Índice

1. algorithm

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element (median of medians)	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	$T = i + [f1, l1) \cdot [f2, \dots)$
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: LVL $\geq$ ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
1   struct RMQ{
2     #define LVL 10
3     tipo vec[LVL][1<<(LVL+1)];
4     tipo &operator[](int p){return vec[0][p];}
5     tipo get(int i, int j) {//intervalo [i,j)
6       int p = 31-__builtin_clz(j-i);
7       return min(vec[p][i],vec[p][j-(1<<p)]);
8     }
9     void build(int n) {//O(nlogn)
10      int mp = 31-__builtin_clz(n);
11      forn(p, mp) forn(x, n-(1<<p))
12        vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
13  }};
```

2.2. RMQ (dynamic)

```
1   //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
        sobre el rango [i, j).
2   #define MAXN 100000
3   #define operacion(x, y) max(x, y)
4   const int neutro=0;
5   struct RMQ{
6     int sz;
7     tipo t[4*MAXN];
8     tipo &operator[](int p){return t[sz+p];}
9     void init(int n){//O(nlgn)
10      sz = 1 << (32-__builtin_clz(n));
11      forn(i, 2*sz) t[i]=neutro;
12    }
13    void updall(){//O(n)
14      dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
15    tipo get(int i, int j){return get(i,j,1,0,sz);} // [i,j) !
16    tipo get(int i, int j, int n, int a, int b){//O(lgn)
17      if(j<=a || i>=b) return neutro;
18      if(i<=a && b<=j) return t[n];
19      int c=(a+b)/2;
```

```
20      return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
21    }
22    void set(int p, tipo val){//O(lgn)
23      for(p+=sz; p>0 && t[p]!=val;){
24        t[p]=val;
25        p/=2;
26        val=operacion(t[p*2], t[p*2+1]);
27      }
28    }
29  }rmq;
30  //Usage:
31  cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3. RMQ (lazy)

```
1  //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
       sobre el rango [i, j).
2  typedef int Elem;//Elem de los elementos del arreglo
3  typedef int Alt;//Elem de la alteracion
4  #define operacion(x,y) x+y
5  const Elem neutro=0; const Alt neutro2=0;
6  #define MAXN 1024000
7  struct RMQ{
8    int sz;
9    Elem t[4*MAXN];
10   Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem
11   Elem &operator[](int p){return t[sz+p];}
12   void init(int n){//O(nlgn)
13     sz = 1 << (32-__builtin_clz(n));
14     forn(i, 2*sz) t[i]=neutro;
15     forn(i, 2*sz) dirty[i]=neutro2;
16   }
17   void updall(){//O(n)
18     dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
19   void opAltT(int n,int a,int b){//altera el valor del nodo n segun su
          dirty y el intervalo que le corresponde.
20     t[n] += dirty[n]*(b-a); //en este caso la alteracion seria sumarle a
              todos los elementos del intervalo [a,b) el valor dirty[n]
21   }
22   void opAltD(int n ,Alt val){
23     dirty[n]+= val;
24   }//actualiza el valor de Dirty "sumandole" val. podria cambiar el valor
          de dirty dependiendo de la operacion que se quiera al actualizar
       un rango. Ej:11402.cpp
25   void push(int n, int a, int b){//propaga el dirty a sus hijos
26     if(dirty[n]!=neutro2){
27       //t[n]+=dirty[n]*(b-a);//altera el nodo
28       opAltT(n,a,b);
29       if(n<sz){
30         opAltD(2*n,dirty[n]);//dirty[2*n]+=dirty[n];
31         opAltD(2*n+1,dirty[n]);//dirty[2*n+1]+=dirty[n];
32       }
33       dirty[n]=neutro2;
34     }
35   }
36   Elem get(int i, int j, int n, int a, int b){//O(lgn)
37     if(j<=a || i>=b) return neutro;
38     push(n, a, b);//corrige el valor antes de usarlo
39     if(i<=a && b<=j) return t[n];
40     int c=(a+b)/2;
41     return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
42   }
43   Elem get(int i, int j){return get(i,j,1,0,sz);}
44   //altera los valores en [i, j) con una alteracion de val
45   void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
46     push(n, a, b);
47     if(j<=a || i>=b) return;
48     if(i<=a && b<=j){
49       opAltD(n ,val);//actualiza el valor de Dirty por val.
50       push(n, a, b);
51       return;//este nodo esta totalmente contenido por el intervalo a
              alterar, no es necesario que se lo pases a los hijos.. por ahora
              ..
52     }
53     int c=(a+b)/2;
54     alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
55     t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
56   }
57   void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
58 };
```

2.4. RMQ (persistente)

```
1  typedef int tipo;
2  tipo oper(const tipo &a, const tipo &b){
3    return a+b;
```

```
4  }
5  struct node{
6    tipo v; node *l,*r;
7    node(tipo v):v(v), l(NULL), r(NULL) {}
8      node(node *l, node *r) : l(l), r(r){
9          if(!l) v=r->v;
10         else if(!r) v=l->v;
11         else v=oper(l->v, r->v);
12     }
13 };
14 node *build (tipo *a, int tl, int tr) {//modificar para que tome tipo a
15   if (tl+1==tr) return new node(a[tl]);
16   int tm=(tl + tr)>>1;
17   return new node(build(a, tl, tm), build(a, tm, tr));
18 }
19 node *update(int pos, int new_val, node *t, int tl, int tr){
20   if (tl+1==tr) return new node(new_val);
21   int tm=(tl+tr)>>1;
22   if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
23   else return new node(t->l, update(pos, new_val, t->r, tm, tr));
24 }
25 tipo get(int l, int r, node *t, int tl, int tr){
26     if(l==tl && tr==r) return t->v;
27   int tm=(tl + tr)>>1;
28     if(r<=tm) return get(l, r, t->l, tl, tm);
29     else if(l>=tm) return get(l, r, t->r, tm, tr);
30   return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
31 }
```

2.5. Fenwick Tree

```
1  //For 2D threat each column as a Fenwick tree, by adding a nested for in
       each operation
2  struct Fenwick{
3    int sz; //los elementos van de 1 a sz-1
4    tipo t[MAXN][MAXN];
5    void init (int n){
6      sz = n;
7      forn(i,MAXN) forn(j,MAXN) t[i][j] = 0;
8    }
9    //le suma v al valor de (p,q)
10   void adjust(int p, int q, tipo v){//valid with p in [1, sz), q in [1,sz)
          --> O(lgn*lgn)
11     for(int i=p; i<sz; i+=(i&-i))
12       for(int j=q; j<sz; j+=(j&-j))
13         t[i][j]+=v; }
14   tipo sum(int p, int q){//cumulative sum in [(1,1), (p,q)], O(lgn*lgn) --
          OJO: los rangos son cerrados!
15     tipo s=0;
16     for(int i=p; i; i-=(i&-i)) for(int j=q; j; j-=(j&-j)) s+=t[i][j];
17     return s;
18   }
19   tipo sum(int a1, int b1, int a2, int b2){return sum(a2,b2)-sum(a1-1,b2) -
          sum(a2,b1-1) + sum(a1-1,b1-1);}
20   //get largest value with cumulative sum less than or equal to x;
21   //for smallest, pass x-1 and add 1 to result
22   int getind(tipo x) {//O(lgn) -- VER!
23       int idx = 0, mask = N;
24       while(mask && idx < N) {
25         int t = idx + mask;
26       if(x >= tree[t])
27           idx = t, x -= tree[t];
28         mask >>= 1;
29       }
30       return idx;
31   }} f;
```

2.6. Union Find

```
1  struct UnionFind{
2    vector<int> f;//the array contains the parent of each node
3    void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
4    int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));}//O(1)
5    bool join(int i, int j) {
6      bool con=comp(i)==comp(j);
7      if(!con) f[comp(i)] = comp(j); //pa no romper la super complejidad
8      return con;
9    }};
```

2.7. Disjoint Intervals

```
1  bool operator< (const ii &a, const ii &b) {return a.fst<b.fst;}
2  //Stores intervals as [first, second]
3  //in case of a collision it joins them in a single interval
4  struct disjoint_intervals {
5    set<ii> segs;
6    void insert(ii v) {//O(lgn)
```

```
7     if(v.snd-v.fst==0.) return;//OJO
8     set<ii>::iterator it,at;
9     at = it = segs.lower_bound(v);
10    if (at!=segs.begin() && (--at)->snd >= v.fst)
11      v.fst = at->fst, --it;
12    for(; it!=segs.end() && it->fst <= v.snd; segs.erase(it++))
13      v.snd=max(v.snd, it->snd);
14    segs.insert(v);
15  }
16 };
```

2.8. RMQ (2D)

```
1  struct RMQ2D{//n filas x m columnas
2    int sz;
3    RMQ t[4*MAXN];
4    void init(int n, int m){//O(n*m)
5      sz = 1 << (32-__builtin_clz(n));
6      forn(i, 2*sz) t[i].init(m); }
7    void set(int i, int j, tipo val){//O(lgm.lgn)
8      for(i+=sz; i>0;){
9        t[i].set(j, val);
10       i/=2;
11       val=operacion(t[i*2][j], t[i*2+1][j]);
12     } }
13   tipo get(int i1, int j1, int i2, int j2){return get(i1,j1,i2,j2,1,0,sz);}
14   //O(lgm.lgn), rangos cerrado abierto
15   int get(int i1, int j1, int i2, int j2, int n, int a, int b){
16     if(i2<=a || i1>=b) return neutro;
17     if(i1<=a && b<=i2) return t[n].get(j1, j2);
18     int c=(a+b)/2;
19     return operacion(get(i1, j1, i2, j2, 2*n, a, c),
20         get(i1, j1, i2, j2, 2*n+1, c, b));
21   }
22 } rmq;
23 //Example to initialize a grid of M rows and N columns:
24 RMQ2D rmq; rmq.init(n,m);
25 forn(i, n) forn(j, m){
26   int v; cin >> v; rmq.set(i, j, v);}
```

