
    matchear con s[i+1]
vi prefix_function(string &s) {
    int n = si(s); vi pi(n);
    forsn(i, 1, n) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j]) j = pi[j-1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}

vi find_occurrences(string &s, string &t) { //apariciones de t en s
    vi pre = prefix_function(t), res;
    int n = si(s), m = si(t), j = 0;
    forn(i, n) {
        while (j > 0 && s[i] != t[j]) j = pre[j-1];
        if (s[i] == t[j]) j++;
        if (j == m) {
            res.pb(i-j+1);
            j = pre[j-1];
        }
    }
    return res;
}

// aut[i][c] = (next o failure function) al intentar matchear s[i] con c
void compute_automaton(string s, vector<vi>& aut) {
    s += '#'; // separador!
    int n = si(s);
    vi pi = prefix_function(s);
    aut.assign(n, vi(26));

    forn(i, n) forn(c, 26)
        if (i > 0 && 'a' + c != s[i])
            aut[i][c] = aut[pi[i-1]][c];
        else
            aut[i][c] = i + ('a' + c == s[i]);
}
```

4.3. Trie

```
struct trie {
    int p = 0, w = 0;
    map<char,trie*> c;
    trie(){}
    void add(const string &s){
        trie *x = this;
        forn(i,si(s)){
```

```
            if(!x->c.count(s[i])) x->c[s[i]] = new trie();
            x = x->c[s[i]];
            x->p++;
        }
        x->w++;
    }
    int find(const string &s){
        trie *x = this;
        forn(i,si(s)){
            if(x->c.count(s[i])) x = x->c[s[i]];
            else return 0;
        }
        return x->w;
    }
    void erase(const string &s){
        trie *x = this, *y;
        forn(i,si(s)){
            if(x->c.count(s[i])) y = x->c[s[i]], y->p--;
            else return;
            if(!y->p){
                x->c.erase(s[i]);
                return;
            }
            x = y;
        }
        x->w--;
    }
  void print(string tab = "") {
    for(auto &i : c) {
      cerr << tab << i.fst << endl;
      i.snd->print(tab + "--");
    }
  }
};
```

4.4. Suffix Array (largo, nlogn)

```
const int MAXN = 1e3+10;
#define rBOUND(x) (x<n? r[x] : 0)
//sa will hold the suffixes in order.
int sa[MAXN], r[MAXN], n;
string s; //input string, n=si(s)

int f[MAXN], tmpsa[MAXN];
void countingSort(int k){
    fill(f, f+MAXN, 0);
  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=si(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[MAXN];
    tmpr[sa[0]]=rank=0;
    forsn(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;
    }
}

//returns (lowerbound, upperbound) of the search
```

4.5. String Matching With Suffix Array

```
//returns (lowerbound, upperbound) of the search
pii stringMatching(string P){ //O(si(P)lgn)
  int lo=0, hi=n-1, mid=lo;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(sa[mid], si(P), P);
    if(res>=0) hi=mid;
```

```
    else lo=mid+1;
  }
  if(s.compare(sa[lo], si(P), P)!=0) return pii(-1, -1);
  pii ans; ans.first=lo;
  lo=0, hi=n-1, mid;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(sa[mid], si(P), P);
    if(res>0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(sa[hi], si(P), P)!=0) hi--;
    // para verdadero upperbound sumar 1
  ans.second=hi;
  return ans;
```

4.6. 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[MAXN], phi[MAXN], PLCP[MAXN];
void computeLCP(){//O(n)
  phi[sa[0]]=-1;
  forsn(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]];
```

4.7. Corasick

```
struct trie{
  map<char, trie> next;
  trie* tran[256];//transiciones del automata
  int idhoja, sihoja;//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<si(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, sihoja=si(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-
        sihoja << endl;
    if(get_nxthoja()) get_nxthoja()->print(p); }
  void matching(const string &s, int p=0){
    print(p); if(p<si(s)) get_tran(s[p])->matching(s, p+1); }
}tri;
```

4.8. Suffix Automaton

Definición Un suffix automaton A es un autómata minimal que reconoce los sufijos de una cadena s.

Conceptos importantes

- A *reconoce* a una cadena s si comenzando desde el nodo inicial llegamos a un terminal.
- Llamamos *endpoints* de una cadena u en s a las posiciones i tal que $u = s_{i-|u|,i-1}$
- Dos substrings u y v de s son *equivalentes* si recorrer el autómata con u y con v nos lleva al mismo nodo. Esto equivale a que los endpoints de u y de v en s sean los mismos.

- Los nodos del automáta corresponden a las *clases de equivalencia* bajo la relación anterior.
- Si tomamos la *cadena minimal* de una clase y removemos la primera letra, nos movemos al padre en el *suffix tree* de s'. Esto se señaliza mediante *suffix links*.
- Si tomamos la *cadena maximal* de una clase y agregamos una letra al principio, nos movemos a otra clase de equivalencia, recorriendo un *suffix link* en sentido inverso, es decir, nos movemos a los hijos en el suffix tree de s'.

Problemas clásicos

- Determinar si w es subcadena de s: simplemente correr el autómata.
- Determinar si w es sufijo de s: correr el autómata y ver si caemos en un terminal.
- Contar cantidad de subcadenas distintas de s: esto es igual a la cantidad de caminos en el autómata y se calcula mediante una DP.
- Contar cantidad de apariciones de w en s: correr autómata con w. Llamemos u al nodo en el que terminamos, la cantidad de apariciones es la cantidad de caminos en A que comienzan en u y llegan a un terminal.
- Encontrar dónde aparece w por primera vez en s: correr autómata con w. Llamemos u al nodo en el que terminamos, esto equivale a calcular el camino más largo del autómata a partir del nodo u.
- Encontrar las posiciones de todas las apariciones de w en s: agregar '$á s, encontrar el nodo u en el que finaliza w, armar el *suffix tree*, encontrar todas las hojas en el subárbol con raíz en u, cada hoja corresponde a un prefijo y por lo tanto a una aparición.

```
struct state {
  int len, link;
  map<char,int> next;
  state() { }
};
const int MAXLEN = 1e5+10;
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;
}
```

4.9. Z Function

Definición La función Z para una string s de longitud n es un arreglo a de la misma longitud tal que $a[i]$ es la *máxima cantidad de caracteres* comenzando desde la posición i que coinciden con los primeros caracteres de s. Es decir, es el *máximo prefijo común*.
Observación $z[0]$ no está bien definido, pero se asume igual a 0.
Algoritmo La idea es mantener el máximo *match* (es decir, el segmento $[l, r]$ con máximo r tal que se sabe que $s[0..r-l] = s[l..r]$).
Siendo i el índice actual (del que queremos calcular la función Z), el algoritmo se divide en dos casos:

- $i > r$: la posición está fuera de lo que hemos procesado. Se corre el *algoritmo trivial*.
- $i <= r$: la posición está dentro del *match actual*, por lo que se puede utilizar como aproximación inicial $z[i] = min(r-i+1, z[i-l])$, y luego correr el *algoritmo trivial*.