2.9. Big Int

```
1 #define BASEXP 6
2 #define BASE 1000000
```

```
3  #define LMAX 1000
4  struct bint{
5    int l;
6    ll n[LMAX];
7    bint(ll x=0){
8      l=1;
9      forn(i, LMAX){
10         if (x) l=i+1;
11         n[i]=x%BASE;
12         x/=BASE;
13
14     }
15   }
16   bint(string x){
17   l=(x.size()-1)/BASEXP+1;
18     fill(n, n+LMAX, 0);
19     ll r=1;
20     forn(i, sz(x)){
21         n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
22         r*=10; if(r==BASE)r=1;
23     }
24   }
25   void out(){
26   cout << n[l-1];
27   dforn(i, l-1) printf("%6.6llu", n[i]);//6=BASEXP!
28   }
29   void invar(){
30     fill(n+l, n+LMAX, 0);
31     while(l>1 && !n[l-1]) l--;
32   }
33 };
34 bint operator+(const bint&a, const bint&b){
35   bint c;
36     c.l = max(a.l, b.l);
37     ll q = 0;
38     forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
39     if(q) c.n[c.l++] = q;
40     c.invar();
41     return c;
42 }
43 pair<bint, bool> lresta(const bint& a, const bint& b)    // c = a - b
44 {
45   bint c;
```

```
46      c.l = max(a.l, b.l);
47      ll q = 0;
48      forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/BASE
            -1;
49      c.invar();
50      return make_pair(c, !q);
51  }
52  bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).first;}
53  bint operator- (const bint&a, const bint&b){return lresta(a, b).first;}
54  bool operator< (const bint&a, const bint&b){return !lresta(a, b).second;}
55  bool operator<= (const bint&a, const bint&b){return lresta(b, a).second;}
56  bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
57  bint operator*(const bint&a, ll b){
58      bint c;
59      ll q = 0;
60      forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
61      c.l = a.l;
62      while(q) c.n[c.l++] = q %BASE, q/=BASE;
63      c.invar();
64      return c;
65  }
66  bint operator*(const bint&a, const bint&b){
67      bint c;
68      c.l = a.l+b.l;
69      fill(c.n, c.n+b.l, 0);
70      forn(i, a.l){
71          ll q = 0;
72          forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q/=
                BASE;
73          c.n[i+b.l] = q;
74      }
75      c.invar();
76      return c;
77  }
78  pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
79    bint c;
80    ll rm = 0;
81    dforn(i, a.l){
82          rm = rm * BASE + a.n[i];
83          c.n[i] = rm / b;
84          rm %= b;
85      }
86      c.l = a.l;
87      c.invar();
88      return make_pair(c, rm);
89  }
90  bint operator/(const bint&a, ll b){return ldiv(a, b).first;}
91  ll operator%(const bint&a, ll b){return ldiv(a, b).second;}
92  pair<bint, bint> ldiv(const bint& a, const bint& b){
93    bint c;
94      bint rm = 0;
95      dforn(i, a.l){
96          if (rm.l==1 && !rm.n[0])
97              rm.n[0] = a.n[i];
98          else{
99              dforn(j, rm.l) rm.n[j+1] = rm.n[j];
100             rm.n[0] = a.n[i];
101             rm.l++;
102         }
103         ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
104         ll u = q / (b.n[b.l-1] + 1);
105         ll v = q /  b.n[b.l-1] + 1;
106         while (u < v-1){
107             ll m = (u+v)/2;
108             if (b*m <= rm) u = m;
109             else v = m;
110         }
111         c.n[i]=u;
112         rm-=b*u;
113     }
114   c.l=a.l;
115     c.invar();
116     return make_pair(c, rm);
117 }
118 bint operator/(const bint&a, const bint&b){return ldiv(a, b).first;}
119 bint operator%(const bint&a, const bint&b){return ldiv(a, b).second;}
```

2.10. HashTables

```
1  //Compilar: g++ --std=c++11
2  struct Hash{
3    size_t operator()(const ii &a)const{
4      size_t s=hash<int>()(a.fst);
5      return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
6    }
7    size_t operator()(const vector<int> &v)const{
```

```cpp
    size_t s=0;
    for(auto &e : v)
      s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
    return s;
  }
};
unordered_set<ii, Hash> s;
unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

2.11.   Modnum

```cpp
//lindos valores para hash
#define MOD 1000000000000000009
#define PRIME 1009

ll mul(ll a, ll b, ll m) { //hace (a*b)%m
  ll q = (ll)((long double)a*b/m);
  ll r = a*b-m*q;
  while(r<0) r += m;
  while(r>=m) r -= m;
  return r;
}

struct mnum{
  static const tipo mod=MOD;
  tipo v;
  mnum(tipo v=0): v(v%mod) {}
  mnum operator+(mnum b){return v+b.v;}
  mnum operator-(mnum b){return ((v-b.v)%mod)+mod;}
  //mnum operator*(mnum b){return v*b.v;} //Si mod<=1e9+9
  mnum operator*(mnum b){return mul(v,b.v,mod);} //Si mod<=1e18+9
  mnum operator^(int n){
    if(!n) return 1;
    return n%2 ? ((*this)^(n/2))*(*this) : (*this)^(n/2);}
};

/*
DIVISIÓN MODULAR
Para dividir hay que multiplicar por el inverso multiplicativo. x/y = x*(y
    ^-1).
El inverso multiplicativo de y módulo n es y^-1 tal que y*(y^-1) = 1 mod n.
Por ejemplo, si n=7, y=2, o sea que quiero dividir por y,
y^-1 = 4 porque y*(y^-1) = 8 = 1 mod 7.
```

```cpp
*/
```

2.12.   Treap para set

```cpp
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);
```

```
40      else if (key < t->key) erase(t->l, key);
41      else erase(t->r, key);
42      if(t) pull(t);
43  }
44
45  ostream& operator<<(ostream &out, const pnode &t) {
46    if(!t) return out;
47      return out << t->l << t->key << '␣' << t->r;
48  }
49  pnode find(pnode t, Key key) {
50      if (!t) return 0;
51      if (key == t->key) return t;
52      if (key < t->key) return find(t->l, key);
53      return find(t->r, key);
54  }
55  struct treap {
56      pnode root;
57      treap(pnode root=0): root(root) {}
58      int size() { return ::size(root); }
59      void insert(Key key) {
60          pnode t1, t2; split(root, key, t1, t2);
61          t1=::merge(t1,new node(key));
62          root=::merge(t1,t2);
63      }
64      void erase(Key key1, Key key2) {
65          pnode t1,t2,t3;
66          split(root,key1,t1,t2);
67          split(t2,key2, t2, t3);
68          root=merge(t1,t3);
69      }
70      void erase(Key key) {::erase(root, key);}
71      pnode find(Key key) { return ::find(root, key); }
72      Key &operator[](int pos){return find(pos)->key;}//ojito
73  };
74  treap merge(treap a, treap b) {return treap(merge(a.root, b.root));}
```

2.13. Treap para arreglo

```
1   typedef struct node *pnode;
2   struct node{
3       Value val, mini;
4       int dirty;
5       int prior, size;
6       pnode l,r,parent;
7       node(Value val): val(val), mini(val), dirty(0), prior(rand()), size(1),
8           l(0), r(0), parent(0) {}
9   };
10  static int size(pnode p) { return p ? p->size : 0; }
11  void push(pnode p) {//propagar dirty a los hijos(aca para lazy)
12    p->val.fst+=p->dirty;
13    p->mini.fst+=p->dirty;
14    if(p->l) p->l->dirty+=p->dirty;
15    if(p->r) p->r->dirty+=p->dirty;
16    p->dirty=0;
17  }
18  static Value mini(pnode p) { return p ? push(p), p->mini : ii(1e9, -1); }
19  // Update function and size from children's Value
20  void pull(pnode p) {//recalcular valor del nodo aca (para rmq)
21    p->size = 1 + size(p->l) + size(p->r);
22    p->mini = min(min(p->val, mini(p->l)), mini(p->r));//operacion del rmq!
23    p->parent=0;
24    if(p->l) p->l->parent=p;
25    if(p->r) p->r->parent=p;
26  }
27  //junta dos arreglos
28  pnode merge(pnode l, pnode r) {
29    if (!l || !r) return l ? l : r;
30    push(l), push(r);
31    pnode t;
32    if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
33    else r->l=merge(l, r->l), t = r;
34    pull(t);
35    return t;
36  }
37  //parte el arreglo en dos, sz(l)==tam
38  void split(pnode t, int tam, pnode &l, pnode &r) {
39    if (!t) return void(l = r = 0);
40    push(t);
41    if (tam <= size(t->l)) split(t->l, tam, l, t->l), r = t;
42    else split(t->r, tam - 1 - size(t->l), t->r, r), l = t;
43    pull(t);
44  }
45  pnode at(pnode t, int pos) {
46    if(!t) exit(1);
47    push(t);
48    if(pos == size(t->l)) return t;
```

```
48    if(pos < size(t->l)) return at(t->l, pos);
49    return at(t->r, pos - 1 - size(t->l));
50  }
51  int getpos(pnode t){//inversa de at
52    if(!t->parent) return size(t->l);
53    if(t==t->parent->l) return getpos(t->parent)-size(t->r)-1;
54    return getpos(t->parent)+size(t->l)+1;
55  }
56  void split(pnode t, int i, int j, pnode &l, pnode &m, pnode &r) {
57    split(t, i, l, t), split(t, j-i, m, r);}
58  Value get(pnode &p, int i, int j){//like rmq
59    pnode l,m,r;
60    split(p, i, j, l, m, r);
61    Value ret=mini(m);
62    p=merge(l, merge(m, r));
63    return ret;
64  }
65  void print(const pnode &t) {//for debugging
66    if(!t) return;
67    push(t);
68    print(t->l);
69    cout << t->val.fst << '␣';
70    print(t->r);
71  }
```

2.14. Convex Hull Trick

Dado un conjunto de funciones de la pinta f(x)=mx+b, encontrar $min_f(f(x))$, para un x dado. Lineal si los x vienen ordenados.

```
1   struct Line{tipo m,h;};
2   tipo inter(Line a, Line b){
3       // guarda que se rompe con paralelas
4       // ni idea con misma linea
5       tipo x=b.h-a.h, y=a.m-b.m;
6       return x/y+(x%y?!((x>0)^(y>0)):0);//==ceil(x/y)
7   }
8   struct CHT {
9     vector<Line> c;
10    bool mx;
11    int pos;
12    CHT(bool mx=0):mx(mx),pos(0){}//mx=1 si las query devuelven el max
13      //Creo que te da la iesima con m mas grande
14    inline Line acc(int i){return c[c[0].m>c.back().m? i : sz(c)-1-i];}
```

```
15    inline bool irre(Line x, Line y, Line z){
16      return c[0].m>z.m? inter(y, z) <= inter(x, y)
17                       : inter(y, z) >= inter(x, y);
18    }
19    void add(tipo m, tipo h) {//O(1) amortizado, los m tienen que entrar
        ordenados
20        if(mx) m*=-1, h*=-1;
21      Line l=(Line){m, h};
22        if(sz(c) && m==c.back().m) { l.h=min(h, c.back().h), c.pop_back();
              if(pos) pos--; }
23        while(sz(c)>=2 && irre(c[sz(c)-2], c[sz(c)-1], l)) { c.pop_back();
              if(pos) pos--; }
24        c.pb(l);
25    }
26    inline bool fbin(tipo x, int m) {return inter(acc(m), acc(m+1))>x;}//esta
        x en el bin m o antes?
27    tipo eval(tipo x){
28      int n = sz(c);
29      //query con x no ordenados O(lgn)
30      int a=-1, b=n-1;
31      while(b-a>1) { int m = (a+b)/2;
32        if(fbin(x, m)) b=m;
33        else a=m;
34      }
35      return (acc(b).m*x+acc(b).h)*(mx?-1:1);
36        //query O(1) amorrtizado
37      while(pos>0 && fbin(x, pos-1)) pos--;
38      while(pos<n-1 && !fbin(x, pos)) pos++;
39      return (acc(pos).m*x+acc(pos).h)*(mx?-1:1);
40    }
41  } ch;
```

2.15. Convex Hull Trick (Dynamic)

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

```
 9        if(!s) return 0;
10        ll x = rhs.m;
11        return b - s->b < (s->m - m) * x;
12    }
13 };
14 struct HullDynamic : public multiset<Line>{ // will maintain upper hull for
       maximum
15    bool bad(iterator y) {
16        iterator z = next(y);
17        if (y == begin()) {
18            if (z == end()) return 0;
19            return y->m == z->m && y->b <= z->b;
20        }
21        iterator x = prev(y);
22        if (z == end()) return y->m == x->m && y->b <= x->b;
23        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
24    }
25    iterator next(iterator y){return ++y;}
26    iterator prev(iterator y){return --y;}
27    void insert_line(ll m, ll b) {
28        iterator y = insert((Line) { m, b });
29        y->it=y;
30        if (bad(y)) { erase(y); return; }
31        while (next(y) != end() && bad(next(y))) erase(next(y));
32        while (y != begin() && bad(prev(y))) erase(prev(y));
33    }
34    ll eval(ll x) {
35        Line l = *lower_bound((Line) { x, is_query });
36        return l.m * x + l.b;
37    }
38 }h;
39 const Line *Line::succ(multiset<Line>::iterator it) const{
40    return (++it==h.end()? NULL : &*it);}
```

2.16.   Gain-Cost Set

```
1 //esta estructura mantiene pairs(beneficio, costo)
2 //de tal manera que en el set quedan ordenados
3 //por beneficio Y COSTO creciente. (va borrando los que no son optimos)
4 struct V{
5   int gain, cost;
6   bool operator<(const V &b)const{return gain<b.gain;}
7 };
```

```
 8 set<V> s;
 9 void add(V x){
10   set<V>::iterator p=s.lower_bound(x);//primer elemento mayor o igual
11   if(p!=s.end() && p->cost <= x.cost) return;//ya hay uno mejor
12   p=s.upper_bound(x);//primer elemento mayor
13   if(p!=s.begin()){//borro todos los peores (<=beneficio  y >=costo)
14     --p;//ahora es ultimo elemento menor o igual
15     while(p->cost >= x.cost){
16       if(p==s.begin()){s.erase(p); break;}
17       s.erase(p--);
18     }
19   }
20   s.insert(x);
21 }
22 int get(int gain){//minimo costo de obtener tal ganancia
23   set<V>::iterator p=s.lower_bound((V){gain, 0});
24   return p==s.end()? INF : p->cost;}
```

2.17.   Set con busq binaria

```
 1 #include<bits/stdc++.h>
 2 #include<ext/pb_ds/assoc_container.hpp>
 3 #include<ext/pb_ds/tree_policy.hpp>
 4 using namespace __gnu_pbds;
 5 using namespace std;
 6
 7 template <typename T>
 8 using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
 9
10 //o bien usar así:
11 typedef tree<int,null_type,less<int>,//key, mapped type, comparator. Se
      puede usar como map<a,b> poniendo tree<a,b,...
12     rb_tree_tag,tree_order_statistics_node_update> set_t;
13
14 int main(){
15   ordered_set<int>  s;
16   s.insert(1);
17   s.insert(3);
18   cout << s.order_of_key(3) << endl; // s.order_of_key(x): number of
         elements in s strictly less than x.
19   cout << *s.find_by_order(0) << endl; // s.find_by_order(i): i-th smallest
          number in s. (empieza en 0)
```

```
20    cout << *s.lower_bound(1) << endl;
21  }
22  //order_of_key(k): devuelve la pos del lower bound de k
23  //find_by_order(i) devuelve iterador al i-esimo elemento
24  //Ej: 12, 100, 505, 1000, 10000.
25  //order_of_key(10) == 0, order_of_key(100) == 1,
26  //order_of_key(707) == 3, order_of_key(9999999) == 5
27
28  /*
29  Si son int se puede hacer con un rmq y busqueda binaria.
30
31  rmq[i] = 1 si i esta
32  rmq[i] = 0 si i no esta
33
34  rmq.get(i,j) = suma en el intervalo [i,j]
35
36  order_of_key(i) == rmq.get(0,i)
37  find_by_order(o) == busqueda binaria en i / rmq.get(0,i+1) == o
38  lower_bound(i) == find_by_order(order_of_key(i)-1)
39  */
```

2.18.　RMQ Mixed

```
1  //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
        sobre el rango [i, j).
2  typedef int Elem;//Elem de los elementos del arreglo
3  typedef int Alt;//Elem de la alteracion//*
4  #define operacion(x,y) x+y
5  const Elem neutro=0; const Alt neutro2=0;//*
6  #define MAXN 100000
7  struct RMQ{
8    int sz;
9    Elem t[4*MAXN];
10   Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem//*
11   Elem &operator[](int p){return t[sz+p];}
12   void init(int n){//O(nlgn)
13     sz = 1 << (32-__builtin_clz(n));
14     forn(i, 2*sz) t[i]=neutro;
15     forn(i, 2*sz) dirty[i]=neutro2;//*
16   }
17   void updall(){//O(n)//para lazy limpiar dirty!
18     dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
19   void push(int n, int a, int b){//propaga el dirty a sus hijos//*
20     if(dirty[n]!=0){
21       t[n]+=dirty[n]*(b-a);//altera el nodo
22       if(n<sz){
23         dirty[2*n]+=dirty[n];
24         dirty[2*n+1]+=dirty[n];
25       }
26       dirty[n]=0;
27     }
28   }
29   Elem get(int i, int j, int n, int a, int b){//O(lgn)
30     if(j<=a || i>=b) return neutro;
31     push(n, a, b);//corrige el valor antes de usarlo//*
32     if(i<=a && b<=j) return t[n];
33     int c=(a+b)/2;
34     return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
35   }
36   Elem get(int i, int j){return get(i,j,1,0,sz);}
37   //altera los valores en [i, j) con una alteracion de val
38   void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)//*
39     push(n, a, b);
40     if(j<=a || i>=b) return;
41     if(i<=a && b<=j){
42       dirty[n]+=val;
43       push(n, a, b);
44       return;
45     }
46     int c=(a+b)/2;
47     alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
48     t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
49   }
50   void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}//*
51   void set(int p, tipo val){//O(lgn)//No usar con lazy!!
52     for(p+=sz; p>0 && t[p]!=val;){
53       t[p]=val;
54       p/=2;
55       val=operacion(t[p*2], t[p*2+1]);
56     }
57   }
58  }rmq;
```

2.19.　BIT

Me pa que es tipo un segment tree, pero hecho como un trie, se va haciendo dinámicamente.