Problemas clásicos

- Buscar una subcadena: concatenamos p con t (utilizando un separador). Hay una aparición si la función Z matcheó tantos caracteres como la longitud de p.

```
int z[N]; // z[i] = i==0 ? 0 : max k tq s[0,k) match with s[i,i+k)
void z_function(string &s, int z[]) {
    int n = si(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;
    }
}
```

4.10. Palindrome

```
bool palindrome(ll x){
  string s = to_string(x); int n = si(s);
  forn(i,n/2) if(s[i] != s[n-i-1]) return 0;
  return 1;
}
```

5. Geometría

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();}
double dist_sq(pto a, pto b){return (b-a).norm_sq();}
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 = 0;
  forn(i,si(P)){
    int j = (i+1) % si(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=si(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[si(pt)-1], pt[0])) return 0;
  int a=1, b=si(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 >=-EPS to delete collinear points!
void chull(vector<pto>& P, vector<pto> &S){
  S.clear();
  sort(P.begin(), P.end());//first x, then y
  forn(i, si(P)){//lower hull
    while(si(S)>= 2 && S[si(S)-1].left(S[si(S)-2], P[i])) S.pop_back();
    S.pb(P[i]);
  }
  S.pop_back();
  int k=si(S);
  dforn(i, si(P)){//upper hull
    while(si(S) >= k+2 && S[si(S)-1].left(S[si(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)

```cpp
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. DP Opt

Observaciones:
$A[i][j]$ el menor k que logra la solución óptima. En Knuth y D&C la idea es aprovechar los rangos determinados por este arreglo.

6.1. Knuth

Problema de ejemplo: dado un palito de longitud l, con n puntos en los que se puede cortar, determinar el costo mínimo para partir el palito en $n+1$ palitos unitarios (la DP se puede adaptar a k agregando un parámetro extra), donde hay un costo fijo por partir el rango i, j que *cumple la condición suficiente*. Una función de costos que cumple es la distancia entre los extremos $j - i$. El problema clásico de esta pinta es el del ABB óptimo.
Recurrencia original: $dp[i][j] = min_{i<k<j} dp[i][k] + dp[k][j] + C[i][j]$
Condición suficiente: $A[i, j-1] \leq A[i, j] \leq A[i+1, j]$
Es decir, si saco un elemento a derecha el óptimo se mueve a izquierda o se mantiene, y análogo si saco un elemento a izquierda.
Complejidad original: $O(n^3)$
Complejidad optimizada: $O(n^2)$
Solución: iteramos por el tamaño *len* del subarreglo (creciente), y para cada extremo izquierdo l, determinamos el extremo derecho $r = l + len$ e iteramos por los k entre $A[l][r-1]$ y $A[l+1][r]$, actualizando la solución del estado actual.

6.2. Chull

Problema de ejemplo:
Recurrencia original:
Condición suficiente:
Complejidad original:
Complejidad optimizada:
Solución:

6.3. Divide & Conquer

Problema de ejemplo: dado un arreglo de n números con valores $a_1, a_1, \ldots, a_n$, dividirlo en k subarreglos, tal que la suma de los cuadrados del peso total de cada subarreglo es mínimo.
Recurrencia original: $dp[i][j] = min_{k<j} dp[i-1][k] + C[k][j]$
Condición suficiente: $A[i][j] \leq A[i][j+1]$ o (normalmente más fácil de probar) $C[a][d] + C[b][c] \geq C[a][c] + C[b][d]$, con $a < b < c < d$..
La segunda condición suficiente es la intuición de que no conviene que los intervalos se crucen.
Complejidad original: $O(kn^2)$
Complejidad optimizada: $O(kn \log(n))$
Solución: la idea es, para un i determinado, partir el rango $[j_{left}, j_{right})$ al que pertenecen los j que queremos calcular a la mitad, determinar el óptimo y utilizarlo como límite para calcular los demás. Para implementar esto de forma sencilla, se suele utilizar la función recursiva $dp(i, j_{left}, j_{right}, opt_{left}, opt_{right})$ que se encarga de, una vez fijado el punto medio m del rango $[j_{left}, j_{right})$ iterar por los k en $[j_{left}, j_{right})$ para determinar el óptimo opt para m, y continuar calculando $dp(i, j_{left}, m, opt_{left}, opt)$ y $dp(i, m, j_{right}, opt, opt_{right})$.

7. Matemática

7.1. Teoría de números

7.1.1. Teorema de Wilson

$(p-1)! \equiv -1 \pmod{p}$ Siendo p primo.

7.1.2. Pequeño teorema de Fermat

$a^p \equiv a \pmod{p}$ Siendo p primo.

7.1.3. Teorema de Euler

$a^{\varphi(n)} \equiv 1 \pmod{n}$

7.2. Numeros combinatorios copados y como calcularlos

7.2.1. Combinatorios

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

```
int MAXC = 1e3+1, C[MAXC][MAXC];
forn(i,MAXC){ // C[n][k] = C(n, k)
    C[i][0] = C[i][i] = 1;
    forsn(k,1,i) C[i][k] = add(C[i-1][k],C[i-1][k-1]);
}
```

7.2.2. Lucas Theorem

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{mi}{n_i} \pmod{p}$$

where $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$,
and $n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$
$\binom{m}{n} = 0$ if $m < n$.

```
ll lucas(ll n, ll k, int p){ // Calcula C(n,k) % p teniendo C[p][p]
    precalculado, p primo
    ll ans = 1;
    while(n + k){ ans = (ans * C[n%p][k%p]) % p; n/=p; k/=p; }
    return ans;
}
```

7.2.3. Stirling

$\left\{ {n \atop k} \right\}$ = cantidad de formas de particionar un conjunto de n elementos en m subconjuntos no vacíos.
$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$
for $k > 0$ with initial conditions
$$\left\{ {0 \atop 0} \right\} = 1 \quad \text{and} \quad \left\{ {n \atop 0} \right\} = \left\{ {0 \atop n} \right\} = 0 \text{ for } n > 0.$$

```
int MAXS = 1e3+1, S[MAXS][MAXS];
void stirling(){
  S[0][0] = 1;
  forsn(i,1,N) S[i][0] = S[0][i] = 0;
  forsn(i,1,N) forsn(j,1,N)
    S[i][j] = add(mul(S[i-1][j],j),S[i-1][j-1]);
}
```

7.2.4. Bell

B_n = cantidad de formas de particionar un conjunto de n elementos en subconjuntos no vacíos.
$B_0 = B_1 = 1$
$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k.$
$B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\}.$