```cpp
struct bitrie{
  static const int sz=1<<5;//5=ceil(log(n))
  int V;//valor del nodo
  vector<bitrie> ch;//childs, se puede hacer con un map
  bitrie():V(0){}//NEUTRO
  void set(int p, int v, int bit=sz>>1){//O(log sz)
    if(bit){
      ch.resize(2); //no cambia elementos si estaban
      ch[(p&bit)>0].set(p, v, bit>>1);
      V=max(ch[0].V, ch[1].V);
    }
    else V=v;
  }
  int get(int i, int j, int a=0, int b=sz){//O(log sz)
    if(j<=a || i>=b) return 0;//NEUTRO
    if(i<=a && b<=j) return V;
    if(!sz(ch)) return V;
    int c=(a+b)/2;
    return max(ch[0].get(i, j, a, c), ch[1].get(i, j, c, b));
  }
};
```

3.   Algos

3.1.   Longest Increasing Subsecuence

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

```cpp
  forn(i, N){
    int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
    if (d[j-1].first < a[i]&&a[i] < d[j].first){
      p[i]=d[j-1].second;
      d[j] = ii(a[i], i);
    }
  }
  R.clear();
  dforn(i, N+1) if(d[i].first!=INF){
    rec(d[i].second);//reconstruir
    reverse(R.begin(), R.end());
    return i;//longitud
  }
  return 0;
}
```

3.2.   Alpha-Beta prunning

```cpp
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

```cpp
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;
```

```
10  }
11  void mos(){
12      forn(i, t) qs[i].id=i;
13      sort(qs, qs+t, bymos);
14      int cl=0, cr=0;
15      sq=sqrt(n);
16      curans=0;
17      forn(i, t){ //intervalos cerrado abiertos !!! importante!!
18          Qu &q=qs[i];
19          while(cl>q.l) add(--cl);
20          while(cr<q.r) add(cr++);
21          while(cl<q.l) remove(cl++);
22          while(cr>q.r) remove(--cr);
23          ans[q.id]=curans;
24      }
25  }
```

3.4. huffman

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  typedef long long ll;
6
7  /* idea from following webpage
8   * https://www.siggraph.org/education/materials/HyperGraph/video/mpeg/
9         mpegfaq/huffman_tutorial.html
    */
10
11  struct huff {
12    ll v; /* value */
13    huff *r, *l; /* right, left branches */
14  };
15
16  typedef pair<ll, huff*> pih;
17
18  huff *build_huff(vector<ll> &e)
19  {
20    priority_queue<pih, vector<pih>, greater<pih>> pq;
21    for (auto &x: e)
22      pq.push(make_pair(x, nullptr));
23
24    while(pq.size() != 1) {
25      /* Get 2 nodes with lower value */
26      pih x = pq.top();
27      pq.pop();
28      pih y = pq.top();
29      pq.pop();
30
31      /* Combine them in a new node */
32      huff *w = new huff;
33      w->r = x.second;
34      w->l= y.second;
35      w->v = x.first+y.first;
36
37      /* Push new one to the pq */
38      pq.push(make_pair(w->v, w));
39    }
40
41    /* Only one node left => tree complete */
42    return pq.top().second;
43  }
44
45  ll sum_nuke_huff(huff *x)
46  {
47    /* Recursively sum all the values of the tree nodes while
48     * destroying the tree */
49    if (!x)
50      return 0;
51
52    ll tot = x->v + sum_nuke_huff(x->r) + sum_nuke_huff(x->l);
53    delete x->r;
54    delete x->l;
55    return tot;
56  }
57
58  int main()
59  {
60    ll n;
61    cin >> n;
62
63    for (ll i = 0; i < n; i++) {
64      ll t;
65      cin >> t;
66      vector<ll> a(t);
```

```
67      for (ll j = 0; j < t; j++)
68        cin >> a[j];
69
70      huff *o = build_huff(a);
71      cout << sum_nuke_huff(o) << endl;
72      delete o;
73    }
74  }
```

3.5. Optimizaciones para DP

```
1   //codeforces.com/blog/entry/8219
2
3   convex hull 1: dp[i] = min{dp[j] + b[j] * a[i]}, j < i. Si se cumple b[j]
        >= b[j+1] y a[i] <= a[i+1] entonces pasa de O(n^2) a O(n) sino pasa a O
        (nlogn)
4
5   input N
6   for i   [0..N)
7        input a[i]
8        input b[i]
9   let E = empty lower envelope structure
10  let dp[0] = 0
11  add the line y=mx+b to E, where m=b[0] and b=dp[0] //b is zero
12  for i   [1..N)
13       dp[i] = E.query(a[i])
14       if i<N
15           E.add(m=b[i],b=dp[i])
16  print dp[N]
17
18  //Mirar en ConvexHullTrick
19
20
21  convex hull 2: dp[i][j] = min{dp[i-1][k] + b[k] * a[j]}, k < j. Si se
        cumple b[k] >= b[k+1] y a[j] <= a[j+1] entonces pasa de O(kn^2) a O(kn)
        sino pasa O(knlogn)
22
23
24  divide and conquer: dp[i][j] = min{dp[i-1][k] + C[k+1][j]}, k < j. Se debe
        cumplir: A[i][j] <= A[i][j+1]. Pasa de O(kn^2) a O(knlogn)
25  Donde A[i][j] es el minimo k tal que dp[i][j] = dp[i-1][k] + C[k][j]
26  Tambien es aplicable si:
27  C[a][c] + C[b][d] <= C[a][d] + C[b][c] y C[b][c] <= C[a][d], a<=b<=c<=d
```

```
28
29  def ComputeDP(i, jleft, jright, kleft, kright):
30    # Select the middle point
31    jmid = (jleft + jright) / 2
32    # Compute the value of dp[i][jmid] by definition of DP
33    dp[i][jmid] = +INFINITY
34    bestk = -1
35    for k in range[kleft, jmid):
36    if dp[i - 1][k] + C[k + 1][jmid] < best:
37      dp[i][jmid] = dp[i - 1][k] + C[k + 1][jmid]
38      bestk = k
39    # Divide and conquer
40    if jleft < jmid:
41    ComputeDP(i, jleft, jmid, kleft, bestk)
42    if jmid + 1 < jright:
43    ComputeDP(i, jmid + 1, jright, bestk, kright)
44
45  def ComputeFullDP:
46    Initialize dp for i = 0 somehow
47    for i in range(1, m):
48    ComputeDP(i, 0, n, 0, n)
49
50
51  knuth: dp[i][j ]=min{dp[i][k ]+dp[k][j ]}+C[i][j], i < k < j. Se debe
        cumplir: A[i , j -1]<=A[i ,j ]<=A[i +1,j]. Pasa de O(n^3) a O(n^2)
52  Donde A[i][j] es el minimo k tal que dp[i][j] = dp[i][k ]+dp[k][j] + C[i][j
        ]
53  Tambien es aplicable si:
54  C[a][c] + C[b][d] <= C[a][d] + C[b][c] y C[b][c] <= C[a][d], a<=b<=c<=d
55
56  for (int s = 0; s<=k; s++)
57    for (int l = 0; l+s<=k; l++) {           //l - left point
58      int r = l + s;                         //r - right point
59      if (s < 2) {
60      res[l][r] = 0;                         //DP base - nothing to break
61      A[l][r] = l;                           //A is equal to left border
62      continue;
63      }
64      int aleft = A[l][r-1];                 //Knuth's trick: getting bounds
              on m
65      int aright = A[l+1][r];
66      res[l][r] = INF;
67      for (int a = max(l+1,aleft); a<=min(r-1,aright); a++) {      //iterating
```

```
          for a in the bounds only
68     int act = res[l][a] + res[a][r] + (C[l][r]);
69     if (res[l][r] > act) {                    //relax current solution
70       res[l][r] = act;
71       A[l][r] = a;
72     }
73     }
74   }
```

4. Strings

4.1. Manacher

```
1  int d1[MAXN];//d1[i]=long del maximo palindromo impar con centro en i
2  int d2[MAXN];//d2[i]=analogo pero para longitud par
3  //0 1 2 3 4
4  //a a b c c <--d1[2]=3
5  //a a b b <--d2[2]=2 (estan uno antes)
6  void manacher(){
7    int l=0, r=-1, n=sz(s);
8    forn(i, n){
9      int k=(i>r? 1 : min(d1[l+r-i], r-i));
10     while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) ++k;
11     d1[i] = k--;
12     if(i+k > r) l=i-k, r=i+k;
13   }
14   l=0, r=-1;
15   forn(i, n){
16     int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
17     while(i+k-1<n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
18     d2[i] = --k;
19     if(i+k-1 > r) l=i-k, r=i+k-1;
20   }
```

4.2. KMP

```
1  string T;//cadena donde buscar(where)
2  string P;//cadena a buscar(what)
3  int b[MAXLEN];//back table b[i] maximo borde de [0..i)
4  void kmppre(){//by gabina with love
5    int i =0, j=-1; b[0]=-1;
6    while(i<sz(P)){
7        while(j>=0 && P[i] != P[j]) j=b[j];
8        i++, j++, b[i] = j;
9    }
10 }
11 void kmp(){
12     int i=0, j=0;
13     while(i<sz(T)){
14         while(j>=0 && T[i]!=P[j]) j=b[j];
15         i++, j++;
16         if(j==sz(P)) printf("P␣is␣found␣at␣index␣%d␣in␣T\n", i-j), j=b[j];
17     }
18 }
19
20 int main(){
21     cout << "T=";
22     cin >> T;
23     cout << "P=";
```

4.3. Booth

```
1  // Booth's lexicographically minimal string rotation algorithm
2  template<class U, class T>
3  int boothLCS(T &v) // O(n)
4  {
5    size_t len = 2 * v.size();
6
7    // Duplicate original data to avoid modular arithmetic
8    vector<U> S(len);
9    for (size_t i = 0, sz = v.size(); i < sz; i++)
10     S[i] = S[v.size()+i] = v[i];
11
12   // Failure function
13   vector<int> f(len, -1);
14
15   // Minimal rotation found so far
16   int k = 0;
17
18   for (size_t j = 1; j < S.size(); j++) {
19     int i = f[j-k-1];
20     while (i != -1 && S[j] != S[k+i+1]) {
21       if (S[j] < S[k+i+1])
22         k = j-i-1;
23       i = f[i];
24   }
```

```
25     if (i == -1 && S[j] != S[k+i+1]) {
26       if (S[j] < S[k+i+1])
27         k = j;
28       f[j-k] = -1;
29     } else {
30       f[j-k] = i+1;
31     }
32   }
33
34   return k;
35 }
```

4.4.   Trie

```
1  struct trie{
2    map<char, trie> m;
3    void add(const string &s, int p=0){
4      if(s[p]) m[s[p]].add(s, p+1);
5    }
6    void dfs(){
7      //Do stuff
8      forall(it, m)
9        it->second.dfs();
10   }
11 };
```

4.5.   Regex

```
1  string s = "hola mundo feliz.";
2  regex r("^hola(\\smundo (\\w+\\.))$");
3  if (regex_match(s, r))
4    cout << "match" << endl;
5  smatch sm;
6  if (regex_match(s, sm, r))
7    for (auto &m: sm)
8      cout << "[" << m << "]" << endl;
9  /* match
10    [hola mundo feliz.]
11    [ mundo feliz.]
12    [feliz.] */
```

4.6.   Needleman Wunsnch

```
1  /* Longest common subsequence: DEL=INS=0, MATCH=1, MISMATCH=-INF
```

```
2   * Hamming: DEL=INS=-INF, MATCH=0, MISMATCH=1
3   * String alignment: normalmente DEL=INS=-1, MATCH=+2, MISMATCH=-1 */
4  #define DEL (0)
5  #define INS (0)
6  #define MATCH (1)
7  #define MISMATCH (-10000000)
8  #define MAXLEN 10000
9  ll nwt[MAXLEN][MAXLEN];
10
11 ll needleman_wunsnch(const char *A, const char *B) {
12   ll n = strlen(A), m = strlen(B);
13
14   forn(i, n+1) nwt[i][0] = i * INS;
15   forn(j, m+1) nwt[0][j] = j * DEL;
16
17   forr(i, 1, n+1) forr(j, 1, m+1) {
18     nwt[i][j] = nwt[i-1][j-1] + (A[i-1] == B[j-1] ? MATCH : MISMATCH);
19     nwt[i][j] = max(nwt[i][j], nwt[i - 1][j] + DEL);
20     nwt[i][j] = max(nwt[i][j], nwt[i][j - 1] + INS);
21   }
22
23   return nwt[n][m];
24 }
25
26 string lcs_construct(const char *A, const char *B) {
27   ll len = needleman_wunsnch(A, B), i = strlen(A), j = strlen(B);
28   string s;
29   s.resize(len);
30
31   while (i > 0 && j > 0) {
32     if (nwt[i-1][j] == nwt[i][j]) --i;
33     else if (nwt[i][j-1] == nwt[i][j]) --j;
34     else {
35       s[--len] = A[i-1];
36       --i, --j;
37     }
38   }
39
40   return s;
41 }
```

4.7.   Suffix Array (largo, nlogn)

```
1   #define MAX_N 112345
2   #define rBOUND(x) ((x) < n ? r[(x)] : 0)
3   //sa will hold the suffixes in order.
4   int sa[MAX_N], r[MAX_N], n;// OJO n = s.size()!
5   string s; //input string, n=s.size()
6
7   int f[MAX_N], tmpsa[MAX_N];
8   void countingSort(int k){
9     zero(f);
10    forn(i, n) f[rBOUND(i+k)]++;
11    int sum=0;
12    forn(i, max(255, n)){
13      int t=f[i]; f[i]=sum; sum+=t;}
14    forn(i,n)
15      tmpsa[f[rBOUND(sa[i]+k)]++]=sa[i];
16    forn(i,n) sa[i] = tmpsa[i];
17  }
18  void constructsa(){//O(n log n)
19    n = s.size();
20    forn(i,n) sa[i]=i, r[i]=s[i];
21    for(int k=1; k<n; k<<=1){
22      countingSort(k), countingSort(0);
23      int rank, tmpr[MAX_N];
24      tmpr[sa[0]]=rank=0;
25      forr(i, 1, n)
26        tmpr[sa[i]] = (r[sa[i]]==r[sa[i-1]] && r[sa[i]+k]==r[sa[i-1]+k]) ?
                rank : ++rank;
27      forn(i,n) r[i]=tmpr[i];
28      if(r[sa[n-1]]==n-1) break;
29    }
30  }
31  void print(){//for debugging
32    forn(i, n)
33      cout << i << '␣' <<
34      s.substr(sa[i], s.find('$',sa[i])-sa[i]) << endl;}
```

4.8.   String Matching With Suffix Array

```
1   //returns [lowerbound, upperbound] of the search -- los extremos estan
        incluidos!
2   pll stringMatching(string P){ //O(sz(P)lgn)
3     int lo=0, hi=n-1, mid=lo;
4     while(lo<hi){
```

```
5       mid=(lo+hi)/2;
6       int res=s.compare(sa[mid], sz(P), P);
7       if(res>=0) hi=mid;
8       else lo=mid+1;
9     }
10    if(s.compare(sa[lo], sz(P), P)!=0) return {-1, -1};
11    pll ans; ans.fst=lo;
12    lo=0, hi=n-1, mid;
13    while(lo<hi){
14      mid=(lo+hi)/2;
15      int res=s.compare(sa[mid], sz(P), P);
16      if(res>0) hi=mid;
17      else lo=mid+1;
18    }
19    if(s.compare(sa[hi], sz(P), P)!=0) hi--;
20    ans.snd=hi;
21    return ans;
22  }
```

4.9.   LCP (Longest Common Prefix)

```
1   //Calculates the LCP between consecutives suffixes in the Suffix Array.
2   //LCP[i] is the length of the LCP between sa[i] and sa[i-1]
3   int LCP[MAX_N], phi[MAX_N], PLCP[MAX_N];
4   void computeLCP(){//O(n)
5     phi[sa[0]]=-1;
6     forr(i, 1, n) phi[sa[i]]=sa[i-1];
7     int L=0;
8     forn(i, n){
9       if(phi[i]==-1) {PLCP[i]=0; continue;}
10      while(s[i+L]==s[phi[i]+L]) L++;
11      PLCP[i]=L;
12      L=max(L-1, 0);
13    }
14    forn(i, n) LCP[i]=PLCP[sa[i]];
15  }
```

4.10.   Corasick

```
1
2   struct trie{
3     map<char, trie> next;
4     trie* tran[256];//transiciones del automata
5     int idhoja, szhoja;//id de la hoja o 0 si no lo es
```

```cpp
   //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que es
       hoja
   trie *padre, *link, *nxthoja;
   char pch;//caracter que conecta con padre
   trie(): tran(),  idhoja(), padre(), link() {}
   void insert(const string &s, int id=1, int p=0){//id>0!!!
     if(p<sz(s)){
       trie &ch=next[s[p]];
       tran[(int)s[p]]=&ch;
       ch.padre=this, ch.pch=s[p];
       ch.insert(s, id, p+1);
     }
     else idhoja=id, szhoja=sz(s);
   }
   trie* get_link() {
     if(!link){
       if(!padre) link=this;//es la raiz
       else if(!padre->padre) link=padre;//hijo de la raiz
       else link=padre->get_link()->get_tran(pch);
     }
     return link; }
   trie* get_tran(int c) {
     if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
     return tran[c]; }
   trie *get_nxthoja(){
     if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
     return nxthoja; }
   void print(int p){
     if(idhoja) cout << "found " << idhoja << "  at position " << p-szhoja
         << endl;
     if(get_nxthoja()) get_nxthoja()->print(p); }
   void matching(const string &s, int p=0){
     print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1); }
}tri;


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

4.11. Suffix Automaton

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

```
37        st[clone].len = st[p].len + 1;
38        st[clone].next = st[q].next;
39        st[clone].link = st[q].link;
40        for (; p!=-1 && st[p].next.count(c) && st[p].next[c]==q; p=st[p].link
              )
41          st[p].next[c] = clone;
42        st[q].link = st[cur].link = clone;
43      }
44    }
45    last = cur;
46 }
```

4.12.   Z Function

```
1  char s[MAXN];
2  int z[MAXN]; // z[i] = i==0 ? 0 : max k tq s[0,k) match with s[i,i+k)
3  void z_function(char s[],int z[]) {
4      int n = strlen(s);
5      forn(i, n) z[i]=0;
6      for (int i = 1, l = 0, r = 0; i < n; ++i) {
7          if (i <= r) z[i] = min (r - i + 1, z[i - l]);
8          while (i + z[i] < n && s[z[i]] == s[i + z[i]])  ++z[i];
9          if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
10      }
11 }
12
13 int main() {
14     ios::sync_with_stdio(0);
```