```
int bell[MAXC+1][MAXC+1];
ll bell(int n){
    bell[0] = 1;
    forsn(i,1,n+1) forsn(k,0,i)
        bell[i] = add(bell[i],mul(C[i-1][k],bell[k]));
}
```

7.2.5. Eulerian

$A_{n,m}$ = cantidad de permutaciones de 1 a n con m ascensos (m elementos mayores que el anterior).
$A(n,m) = (n-m)A(n-1,m-1) + (m+1)A(n-1,m).$

7.2.6. Catalan

C_n = cantidad de árboles binarios de n+1 hojas, en los que cada nodo tiene cero o dos hijos.
$C_n = \binom{2n}{n} - \binom{2n}{n-1} \quad \text{con } n \geq 1.$
$C_0 = 1 \quad \text{y} \quad C_{n+1} = \sum_{i=0}^{n} C_i\, C_{n-i} \quad \text{con } n \geq 0.$

7.3. Heron's formula

It states that the area of a triangle whose sides have lengths a, b, and c is $A = \sqrt{s(s-a)(s-b)(s-c)}$, where s is the semiperimeter of the triangle; that is, $s = \frac{a+b+c}{2}$.

7.4. Sumatorias conocidas

$\sum_{i=0}^{n} \binom{n}{i} = 2^n$
$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$
$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$
$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1)$ (doubles) → Sino ver caso impar y par
$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^{n} i]^2$
$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$

7.5. Ec. Característica

$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$
$p(x) = a_0x^k + a_1x^{k-1} + ... + a_k$
Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$
$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$
Las constantes c_{ij} se determinan por los casos base.

7.6. Aritmetica Modular

```
int mod(int a){ return ((a % M) + M) % M;} // Use for negative numbers (
    the 2nd modulo is avoidable)
int add(int a, int b){ return a+b < M ? a+b : a+b-M; }
int sub(int a, int b){ return a-b >= 0 ? a-b : a-b+M; }
int mul(int a, int b){ return int(ll(a)*b % M); }
int neg(int a){ return add(-a, M); }
```

7.7. Exp. de Numeros Mod.

```
ll pot(ll b, ll e){ // O(log e)
  if(!e) return 1;
  ll q = pot(b, e/2); q = mul(q, q);
  return (e & 1 ? mul(b, q) : q);
}
```

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

```
const int S = 2;
int temp[S][S];
void mul(int a[S][S], int b[S][S]){
    forn(i, S) forn(j, S) temp[i][j] = 0;
    forn(i, S) forn(j, S) forn(k, S) temp[i][j]+=a[i][k]*b[k][j];
    forn(i, S) forn(j, S) a[i][j]=temp[i][j];
```

```
}
void powmat(int a[S][S], ll n, int res[S][S]){
    forn(i, S) forn(j, S) res[i][j]=(i==j);
    while(n){
        if(n&1) mul(res, a), n--;
        else mul(a, a), n/=2;
    }
}
```

7.9. 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 si(vec);}
    Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
        Mat m(si(b),si(b[0]));
        forn(i,si(vec)) forn(j,si(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 = si(vec), m = si(vec[0]), t = si(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 = si(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;
    }
};
```

7.10. Primes and factorization

```
map<ll,int> F;
const int N = 1e7;
int lp[N+1],P[N+1],sp=0; // prime_density(n) ~= n/ln(n)

void sieve(){ // O(N)
  forsn(i,2,N+1){
    if(lp[i] == 0) lp[i] = i, P[sp++] = i;
    for(int j=0; j < sp && P[j] <= lp[i] && i*P[j] <= N; j++) lp[i*P[j]]
        = P[j];
  }
}

void factorize(int x){ // O(log(x)), x <= N, sieve needed
    while(x != 1) F[lp[x]]++, x /= lp[x];
}

void factorize(ll x) { // O(sqrt(x)), no sieve needed
    for(int i = 2; i*i <= x; i++)
        while(x % i == 0) F[i]++, x /= i;
    if(x != 1) F[x]++;
}
```

7.11. Euler's Phi

```
const int N = 1e6;
int lp[N+1],P[N/5],phi[N+1],sp=0; // prime_density(n) ~= n/ln(n)
// lp (least prime) allows fast factorization of numbers <= N

// Euler's totient function (phi) counts the positive integers up to a
    given integer n that are relatively prime to n
void init_phi(){ // Primes and Phi <= N in O(N)
  phi[1] = 1;
  forsn(i,2,N+1){
    if(lp[i] == 0) lp[i] = i, P[sp++] = i, phi[i] = i-1;
    else phi[i] = lp[i] == lp[i/lp[i]] ? phi[i/lp[i]]*lp[i] : phi[i/lp[i
        ]]*(lp[i]-1);
```

```
    for(int j = 0; j < sp && P[j] <= lp[i] && i*P[j] <= N; j++) lp[i*P[j
        ]] = P[j];
  }
}

int eulerPhi(int n){ // O(sqrt(n)) (single number)
    int r = n;
    for(int i = 2; i*i <= n; i++) if(n % i == 0){
        r -= r/i;
        while(n % i == 0) n /= i;
    }
    if(n > 1) r -= r/n;
    return r;
}
```

7.12. Criba

```
const int MAXP = 100100; // no inclusive
int criba[MAXP];
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();
  forsn(i, 2, MAXP) if(!criba[i]) primos.push_back(i);
}
```

7.13. 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

```
// TODO: actualizar macros. Ver que sean compatibles con criba
// INCLUIR CRIBA

//factoriza bien numeros hasta MAXP^2
map<ll,ll> fact(ll n){ //O (cant primos)
  map<ll,ll> ret;
  for (ll 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;
}
```

```cpp
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;
}
```

7.14. Phollard's Rho - Miller-Rabin

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

typedef unsigned long long ull;
ull mulmod(ull a, ull b, ull m){ // 0 <= a, b < m
   long double x; ull c; ll r;
   x = a; c = x * b / m;
   r = (ll)(a * b - c * m) % (ll)m;
   return r < 0 ? r + m : r;
}

ll expmod(ll b, ll e, ll m){ // O(log(b))
  ll ans = 1;
  while(e){
        if(e&1)ans = mulmod(ans, b, m);
        b = mulmod(b, b, m); e >>= 1;
  }
  return ans;
}

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 O(n^0.25)
  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))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)d = gcd(x-y, n);
        else d = gcd(y-x, n);
    }
    return d == n ? rho(n) : d;
}
void fact(ll n, map<ll,int>& f){ //O (lg n)^3
  if(n == 1)return;
  if(rabin(n)){ f[n]++; return; }
  ll q = rho(n); fact(q, f); fact(n/q, f);
}
```

7.15. GCD

```cpp
template<class T> T gcd(T a,T b){return b?__gcd(a,b):a;}
//en C++17 gcd(a,b) predefinido
```

7.16. LCM

```cpp
template<class T> T lcm(T a,T b){return a*(b/gcd(a,b));}
```

```
//en C++17 lcm(a,b) predefinido
```

7.17. Euclides extendido

Dados a y b, encuentra x e y tales que $a * x + b * y = gcd(a, b)$.