4.13.   Suffix Tree

```
1  struct suffT{
2      int inf;
3      int maxn;
4      string orig;
5      string s;
6      vector<map<int, int> > to;
7      vector<int> len, fpos, link;
8      int node, pos, sz, n;
9
10     suffT(string k){
11         inf = 1000000000;
12         orig = k;
13         orig += '#'; //Caracter que no aparezca
```

```
14         s = orig;
15         maxn = 4*s.size() + 3;
16         to = vector<map<int, int> > (maxn);
17         len = vector<int> (maxn, 0);
18         fpos = vector<int> (maxn, 0);
19         link = vector<int> (maxn, 0);
20         node = pos = n = 0;
21         len[0] = 1e9;
22         sz = 1;
23
24         for(int i = 0; i < s.size(); i++){
25             debug(i);
26             add_letter(s[i]);
27         }
28     }
29
30     int make_node(int _pos, int _len){
31         fpos[sz] = _pos;
32         len [sz] = _len;
33         return sz++;
34     }
35
36     void go_edge(){
37         while(pos > len[to[node][s[n - pos]]]){
38             node = to[node][s[n - pos]];
39             pos -= len[node];
40         }
41     }
42
43     void add_letter(int c){
44         s[n++] = c;
45         pos++;
46         int last = 0;
47         while(pos > 0){
48             go_edge();
49             int edge = s[n - pos];
50             int &v = to[node][edge];
51             int t = s[fpos[v] + pos - 1];
52             if(v == 0){
53                 v = make_node(n - pos, inf);
54                 link[last] = node;
55                 last = 0;
56             }else if(t == c){
```

```
 57              link[last] = node;
 58              return;
 59          }else{
 60              int u = make_node(fpos[v], pos - 1);
 61              to[u][c] = make_node(n - 1, inf);
 62              to[u][t] = v;
 63              fpos[v] += pos - 1;
 64              len [v] -= pos - 1;
 65              v = u;
 66              link[last] = u;
 67              last = u;
 68          }
 69          if(node == 0)
 70              pos--;
 71          else
 72              node = link[node];
 73      }
 74  }
 75
 76
 77  int search(string k){
 78      int cur = 0;
 79      bool encontrado = true;
 80      int visto = 0;
 81      while (encontrado and visto < k.size()){
 82          if (encontrado = encontrado and to[cur].find(k[visto]) != to[cur].
                 end()){
 83              cur = to[cur][k[visto]];
 84              forsn(i, fpos[cur], len[cur] + fpos[cur]){
 85                  if (visto == k.size()) return true;
 86                  if (k[visto++] != s[i]) return false;
 87              }
 88          }
 89      }
 90      return false;
 91  }
 92
 93  pair<int, int> dfs(int cur = 0, int acum = 0){
 94      if (len[cur] > maxn and cur != 0){
 95          pair<int, int> resp {1, 0};
 96          return resp;
 97      }
 98
 99      int hojas = 0;
100      int contados = 0;
101      bool todosHojas = true;
102      for(const auto&  n : to[cur]){
103          pair<int, int> next = dfs(n.snd, (len[n.snd] > maxn ? s.size() - 1
                  : acum + len[n.snd]));//o len[n.snd]
104          todosHojas = todosHojas && (next.fst == 1);
105          hojas += next.fst;
106          contados += next.snd;
107      }
108      if (hojas > 1 and todosHojas){
109          res += (tint)acum;
110          contados++;
111      }else if (contados > 1){
112          res -= (tint)((contados - 1)*acum);
113          contados = 1;
114      }
115      return make_pair(hojas, contados);
116  }
117
118  };
```

5. Geometria

5.1. Punto

```
 1  struct pto{
 2      double x, y;
 3      pto(double x=0, double y=0):x(x),y(y){}
 4      pto operator+(pto a){return pto(x+a.x, y+a.y);}
 5      pto operator-(pto a){return pto(x-a.x, y-a.y);}
 6      pto operator+(double a){return pto(x+a, y+a);}
 7      pto operator*(double a){return pto(x*a, y*a);}
 8      pto operator/(double a){return pto(x/a, y/a);}
 9      //dot product, producto interno:
10      double operator*(pto a){return x*a.x+y*a.y;}
11      //module of the cross product or vectorial product:
12      //if a is less than 180 clockwise from b, a^b>0
13      double operator^(pto a){return x*a.y-y*a.x;}
14      //returns true if this is at the left side of line qr
15      bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
16      bool operator<(const pto &a) const{return x<a.x-EPS || (abs(x-a.x)<EPS &&
             y<a.y-EPS);}
```

```
17    bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
18    double norm(){return sqrt(x*x+y*y);}
19    double norm_sq(){return x*x+y*y;}
20  };
21  double dist(pto a, pto b){return (b-a).norm();}
22  typedef pto vec;
23
24  //positivo si aob están en sentido antihorario con un ángulo <180º
25  double angle(pto a, pto o, pto b){ //devuelve radianes! (-pi,pi)
26    pto oa=a-o, ob=b-o;
27    return atan2(oa^ob, oa*ob);}
28
29  //rotate p by theta rads CCW w.r.t. origin (0,0)
30  pto rotate(pto p, double theta){
31    return pto(p.x*cos(theta)-p.y*sin(theta),
32       p.x*sin(theta)+p.y*cos(theta));
```

5.2.   Orden radial de puntos

```
1  struct Cmp{//orden total de puntos alrededor de un punto r
2    pto r;
3    Cmp(pto r):r(r) {}
4    int cuad(const pto &a) const{
5      if(a.x > 0 && a.y >= 0)return 0;
6      if(a.x <= 0 && a.y > 0)return 1;
7      if(a.x < 0 && a.y <= 0)return 2;
8      if(a.x >= 0 && a.y < 0)return 3;
9      assert(a.x ==0 && a.y==0);
10     return -1;
11   }
12   bool cmp(const pto&p1, const pto&p2)const{
13     int c1 = cuad(p1), c2 = cuad(p2);
14     if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
15        else return c1 < c2;
16   }
17     bool operator()(const pto&p1, const pto&p2) const{
18     return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
19     }
20  };
```

5.3.   Line

```
1  int sgn(ll x){return x<0? -1 : !!x;}
2  struct line{
```

```
3    line() {}
4    double a,b,c;//Ax+By=C
5  //pto MUST store float coordinates!
6    line(double a, double b, double c):a(a),b(b),c(c){}
7    line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
8    int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
9  };
10 bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
11 pto inter(line l1, line l2){//intersection
12   double det=l1.a*l2.b-l2.a*l1.b;
13   if(abs(det)<EPS) return pto(INF, INF);//parallels
14   return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
15 }
```

5.4.   Segment

```
1  struct segm{
2    pto s,f;
3    segm(pto s, pto f):s(s), f(f) {}
4    pto closest(pto p) {//use for dist to point
5      double l2 = dist_sq(s, f);
6      if(l2==0.) return s;
7      double t =((p-s)*(f-s))/l2;
8      if (t<0.) return s;//not write if is a line
9      else if(t>1.)return f;//not write if is a line
10     return s+((f-s)*t);
11   }
12     bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS;}
13 };
14
15 pto inter(segm s1, segm s2){
16   pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
17     if(s1.inside(r) && s2.inside(r)) return r;
18   return pto(INF, INF);
19 }
```

5.5.   Rectangle

```
1  struct rect{
2    //lower-left and upper-right corners
3    pto lw, up;
4  };
5  //returns if there's an intersection and stores it in r
6  bool inter(rect a, rect b, rect &r){
```

```
7    r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
8    r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
9  //check case when only a edge is common
10   return r.lw.x<r.up.x && r.lw.y<r.up.y;
11 }
```

5.6. Polygon Area

```
1  double area(vector<pto> &p){//O(sz(p))
2    double area=0;
3    forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4    //if points are in clockwise order then area is negative
5    return abs(area)/2;
6  }
7  //Area ellipse = M_PI*a*b where a and b are the semi axis lengths
8  //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

5.7. Circle

```
1  vec perp(vec v){return vec(-v.y, v.x);}
2  line bisector(pto x, pto y){
3    line l=line(x, y); pto m=(x+y)/2;
4    return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5  }
6  struct Circle{
7    pto o;
8    double r;
9    Circle(pto x, pto y, pto z){
10     o=inter(bisector(x, y), bisector(y, z));
11     r=dist(o, x);
12   }
13   pair<pto, pto> ptosTang(pto p){
14     pto m=(p+o)/2;
15     tipo d=dist(o, m);
16     tipo a=r*r/(2*d);
17     tipo h=sqrt(r*r-a*a);
18     pto m2=o+(m-o)*a/d;
19     vec per=perp(m-o)/d;
20     return make_pair(m2-per*h, m2+per*h);
21   }
22 };
23 //finds the center of the circle containing p1 and p2 with radius r
24 //as there may be two solutions swap p1, p2 to get the other
25 bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
```

```
26     double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
27     if(det<0) return false;
28     c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
29     return true;
30 }
31 #define sqr(a) ((a)*(a))
32 #define feq(a,b) (fabs((a)-(b))<EPS)
33 pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
34   tipo dx = sqrt(b*b-4.0*a*c);
35   return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
36 }
37 pair<pto, pto> interCL(Circle c, line l){
38   bool sw=false;
39   if((sw=feq(0,l.b))){
40     swap(l.a, l.b);
41     swap(c.o.x, c.o.y);
42   }
43   pair<tipo, tipo> rc = ecCuad(
44     sqr(l.a)+sqr(l.b),
45     2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
46     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
47   );
48   pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
49           pto(rc.second, (l.c - l.a * rc.second) / l.b) );
50   if(sw){
51     swap(p.first.x, p.first.y);
52     swap(p.second.x, p.second.y);
53   }
54   return p;
55 }
56 pair<pto, pto> interCC(Circle c1, Circle c2){
57   line l;
58   l.a = c1.o.x-c2.o.x;
59   l.b = c1.o.y-c2.o.y;
60   l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
61   -sqr(c2.o.y))/2.0;
62   return interCL(c1, l);
63 }
```

5.8. Point in Poly

```
1  //checks if v is inside of P, using ray casting
2  //works with convex and concave.
```

```
3   //excludes boundaries, handle it separately using segment.inside()
4   bool inPolygon(pto v, vector<pto>& P) {
5     bool c = false;
6     forn(i, sz(P)){
7       int j=(i+1)%sz(P);
8       if((P[j].y>v.y) != (P[i].y > v.y) &&
9       (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
10          c = !c;
11    }
12    return c;
13  }
```

5.9.   Point in Convex Poly log(n)