```
pair<ll,ll> extendedEuclid (ll a, ll b){ //a * x + b * y = gcd(a,b)
  ll x,y;
  if (b==0) return mp(1,0);
  auto p=extendedEuclid(b,a%b);
  x=p.snd;
  y=p.fst-(a/b)*x;
    if (a*x + b*y == -gcd(a,b)) a = -a, b = -b;
  return mp(x,y);
}
```

7.18. Inversos

```
const int MAXM = 15485867; // Tiene que ser primo
ll inv[MAXM]; //inv[i]*i=1 M M
void calc(int p){//O(p)
  inv[1]=1;
  forsn(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
}
// Llamar calc(MAXM);

int inv(int x){//O(log x)
  return pot(x, eulerphi(M)-1);//si M no es primo(sacar a mano)
  return pot(x, M-2);//si M es primo
}

// Inversos con euclides en O(log(x)) sin precomputo:
// extendedEuclid(a, -m).fst (si coprimos a y m)
```

7.19. Ecuaciones diofánticas

Basado en Euclides extendido. Dados a, b, y r obtiene x e y tales que $a * x + b * y = r$, suponiendo que $gcd(a, b)|r$. Las soluciones son de la forma $(x, y) = (x_1 - b/gcd(a, b) * k_1, x_2 + a/gcd(a, b) * k_2)$ donde x_1 y x_2 son las soluciones particulares que obtuvo Euclides.

```
pair<pair<ll,ll>,pair<ll,ll> > diophantine(ll a,ll b, ll r) {
  //a*x+b*y=r where r is multiple of gcd(a,b);
  ll d=gcd(a,b);
  a/=d; b/=d; r/=d;
  auto p = extendedEuclid(a,b);
  p.fst*=r; p.snd*=r;
  assert(a*p.fst+b*p.snd==r);
  return mp(p,mp(-b,a)); // solutions: (p.fst - b*k1, p.snd + a*k2)
}
```

7.20. Teorema Chino del Resto

Dadas k ecuaciones de la forma $a_i * x \equiv a_i \pmod{n_i}$, encuentra x tal que es solución. Existe una única solución módulo $lcm(n_i)$.

```
#define mod(a,m) ((a)%(m) < 0 ? (a)%(m)+(m) : (a)%(m)) // evita overflow
     al no sumar si >= 0
typedef tuple<ll,ll,ll> ec;
pair<ll,ll> sol(ec c){ //requires inv, diophantine
    ll a=get<0>(c), x1=get<1>(c), m=get<2>(c), d=gcd(a,m);
    if (d==1) return mp(mod(x1*inv(a,m),m), m);
    else return x1%d ? mp(-1LL,-1LL) : sol({a/d,x1/d,m/d});
}
pair<ll,ll> crt(vector< ec > cond) { // returns: (sol, lcm)
  ll x1=0,m1=1,x2,m2;
  for(auto t:cond){
    tie(x2,m2)=sol(t);
    if((x1-x2)%gcd(m1,m2))return mp(-1,-1);
    if(m1==m2)continue;
    ll k=diophantine(m2,-m1,x1-x2).fst.snd,l=m1*(m2/gcd(m1,m2));
    x1=mod(m1*mod(k, l/m1)+x1,l);m1=l; // evita overflow con prop modulo
  }
  return sol(make_tuple(1,x1,m1));
} //cond[i]={ai,bi,mi} ai*xi=bi (mi); assumes lcm fits in ll
```

7.21. 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.;}
```

7.22. Fraction

```
template<class T> T gcd(T a,T b){return b==0?a:gcd(b,a%b);}

struct frac{
  int p,q;
  frac(int p=0, int q=1):p(p),q(q) {norm();}
  void norm(){
    int a = gcd(p,q);
    p/=a, q/=a;
    if(q < 0) q=-q, p=-p;}
  frac operator+(const frac& o){
    int a = gcd(q,o.q);
    return frac(add(mul(p,o.q/a), mul(o.p,q/a)), mul(q,o.q/a));}
  frac operator-(const frac& o){
    int a = gcd(q,o.q);
    return frac(sub(mul(p,o.q/a), mul(o.p,q/a)), mul(q,o.q/a));}
  frac operator*(frac o){
    int a = gcd(q,o.p), b = gcd(o.q,p);
    return frac(mul(p/b,o.p/a), mul(q/a,o.q/b));}
  frac operator/(frac o){
    int a = gcd(q,o.q), b = gcd(o.p,p);
    return frac(mul(p/b,o.q/a), mul(q/a,o.p/b));}
  bool operator<(const frac &o) const{return ll(p)*o.q < ll(o.p)*q;}
  bool operator==(frac o){return p==o.p && q==o.q;}
  bool operator!=(frac o){return p!=o.p || q!=o.q;}
};
```

7.23. Polinomio

```
struct poly {
    vector<tipo> c;//guarda los coeficientes del polinomio
    poly(const vector<tipo> &c): c(c) {}
    poly() {}
  bool isnull() {return c.empty();}
    poly operator+(const poly &o) const {
        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;
}
```

7.24. 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;
}
```

7.25. FFT y NTT

Base teórica

Dado el espacio lineal con producto interno (definido como una integral loca) E, de funciones continuas definidas por partes $f: [-\pi, \pi] \to \mathbb{C}$, un **sistema ortonormal cerrado infinito** es $\{1/\sqrt{(2)}, \sin(x), \cos(x), \sin(2x), \cos(2x), ...\}$. Por lo tanto, cualquier funcion $f \in E$ puede ser representada por $\sum_{n=1}^{\infty} \langle f,\ e_n \rangle e_n$. Esta combinación lineal (utilizando la sumatoria y el sistema ya definidos), es la **serie de Fourier**.

También se puede definir la **serie compleja de Fourier** mediante el sistema $\{1, e^{ix}, e^{-ix}, e^{i2x}, e^{-i2x}, ...\}$.

Una **transformada de Fourier** permite trabajar con funciones que no están restringidas al intervalo $[-\pi, \pi]$. La principal diferencia es que el sistema ortonormal pasa de ser discreto a continuo.

Sin embargo, existe una versión discreta de la transformada, la **transformada discreta de Fourier** (DFT).

Una de las propiedades importantes de la transformada es que la **convolución** de funciones sin transformar se traduce en multiplicar las transformadas.

FFT, el algoritmo para calcular rápidamente la DFT, se basa en que dado un polinomio $A(x)$, $A(x) = A_0(x^2) + x * A_1(x^2)$, donde $A_0(x)$ y $A_1(x)$ son los polinomios que se forman al tomar los términos pares e impares respectivamente.

NTT es un algoritmo más lento pero más preciso para calcular la DFT, ya que trabaja con enteros módulo un primo p.

```
// MODNTT-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
// const ll MODNTT = 2305843009255636993;
// const int RT = 5;
// struct for FFT, for NTT is simple (ll with mod operations)
struct CD {  // or typedef complex<double> CD; (but 4x slower)
  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); // FFT
CD cp1[MAXN+9],cp2[MAXN+9];  // MAXN must be power of 2 !!
int R[MAXN+9];
//CD root(int n, bool inv){ // NTT
//  ll r=pot(RT,(MODNTT-1)/n); // pot: modular exponentiation
```

```
//  return CD(inv?pot(r,MODNTT-2):r);
//}
void dft(CD* a, int n, bool inv){
  forn(i,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) forn(i,n)a[i]/=n; // FFT
  //if(inv){ // NTT
  //  CD z(pot(n,MODNTT-2)); // pot: modular exponentiation
  //  forn(i,n)a[i]=a[i]*z;
  //}
}
vi multiply(vi& p1, vi& p2){
  int n=si(p1)+si(p2)+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  forn(i,m){R[i]=0;forn(j,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
  forn(i,m)cp1[i]=0,cp2[i]=0;
  forn(i,si(p1))cp1[i]=p1[i];
  forn(i,si(p2))cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  forn(i,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
  vi res;
  n-=2;
  forn(i,n)res.pb((ll)floor(cp1[i].real()+0.5)); // change for NTT
  return res;
}
```

7.26. 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
$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$
Observación: Una buena aproximación es $x/ln(x)$.

Divisores
Cantidad de divisores (σ_0) para *algunos* $n / \neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$
Referencias: $\sigma_0(10^9) = 1344$ y $\sigma_0(10^{18}) = 103680$
$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$
Observación: Una buena aproximación es $x^{1/3}$.

Suma de divisores (σ_1) para *algunos* $n / \neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

8. Grafos

8.1. Teoremas y fórmulas

8.1.1. Teorema de Pick

$A = I + \frac{B}{2} - 1$
Donde A es el área, I es la cantidad de puntos interiores, y B la cantidad de puntos en el borde.

8.1.2. Formula de Euler

$v - e + f = k + 1$
Donde v es la cantidad de vértices, e la cantidad de arcos, f la cantidad de caras y k la cantidad de componentes conexas.

8.2. Dijkstra

```
vector<pii> adj[N]; // IMPORTANTE: ver tipo arco
//To add an edge (u,v) with cost p use G[u].pb(v,p)
ll dist[N];
int dad[N];
bool seen[N];

ll dijkstra(int s=0, int t=-1) {//O(|E| log |V|)
    fill(dist, dist+N, INF);
    fill(dad, dad+N, -1);
    fill(seen, seen+N, false);

  priority_queue<pii, vector<pii>, greater<pii>> pq;
  pq.emplace(0, s); dist[s] = 0;

  while (!pq.empty()){
    int u = pq.top().snd; pq.pop();

        if (seen[u]) continue;
        seen[u] = true;

    if (u == t) break;

    for (auto e : adj[u]) {
            int v, p; tie(v, p) = e;
      if (dist[u] + p < dist[v]) {
        dist[v] = dist[u] + p;
        dad[v] = u;
        pq.emplace(dist[v], v);
            }
        }
  }

  return t != -1 ? dist[t] : 0;
}
// path generator
if (dist[t] < INF)
    for (int u = t; u != -1; u = dad[u])
        cout << u << "␣\n"[u == s];
```

8.3. 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) for(auto u: G[j])
    dist[u.second]=min(dist[u.second], dist[j]+u.first);
}

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

8.4. 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;
}
```

8.5. Kruskal

```
struct Ar{int a,b,w;};
bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
vector<Ar> E;
ll kruskal(){
    ll cost=0;
    sort(E.begin(), E.end());//ordenar aristas de menor a mayor
    uf.init(n);
    forall(it, E){
        if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
            uf.unir(it->a, it->b);//conectar
            cost+=it->w;
        }
    }
    return cost;
}
```

8.6. 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;
}
```

8.7. 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];
```

```cpp
//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;
  for(auto u : G[v]){
    if(!idx[u] || cmp[u]==-2){
      if(!idx[u]) tjn(u);
      lw[v]=min(lw[v], lw[u]);
    }
  }
  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;
}
```

8.8. Kosaraju

```cpp
struct Kosaraju {
  static const int default_sz = 1e5+10;
  int n;
  vector<vi> G, revG, C, ady; // ady is the condensed graph
  vector<int> used, where;
  Kosaraju(int sz = default_sz){
    n = sz;
    G.assign(sz, vector<int>());
    revG.assign(sz, vector<int>());
    used.assign(sz, 0);
    where.assign(sz, -1);
  }
  void addEdge(int a, int b){ G[a].pb(b); revG[b].pb(a); }
  void dfsNormal(vector<int> &F, int v){
    used[v] = true;
    for (int u : G[v]) if(!used[u])
      dfsNormal(F, u);
    F.pb(v);
  }
  void dfsRev(vector<int> &F, int v){
    used[v] = true;
    for (int u : revG[v]) if(!used[u])
      dfsRev(F, u);
    F.pb(v);
  }
  void build(){
    vector<int> T;
    fill(all(used), 0);
    forn(i, n) if(!used[i]) dfsNormal(T, i);
    reverse(all(T));
    fill(all(used), 0);
    for (int u : T)
        if(!used[u]){
          vector<int> F;
          dfsRev(F, u);
          for (int v : F) where[v] = si(C);
          C.pb(F);
        }
    ady.resize(si(C)); // Create edges between condensed nodes
    forn(u, n) for(int v : G[u]){
      if(where[u] != where[v]){
        ady[where[u]].pb(where[v]);
      }
    }
    forn(v, si(C)){
      sort(all(ady[v]));
      ady[v].erase(unique(all(ady[v])), ady[v].end());
    }
  }
};
```

8.9. Articulation Points

```cpp
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;
  for(auto u: G[v])
    if(!V[u]){
      dfs(u, v);
      L[v] = min(L[v], L[u]);
      P[v]+= L[u]>=V[v];
    }
    else if(u!=f)
      L[v]=min(L[v], V[u]);
}
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;
}
```

8.10. Comp. Biconexas y Puentes

```cpp
struct bridge {

  struct edge {
    int u,v, comp;
    bool bridge;
  };

  int n;
  vi *adj;
  vector<edge> e;

  bridge(int n): n(n) {
    adj = new vi[n];
    e.clear();
    initDfs();
  }

  void initDfs() {
    d = new int[n];
    b = new int[n];
    comp = new int[n];
    nbc = t = 0;
        forn(u, n) d[u] = -1;
  }

  void addEdge(int u, int v) {
    adj[u].pb(si(e)); adj[v].pb(si(e));
    e.pb((edge){u,v,-1,false});
  }


    //d[i]=id de la dfs
    //b[i]=lowest id reachable from i
  int *d, *b, t, nbc;
  int *comp;
  stack<int> st;
  void dfs(int u=0, int pe=-1) {
    b[u] = d[u] = t++;
    comp[u] = (pe != -1);

    forn(i,si(adj[u])) {
      int ne = adj[u][i];
      if (ne == pe) continue;
      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]);
      }
    }
  }
};
```