```
1   void normalize(vector<pto> &pt){//delete collinear points first!
2     //this makes it clockwise:
3       if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
4     int n=sz(pt), pi=0;
5     forn(i, n)
6       if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
7         pi=i;
8     vector<pto> shift(n);//puts pi as first point
9       forn(i, n) shift[i]=pt[(pi+i)%n];
10    pt.swap(shift);
11  }
12
13  /* left debe decir >0 para que considere los bordes. Ojo que Convex Hull
14      necesita que left diga >= 0 para limpiar los colineales, hacer otro
          left
15      si hace falta */
16  bool inPolygon(pto p, const vector<pto> &pt){
17    //call normalize first!
18    if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
19    int a=1, b=sz(pt)-1;
20    while(b-a>1){
21      int c=(a+b)/2;
22      if(!p.left(pt[0], pt[c])) a=c;
23      else b=c;
24    }
25    return !p.left(pt[a], pt[a+1]);
26  }
```

5.10.   Convex Check CHECK

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

5.11.   Convex Hull

```
1   //stores convex hull of P in S, CCW order
2   //left must return >=0 to delete collinear points!
3   void CH(vector<pto>& P, vector<pto> &S){
4     S.clear();
5     sort(P.begin(), P.end());//first x, then y
6     forn(i, sz(P)){//lower hull
7       while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
8       S.pb(P[i]);
9     }
10    S.pop_back();
11    int k=sz(S);
12    dforn(i, sz(P)){//upper hull
13      while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
14      S.pb(P[i]);
15    }
16    S.pop_back();
17  }
```

5.12.   Cut Polygon

```
1   //cuts polygon Q along the line ab
2   //stores the left side (swap a, b for the right one) in P
3   void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
4     P.clear();
5     forn(i, sz(Q)){
6       double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
7       if(left1>=0) P.pb(Q[i]);
8       if(left1*left2<0)
9         P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
10    }
11  }
```

5.13.   Bresenham

```
1  //plot a line approximation in a 2d map
2  void bresenham(pto a, pto b){
3    pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
4    pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
5    int err=d.x-d.y;
6    while(1){
7      m[a.x][a.y]=1;//plot
8      if(a==b) break;
9      int e2=err;
10     if(e2 >= 0) err-=2*d.y, a.x+=s.x;
11     if(e2 <= 0) err+= 2*d.x, a.y+= s.y;
12   }
13 }
```

5.14. Rotate Matrix

```
1  //rotates matrix t 90 degrees clockwise
2  //using auxiliary matrix t2(faster)
3  void rotate(){
4    forn(x, n) forn(y, n)
5      t2[n-y-1][x]=t[x][y];
6    memcpy(t, t2, sizeof(t));
7  }
```

5.15. Interseccion de Circulos en n3log(n)

```
1  struct event {
2      double x; int t;
3      event(double xx, int tt) : x(xx), t(tt) {}
4      bool operator <(const event &o) const { return x < o.x; }
5  };
6  typedef vector<Circle> VC;
7  typedef vector<event> VE;
8  int n;
9  double cuenta(VE &v, double A,double B) {
10     sort(v.begin(), v.end());
11     double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
12     int contador = 0;
13     forn(i,sz(v)) {
14         //interseccion de todos (contador == n), union de todos (contador >
               0)
15         //conjunto de puntos cubierto por exacta k Circulos (contador == k)
16         if (contador == n) res += v[i].x - lx;
17         contador += v[i].t, lx = v[i].x;
```

```
18     }
19     return res;
20 }
21 // Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
22 inline double primitiva(double x,double r) {
23     if (x >= r) return r*r*M_PI/4.0;
24     if (x <= -r) return -r*r*M_PI/4.0;
25     double raiz = sqrt(r*r-x*x);
26     return 0.5 * (x * raiz + r*r*atan(x/raiz));
27 }
28 double interCircle(VC &v) {
29     vector<double> p; p.reserve(v.size() * (v.size() + 2));
30     forn(i,sz(v))  p.push_back(v[i].c.x + v[i].r), p.push_back(v
           [i].c.x - v
           [i].r);
31     forn(i,sz(v)) forn(j,i) {
32         Circle &a = v[i], b = v[j];
33         double d = (a.c - b.c).norm();
34         if (fabs(a.r - b.r) < d && d < a.r + b.r) {
35             double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d *
                   a.r));
36             pto vec = (b.c - a.c) * (a.r / d);
37             p.pb((a.c + rotate(vec, alfa)).x), p.pb((a.c + rotate(vec, -
                   alfa)).x);
38         }
39     }
40     sort(p.begin(), p.end());
41     double res = 0.0;
42     forn(i,sz(p)-1) {
43         const double A = p[i], B = p[i+1];
44         VE ve; ve.reserve(2 * v.size());
45         forn(j,sz(v)) {
46             const Circle &c = v[j];
47             double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
48             double base = c.c.y * (B-A);
49             ve.push_back(event(base + arco,-1));
50             ve.push_back(event(base - arco, 1));
51         }
52         res += cuenta(ve,A,B);
53     }
54     return res;
55 }
```

6. Math

6.1. Identidades

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

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

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

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

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

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

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

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

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

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

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

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

6.2. Ec. Caracteristica

$$a_0 T(n) + a_1 T(n-1) + ... + a_k T(n-k) = 0$$

$$p(x) = a_0 x^k + a_1 x^{k-1} + ... + a_k$$

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.

6.3. Combinatorio

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

6.4. Stars and bars

Teniendo n estrellas y k cajas (k-1 barras):

- conjuntos no vacios: $\binom{n-1}{k-1}$;

- conjuntos vacios: $\binom{n+k-1}{k-1}$.

6.5. Teoria de Juegos (básico)

<u>Nim</u>: tengo varias pilas de tamaño n_i, puedo sacar la cantidad que quiera de una sola pila, al menos uno. Pierdo si no hay pilas. La posición es ganadora sii $n_1 \ xor \ n_2 \ xor...n_k$ es positivo.

<u>Otros juegos</u>, para cada posición obtengo la pila de Nim equivalente:

- Si la posición es perderora, su nimber es 0;

- sino, es el mex, de los nimbers de las posiciones alcanzables desde esta.

6.6. Exp. de Numeros Mod.

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

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

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

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

```
struct Mat {
    vector<vector<double> > vec;
    Mat(int n): vec(n, vector<double>(n) ) {}
```

```
4      Mat(int n, int m): vec(n, vector<double>(m) ) {}
5      vector<double> &operator[](int f){return vec[f];}
6      const vector<double> &operator[](int f) const {return vec[f];}
7      int size() const {return sz(vec);}
8      Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
9          Mat m(sz(b),sz(b[0]));
10         forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j];
11         return m;     }
12     Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
13         int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
14         Mat mat(n,t);
15         forn(i,n) forn(j,t) forn(k,m) mat[i][j] += vec[i][k] * b[k][j];
16         return mat;     }
17     double determinant(){//sacado de e maxx ru
18         double det = 1;
19         int n = sz(vec);
20         Mat m(*this);
21         forn(i, n){//para cada columna
22             int k = i;
23             forr(j, i+1, n)//busco la fila con mayor val abs
24                 if(abs(m[j][i])>abs(m[k][i])) k = j;
25             if(abs(m[k][i])<1e-9) return 0;
26             m[i].swap(m[k]);//la swapeo
27             if(i!=k) det = -det;
28             det *= m[i][i];
29             forr(j, i+1, n) m[i][j] /= m[i][i];
30             //hago 0 todas las otras filas
31             forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
32                 forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
33         }
34         return det;
35     }
36 };
37
38 int n;
39 int main() {
40 //DETERMINANTE:
41 //https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&
      page=show_problem&problem=625
42   freopen("input.in", "r", stdin);
43     ios::sync_with_stdio(0);
44     while(cin >> n && n){
45         Mat m(n);
```

```
46         forn(i, n) forn(j, n) cin >> m[i][j];
47         cout << (ll)round(m.determinant()) << endl;
48     }
49     cout  << "*" << endl;
50   return 0;
51 }
```

6.9.   Teorema Chino del Resto

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

```
1  struct dxy {tint d,x,y;};
2  dxy mcde(tint a, tint b) {
3      dxy r, t;
4      if (b == 0) {
5          r.d = a; r.x = 1; r.y = 0;
6      } else {
7          t = mcde(b,a %b);
8          r.d = t.d; r.x = t.y;
9          r.y = t.x - a/b*t.y;
10     }
11     return r;
12 }
13 tint modq(x, q) { return (x %q + q) %q ; }
14 tint tcr(tint* r, tint* m, int n) {
15     tint p=0, q=1;
16     forn(i, n) {
17         p = modq(p-r[i], q);
18         dxy w = mcde(m[i], q);
19         if (p %w.d) return -1; // sistema incompaible
20         q = q / w.d * m[i];
21         p = modq(r[i] + m[i] * p / w.d * w.x, q);
22     }
23     return p; // x \equiv p (q)
24 }
```

6.10.   Criba

```
1  #define MAXP 100000 //no necesariamente primo
```

```
2   int criba[MAXP+1];
3   void crearcriba(){
4     int w[] = {4,2,4,2,4,6,2,6};
5     for(int p=25;p<=MAXP;p+=10) criba[p]=5;
6     for(int p=9;p<=MAXP;p+=6) criba[p]=3;
7     for(int p=4;p<=MAXP;p+=2) criba[p]=2;
8     for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
9       for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
10  }
11  vector<int> primos;
12  void buscarprimos(){
13    crearcriba();
14    forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
15  }
16  //~ Useful for bit trick: #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) ), #
        define INDEX(i) ( (criba[i>>5]>>((i)&31))&1 ), unsigned int criba[MAXP
        /32+1];
17
18
19  int main() {
20    freopen("primos", "w", stdout);
21    buscarprimos();
```

6.11.   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