8.11. LCA + Climb

```
const int MAXN=100001;
const int LOGN=20;
//f[v][k] holds the 2^k father of v
//L[v] holds the level of v
int N, f[MAXN][LOGN], L[MAXN]; //INICIALIZAR N!!!!!!!!!!!!!!1
//call before build:
void dfs(int v, int fa=-1, int lvl=0){//generate required data
  f[v][0]=fa, L[v]=lvl;
  for(auto u: G[v])if(u!=fa) dfs(u, v, lvl+1); }
void build(){//f[i][0] must be filled previously, O(nlgn)
  forn(k, LOGN-1) forn(i, N) if (f[i][k]!=-1) f[i][k+1]=f[f[i][k]][k];}
#define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
int climb(int a, int d){//O(lgn)
  if(!d) return a;
  dforsn(i,0, lg(L[a])+1) if(1<<i<=d) a=f[a][i], d-=1<<i;
    return a;}
int lca(int a, int b){//O(lgn)
  if(L[a]<L[b]) swap(a, b);
  a=climb(a, L[a]-L[b]);
  if(a==b) return a;
  dforsn(i,0, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
  return f[a][0]; }
int dist(int a, int b) {//returns distance between nodes
  return L[a]+L[b]-2*L[lca(a, b)];}
```

8.12. Heavy Light Decomposition

```
// Usa RMQ Dynamic
// ATENCION: valores en nodos. Ver comments para valores en arcos.
template <int V, class T>
class HeavyLight {
    int parent[V], heavy[V], depth[V];
    int root[V], treePos[V];
    RMQ<V, T, T> tree;

    template <class G>
    int dfs(const G& graph, int v) {
        int size = 1, maxSubtree = 0;
        for (int u : graph[v]) if (u != parent[v]) {
            parent[u] = v;
            depth[u] = depth[v] + 1;
            int subtree = dfs(graph, u);
            if (subtree > maxSubtree) heavy[v] = u, maxSubtree =
                subtree;
            size += subtree;
        }
        return size;
    }

    template <class BinaryOperation>
    void processPath(int u, int v, BinaryOperation op) {
        for (; root[u] != root[v]; v = parent[root[v]]) {
            if (depth[root[u]] > depth[root[v]]) swap(u, v);
            op(treePos[root[v]], treePos[v] + 1);
        }
        if (depth[u] > depth[v]) swap(u, v);
        // ATENCION: para valores almacenados en arcos: cambiar por
            op(treePos[u]+1, treePos[v]+1)
        op(treePos[u], treePos[v] + 1);
    }

public:
// ATENCION: grafo como vector<vector<int>>
template <class G>
    void init(const G& graph) {
        int n = si(graph);
        fill_n(heavy, n, -1);
        parent[0] = -1;
        depth[0] = 0;
        dfs(graph, 0);
        for (int i = 0, currentPos = 0; i < n; ++i)
            if (parent[i] == -1 || heavy[parent[i]] != i)
                for (int j = i; j != -1; j = heavy[j]) {
                    root[j] = i;
                    treePos[j] = currentPos++;
                }
        tree.init(n);
    }
```

```
50
51      void set(int v, const T& value) {
52          tree.modify(treePos[v], treePos[v]+1, value);
53      }
54
55      void modifyPath(int u, int v, const T& value) {
56          processPath(u, v, [this, &value](int l, int r) { tree.modify(
                value, l, r); });
57      }
58
59      T queryPath(int u, int v) {
60          T res = T();
61          processPath(u, v, [this, &res](int l, int r) { res += tree.get(l
                , r); });
62          return res;
63      }
64  };
```

8.13. Centroid Decomposition

```
1   vector<int> G[MAXN];
2   bool taken[MAXN];//poner todos en FALSE al principio!!
3   int padre[MAXN];//padre de cada nodo en el centroid tree
4   int szt[MAXN];
5   void calcsz(int v, int p) {
6     szt[v] = 1;
7     for (int u : G[v]) if (u!=p && !taken[u])
8       calcsz(u,v), szt[v]+=szt[u];
9   }
10  void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
11    if(tam==-1) calcsz(v, -1), tam=szt[v];
12    for(int u : G[v]) if(!taken[u] && szt[u]>=tam/2)
13      {szt[v]=0; centroid(u, f, lvl, tam); return;}
14    taken[v]=true;
15    padre[v]=f;
16    for(int u : G[v]) if(!taken[u])
17      centroid(u, v, lvl+1, -1);
18  }
```

8.14. Euler Cycle

```
1   int n,m,ars[MAXE], eq;
2   vector<int> G[MAXN];//fill G,n,m,ars,eq
3   list<int> path;
4   int used[MAXN];
5   bool usede[MAXE];
6   queue<list<int>::iterator> q;
7   int get(int v){
8     while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
9     return used[v];
10  }
11  void explore(int v, int r, list<int>::iterator it){
12    int ar=G[v][get(v)]; int u=v^ars[ar];
13    usede[ar]=true;
14    list<int>::iterator it2=path.insert(it, u);
15    if(u!=r) explore(u, r, it2);
16    if(get(v)<sz(G[v])) q.push(it);
17  }
18  void euler(){
19    zero(used), zero(usede);
20    path.clear();
21    q=queue<list<int>::iterator>();
22    path.push_back(0); q.push(path.begin());
23    while(sz(q)){
24      list<int>::iterator it=q.front(); q.pop();
25      if(used[*it]<sz(G[*it])) explore(*it, *it, it);
26    }
27    reverse(path.begin(), path.end());
28  }
29  void addEdge(int u, int v){
30    G[u].pb(eq), G[v].pb(eq);
31    ars[eq++]=u^v;
32  }
```

8.15. Diametro árbol

```
1   int n;
2   vi adj[N];
3
4   pii farthest(int u, int p = -1) {
5       pii ans = {-1, u};
6
7       for (int v : adj[u])
8           if (v != p)
9               ans = max(ans, farthest(v, u));
10
11      ans.fst++;
```

```
    return ans;
}

int diam(int r) {
    return farthest(farthest(r).snd).fst;
}

bool path(int s, int e, vi &p, int pre = -1) {
    p.pb(s);
    if (s == e) return true;

    for (int v : adj[s])
        if (v != pre && path(v, e, p, s))
            return true;

    p.pop_back();
    return false;
}

int center(int r) {
    int s = farthest(r).snd, e = farthest(s).snd;
    vi p; path(s, e, p);
    return p[si(p)/2];
}
```

8.16. 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;
    }
  }
}
```

8.17. 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;
}
```

8.18. 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;
```

9. Flujo

9.1. Dinic

```
// 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
// MAXN Independent Set: tomar los vertices NO tomados por el Min Vertex Cover
// MAXN Clique: construir la red de G complemento (debe ser bipartito!) y encontrar un MAXN Independet Set
// Min Edge Cover: tomar las aristas del matching + para todo vertices no cubierto hasta el momento, tomar cualquier arista de el

// Tiempos! O(V^2*E) en general. O(sqrt(V)*E) en matching bipartito. O(min(E^(2/3), V^(1/2)*E) si capacidad 1.
template<int MAXN>
struct dinic {

    struct edge {
        int u,v; ll c,f;
        ll r() { return c-f; }
    };

    static const ll INF = 1e18;

    int N,S,T;
    vector<edge> e;
    //edge red[MAXN][MAXN];
    vi adjG[MAXN];

    void reset() {
        forn(u,N) for (auto ind : adjG[u]) {
            auto &ei = e[ind];
            ei.f = 0;
```

```
        }
    }

    void initGraph(int n, int s, int t) {
        N = n; S = s; T = t;
        e.clear();
        forn(u,N) adjG[u].clear();
    }

    void addEdge(int u, int v, ll c) {
        adjG[u].pb(si(e)); e.pb((edge){u,v,c,0});
        adjG[v].pb(si(e)); e.pb((edge){v,u,0,0});
    }

    int dist[MAXN];
    bool dinic_bfs() {
        forn(u,N) dist[u] = -1;
        queue<int> q; q.push(S); dist[S] = 0;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto ind : adjG[u]) {
                auto &ei = e[ind];
                int v = ei.v;
                if (dist[v] != -1 || ei.r() == 0) continue;
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
        return dist[T] != -1;
    }

    ll dinic_dfs(int u, ll cap) {
        if (u == T) return cap;

        ll res = 0;
        for (auto ind : adjG[u]) {
            auto &ei = e[ind], &ej = e[ind^1];
            int v = ei.v;
            if (ei.r() && dist[v] == dist[u] + 1) {
                ll send = dinic_dfs(v,min(cap, ei.r()));
                ei.f += send; ej.f -= send;
                res += send; cap -= send;
                if (cap == 0) break;
            }
        }
        if (res == 0) dist[u] = -1;
        return res;
    }

    ll flow() {
        ll res = 0;
        while (dinic_bfs()) res += dinic_dfs(S,INF);
        return res;
    }

    vi cut() {
        dinic_bfs();
        vi ans;
        for (auto u : adjG[S]) if (dist[e[u].v] == -1) ans.pb(e[u].v);
        for (auto u : adjG[T]) if (dist[e[u].v] != -1) ans.pb(e[u].v);
        return ans;
    }

    vi indep() {
        dinic_bfs();
        vi ans;
        for (auto u : adjG[S]) if (dist[e[u].v] != -1) ans.pb(e[u].v);
        for (auto u : adjG[T]) if (dist[e[u].v] == -1) ans.pb(e[u].v);
        return ans;
    }
};
```

9.2. 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);
      }
    }
  }
}
```

9.3. Edmonds Karp's

```
#define MAX_V 1000
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto
//To add an edge use
#define add(a, b, w) G[a][b]=w
int f, p[MAX_V];
void augment(int v, int minE){
  if(v==SRC) f=minE;
  else if(p[v]!=-1){
    augment(p[v], min(minE, G[p[v]][v]));
    G[p[v]][v]-=f, G[v][p[v]]+=f;
  }
}
ll maxflow(){//O(VE^2)
  ll Mf=0;
  do{
    f=0;
    char used[MAX_V]; queue<int> q; q.push(SRC);
    zero(used), memset(p, -1, sizeof(p));
    while(sz(q)){
      int u=q.front(); q.pop();
      if(u==SNK) break;
      forall(it, G[u])
        if(it->snd>0 && !used[it->fst])
          used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
    }
    augment(SNK, INF);
    Mf+=f;
  }while(f);
  return Mf;
}
```

9.4. 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];
    }
  }
}
```

10. Template

```
#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
  #define D(a) cerr << #a << " = " << a << endl
#else
  #define D(a)
  #define cerr false && cerr
#endif
#define fastio ios_base::sync_with_stdio(0); cin.tie(0)
#define dforsn(i,s,n) for(int i=int(n-1);i>=int(s);i--)
#define forsn(i,s,n) for(int i=int(s);i<int(n);i++)
#define dforn(i,n) dforsn(i,0,n)
#define forn(i,n) forsn(i,0,n)
#define all(a) a.begin(),a.end()
#define si(a) int((a).size())
#define pb emplace_back
#define mp make_pair
#define snd second
#define fst first
#define endl '\n'
using pii = pair<int,int>;
using vi = vector<int>;
using ll = long long;

int main() {
  fastio;


  return 0;
}
```

11. Template hash

```
cat template.cpp | tr -d '\0'-'␣' | md5sum
output: 4a1c2d274a716d2cce43340df6b18112
```

12. vimrc

```
colo desert
set number
set norelativenumber
set autochdir
set colorcolumn=80
set ignorecase
set showcmd
augroup cpp
    autocmd!
    autocmd FileType cpp map <f9> :w<enter> :!g++ -std=c++14 -W -Wall -
        Wshadow -Wconversion -DLOCAL -D_GLIBCXX_DEBUG -g3 "%" -o "a" <
        enter>
    autocmd FileType cpp map <f5> :!"./a" < a.in <enter>
    autocmd FileType cpp map <f6> :!"./a" <enter>
augroup END
set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab
set smartindent
set cindent
set clipboard=unnamedplus
nmap <c-h> <c-w><c-h>
nmap <c-j> <c-w><c-j>
```

```
nmap <c-k> <c-w><c-k>
nmap <c-l> <c-w><c-l>
vmap > >gv
vmap < <gv
map j gj
map k gk
nnoremap <silent> [b :bp<CR>
nnoremap <silent> ]b :bn<CR>
nnoremap <silent> [B :bf<CR>
nnoremap <silent> ]B :bl<CR>
set splitright
set nobackup
set nowritebackup
set noswapfile
```

13. misc

```
#include <bits/stdc++.h> // Library that includes the most used libraries
using namespace std;  // It avoids the use of std::func(), instead we can simply use func()

ios_base::sync_with_stdio(0); cin.tie(0); // Speeds up considerably the read speed, very convenient when the input is large

#pragma GCC optimize ("O3") // Asks the compiler to apply more optimizations, that way speeding up the program very much!

Math:
max(a,b);  // Returns the largest of a and b
min(a,b);  // Returns the smallest of a and b
abs(a,b);  // Returns the absolute value of x (integral value)
fabs(a,b); // Returns the absolute value of x (double)
sqrt(x);   // Returns the square root of x.
pow(base,exp); // Returns base raised to the power exp
ceil(x); // Rounds x upward, returning the smallest integral value that is not less than x
floor(x); // Rounds x downward, returning the largest integral value that is not greater than x
exp(x); // Returns the base-e exponential function of x, which is e raised to the power x
log(x); // Returns the natural logarithm of x
log2(x); // Returns the binary (base-2) logarithm of x
log10(x); // Returns the common (base-10) logarithm of x
modf(double x, double *intpart); /* Breaks x into an integral and a fractional part. The integer part is stored in the object
pointed by intpart, and the fractional part is returned by the function. Both parts have the same sign as x. */
sin(),cos(),tan(); asin(),acos(),atan(); sinh(),cosh(),tanh(); // Trigonometric functions
// See http://www.cplusplus.com/reference/cmath/ for more useful math functions!

Strings:
s.replace(pos,len,str); // Replaces the portion of the string that begins at character pos and spans len characters by str
s.replace(start,end,str); // or the part of the string in the range between [start,end)
s.substr(pos = 0,len = npos); // Returns the substring starting at character pos that spans len characters (or until the end of the string, whichever comes first).
// A value of string::npos indicates all characters until the end of the string.
s.insert(pos,str); // Inserts str right before the character indicated by pos
s.erase(pos = 0, len = npos); erase(first,last); erase(iterator p); // Erases part of the string
s.find(str,pos = 0); // Searches the string for the first occurrence of the sequence specified by its arguments after position pos
toupper(char x); // Converts lowercase letter to uppercase. If no such conversion is possible, the value returned is x unchanged.
tolower(char x); // Converts uppercase letter to lowercase. If no such conversion is possible, the value returned is x unchanged.

Constants:
INT_MAX, INT_MIN, LLONG_MIN, LLONG_MAX, ULLONG_MAX
const int maxn = 1e5; // 1e5 means 1x10^5, C++ features scientific notation. e.g.: 4.56e6 = 4.560.000, 7.67e-5 = 0.0000767.
const double pi = acos(-1); // Compute Pi

Algorithms:
swap(a,b); // Exchanges the values of a and b
minmax(a,b); // Returns a pair with the smallest of a and b as first element, and the largest as second.
minmax({1,2,3,4,5}); // Returns a pair with the smallest of all the elements in the list as first element and the largest as second
```

```
next_permutation(a,a+n); // Rearranges the elements in the range [first, last) into the next lexicographically greater permutation.
reverse(first,last); // Reverses the order of the elements in the range [first,last)
rotate(first,middle,last) // Rotates the order of the elements in the range [first,last), in such a way that the element pointed by middle becomes the new first element
remove_if(first,last,func) // Returns an iterator to the element that follows the last element not removed. The range between first and this iterator includes all the elements in the sequence for which func does not return true.
// See http://www.cplusplus.com/reference/algorithm/ for more useful algorithms!

Binary search:
int a[] = {1, 2, 4, 7, 10, 12}, x = 5;
int *l = lower_bound(a,a+6,x); // lower_bound: Returns the first element that is not less than x
cout << (l == a+5 ? -1 : *l) << endl;
cout << x << (binary_search(a,a+6,x)?"␣is\n":"␣isn't\n"); // binary_search: Returns true if any element in the range [first,last) is equivalent to x, and false otherwise.
vi v(a,a+6);
auto i = upper_bound(v.begin(),v.end(),x) // upper_bound: Returns the first element that is greater than x

Random numbers:
mt19937_64 rng(time(0)); //if TLE use 32 bits: mt19937
ll rnd(ll a, ll b) { return a + rng()%(b-a+1); }
Unhackable seed (Codeforces):
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
random_shuffle(a,a+n,rng); // Rearranges the elements in the range [first,last) randomly

Sorting:
sort(a,a+n,comp); /* Sorts the elements in the range [first,last) into ascending order.
The third parameter is optional, if greater<Type> is passed then the array is sorted in descending order.
comp: Binary function that accepts two elements in the range as arguments, and returns a value convertible to bool. The value returned
indicates whether the element passed as first argument is considered to go before the second in the specific strict weak ordering
it defines. The function shall not modify any of its arguments. This can either be a function pointer or a function object. */
stable_sort(a,a+n); // Sorts the elements in the range [first,last) into ascending order, like sort, but stable_sort preserves the relative order of the elements with equivalent values.
sort(a.begin(),a.end()); // Sort using container ranges
sort(a,a+n,[](const node &a, const node &b){ // Custom sort with a "lambda expression": an unnamed function object capable of capturing variables in scope.
  return a.x < b.x || (a.x == b.x && a.y < b.y);   // Custom sort
}); // see https://en.cppreference.com/w/cpp/language/lambda for more details
bool myfunction(const edge &a, const edge &b){ return a.w < b.w; }
sort(myvector.begin()+4, myvector.end(), myfunction); // Using a function as a comparator
struct comp{ bool operator()(const edge &a, const edge &b){ return a.w < b.w; } };
multiset<edge,comp> l; // Using a function object as comparator:
bool operator<(const edge &a, const edge &b){ return a.w < b.w; } // Operator definition (it can be inside or outside the class)

Input/output handling:
freopen("input.txt","r",stdin); // Sets the standard input stream (keyboard) to the file input.txt
freopen("output.txt","w",stdout); // Sets the standard output stream (screen) to the file output.txt
getline(cin,str); // Reads until an end of line is reached from the input stream into str. If we use cin >> str it would read until it finds a whitespace
// Make an extra call if we previously read another thing from the input stream (otherwise it wouldn't work as expected)
cout << fixed << setprecision(n); // Sets the decimal precision to be used to format floating-point values on output operations to n
cout << setw(n); // Sets the field width to be used on output operations to n
cout << setfill('0'); // Sets c as the stream's fill character

Increment stack size to the maximum (Linux):
// #include <sys/resource.h>
struct rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur = rl.rlim_max;
```

```
setrlimit(RLIMIT_STACK, &rl);

String to int and vice versa (might be very useful to parse odd things):
template <typename T> string to_str(T str) { stringstream s; s << str;
    return s.str(); }
template <typename T> int to_int(T n) { int r; stringstream s; s << n; s
     >> r; return r; }
C++11:
to_string(num) // returns a string with the representation of num
stoi,stoll,stod,stold // string to int,ll,double & long double
    respectively

Print structs with cout:
ostream& operator << (ostream &o, pto &p) {
    o << p.x << ' ' << p.y;
    return o;
}
```

14. Ayudamemoria

Cant. decimales

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

Rellenar con espacios(para justificar)

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

Comparación de Doubles

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

Limites

```
#include <limits>
numeric_limits<T>
  ::max()
  ::min()
  ::epsilon()
```

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 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);
}
```

Leer del teclado

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

Iterar subconjunto

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

File setup

```
// tambien se pueden usar comas: {a, x, m, l}
touch {a..l}.in; tee {a..l}.cpp < template.cpp
```

Releer String

```
string s; int n;
getline(cin, s);
stringstream leer(s);
while(leer >> n){
  // do something ...
}
```