```
1   //factoriza bien numeros hasta MAXP^2
2   map<tint,tint> fact(tint n){ //O (cant primos)
3     map<tint,tint> ret;
4     for(auto p : primos){
5       while(!(n%p)){
6         ret[p]++;//divisor found
7         n/=p;
8       }
9     }
10    if(n>1) ret[n]++;
11    return ret;
12  }
13  //factoriza bien numeros hasta MAXP
14  map<tint,tint> fact2(tint n){ //O (lg n)
15    map<tint,tint> ret;
16    while (criba[n]){
17      ret[criba[n]]++;
18      n/=criba[n];
19    }
20    if(n>1) ret[n]++;
21    return ret;
22  }
23  //Usar asi:  divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
24  void divisores(const map<tint,tint> &f, vector<tint> &divs, map<tint,tint
         >::iterator it, tint n=1){
25    if(it==f.begin()) divs.clear();
26    if(it==f.end()) { divs.pb(n);  return; }
27    tint p=it->fst, k=it->snd; ++it;
28    forn(_, k+1) divisores(f, divs, it, n), n*=p;
29  }
30  tint sumDiv (tint n){
31    tint rta = 1;
32    map<tint,tint> f=fact(n);
33    for(auto it : f) {
34    tint pot = 1, aux = 0;
35    forn(i, it.snd+1) aux += pot, pot *= it.fst;
36    rta*=aux;
37    }
38    return rta;
39  }
40  tint eulerPhi (tint n){ // con criba: O(lg n)
41    tint rta = n;
42    map<tint,tint> f=fact(n);
43    for(auto it : f) rta -= rta / it.first;
44    return rta;
45  }
46  tint eulerPhi2 (tint n){ // O (sqrt n)
47    tint r = n;
48    forsn (i,2,n+1){
49      if ((tint)i*i > n) break;
50      if (n % i == 0){
51        while (n%i == 0) n/=i;
52        r -= r/i; }
53    }
54    if (n != 1) r-= r/n;
55    return r;
56  }
```

6.12.   Test de primalidad naive $O\left(\text{sqrt}(n)/6\right)$

```
int __attribute__((const)) is_prime(long long n)
{
  if (n <= 1)
    return 0;
  else if (n <= 3)
    return 1;
  else if (!(n % 2) || !(n % 3))
    return 0;

  long long cap = sqrt(n) + 1;
  for (long long int i = 5; i <= cap; i += 6)
    if (!(n%i) || !(n%(i+2)))
      return 0;

  return 1;
}
```

6.13.   Phollard's Rho (rolando)

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

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

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

bool es_primo_prob (ll n, int a)
{
  if (n == a) return true;
  ll s = 0,d = n-1;
  while (d % 2 == 0) s++,d/=2;

  ll x = expmod(a,d,n);
  if ((x == 1) || (x+1 == n)) return true;

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

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

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

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

```
66   ll factor = rho(n);
67   factRho(factor);
68   factRho(n/factor);
69 }
```

6.14. GCD

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

6.15. Extended Euclid

```
1 void extendedEuclid (ll a, ll b){ //a * x + b * y = d
2   if (!b) { x = 1; y = 0; d = a; return;}
3   extendedEuclid (b, a%b);
4   ll x1 = y;
5   ll y1 = x - (a/b) * y;
6   x = x1; y = y1;
7 }
```

6.16. LCM

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

6.17. Inversos

```
1 #define MAXMOD 15485867
2 ll inv[MAXMOD];//inv[i]*i=1 mod MOD
3 void calc(int p){//O(p)
4   inv[1]=1;
5   forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
6 }
7 int inverso(int x){//O(log x)
8   return expmod(x, eulerphi(MOD)-1);//si mod no es primo(sacar a mano)
         PROBAR!
9   return expmod(x, MOD-2);//si mod es primo
10 }
```

6.18. Simpson

```
1 // Para intervalo [0, 8], polinomio de a lo sumo grado 8, 2700 divisiones
       alcanzaron
2 // con error de a lo sumo 10e-5
3 double integral(double a, double b, int n=10000) {//O(n), n=cantdiv
4   double area=0, h=(b-a)/n, fa=f(a), fb;
5   forn(i, n){
```

```
6     fb=f(a+h*(i+1));
7     area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
8   }
9   return area*h/6.;}
```

6.19. Fraction

```
1 tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
2 struct frac{
3   tipo p,q;
4   frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
5   void norm(){
6     tipo a = mcd(p,q);
7     if(a) p/=a, q/=a;
8     else q=1;
9     if (q<0) q=-q, p=-p;}
10  frac operator+(const frac& o){
11    tipo a = mcd(q,o.q);
12    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
13  frac operator-(const frac& o){
14    tipo a = mcd(q,o.q);
15    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
16  frac operator*(frac o){
17    tipo a = mcd(q,o.p), b = mcd(o.q,p);
18    return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
19  frac operator/(frac o){
20    tipo a = mcd(q,o.q), b = mcd(o.p,p);
21    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
22  bool operator<(const frac &o) const{return p*o.q < o.p*q;}
23  bool operator==(frac o){return p==o.p&&q==o.q;}
24 };
```

6.20. Polinomio

```
1     /*tipo a0=0;
2     while(a0 == 0 && i < sz(c)) a0 = c[i], i++;*/
3     int j = sz(c)-1;
4     tipo an=0;
5     while(an == 0 && j >=i) an = c[j], j--;
6     vector<tipo> d;
7     forr(k,i,j) d.pb(c[k]);
8     c=d;
9   }
10  bool isnull() { simplify(); return c.empty();}
```

```cpp
    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;
    simplify();
    if(c[0]) roots.insert(0);
    int i = 0;
    tipo a0=0;
    while(a0 == 0 && i < sz(c)) a0 = abs(c[i]), i++;
    tipo 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 ) { //sacar esto para obtener
            todas las raices racionales
        tipo root = abs((*pt) / (*qt));
        if (eval(root)==0) roots.insert(root);
        if (eval((-1)*root)==0) roots.insert((-1)*root);// las raices
            tambien pueden ser negativas!
      }
    return roots; }
};
```

```cpp
pair<poly,tipo> ruffini(const poly p, tipo r) { //divive el polinomio p por
    (x-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) { //O(n^2)
    poly A; A.c.pb(1);
    forn(i,sz(x)) { poly aux; aux.c.pb(-x[i]), aux.c.pb(1), A = A * aux; }
        // A = (x-x0) * ... * (x-xn)
  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 -- si se usa mod
        usar el inverso!
    S = S + Li * y[i];  }
  return S;
}
```

6.21. Ec. Lineales

```cpp
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
```

```
17    forr(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de
          compatibilidad
18    x = vector<tipo>(m, 0);
19    dforn(i, rw){
20      tipo s = y[i];
21      forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
22      x[p[i]] = s / a[i][i]; //aca divide
23    }
24    ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
25    forn(k, m-rw) {
26      ev[k][p[k+rw]] = 1;
27      dforn(i, rw){
28        tipo s = -a[i][k+rw];
29        forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
30        ev[k][p[i]] = s / a[i][i]; //aca divide
31      }
32    }
33    return true;
34  }
```

6.22.   FFT

```
1   //~ typedef complex<double> base; //menos codigo, pero mas lento
2   //elegir si usar complejos de c (lento) o estos
3   struct base{
4       double r,i;
5       base(double r=0, double i=0):r(r), i(i){}
6       double real()const{return r;}
7       void operator/=(const int c){r/=c, i/=c;}
8   };
9   base operator*(const base &a, const base &b){
10      return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
11  base operator+(const base &a, const base &b){
12      return base(a.r+b.r, a.i+b.i);}
13  base operator-(const base &a, const base &b){
14      return base(a.r-b.r, a.i-b.i);}
15  vector<int> rev; vector<base> wlen_pw;
16  inline static void fft(base a[], int n, bool invert) {
17      forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
18    for (int len=2; len<=n; len<<=1) {
19      double ang = 2*M_PI/len * (invert?-1:+1);
20      int len2 = len>>1;
21      base wlen (cos(ang), sin(ang));
```

```
22      wlen_pw[0] = base (1, 0);
23          forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
24      for (int i=0; i<n; i+=len) {
25        base t, *pu = a+i, *pv = a+i+len2,  *pu_end = a+i+len2, *pw = &
              wlen_pw[0];
26        for (; pu!=pu_end; ++pu, ++pv, ++pw)
27          t = *pv * *pw, *pv = *pu - t,*pu = *pu + t;
28      }
29    }
30    if (invert) forn(i, n) a[i]/= n;}
31  inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
32      wlen_pw.resize(n), rev.resize(n);
33      int lg=31-__builtin_clz(n);
34      forn(i, n){
35      rev[i] = 0;
36          forn(k, lg) if(i&(1<<k)) rev[i]|=1<<(lg-1-k);
37      }}
38  //multiplica vectores en nlgn
39  inline static void multiply(const vector<int> &a, const vector<int> &b,
        vector<int> &res) {
40    vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
41      int n=1; while(n < max(sz(a), sz(b))) n <<= 1; n <<= 1;
42      calc_rev(n);
43    fa.resize (n),  fb.resize (n);
44    fft (&fa[0], n, false),  fft (&fb[0], n, false);
45    forn(i, n) fa[i] = fa[i] * fb[i];
46    fft (&fa[0], n, true);
47    res.resize(n);
48      forn(i, n) res[i] = int (fa[i].real() + 0.5); }
49  void toPoly(const string &s, vector<int> &P){//convierte un numero a
        polinomio
50      P.clear();
51      dforn(i, sz(s)) P.pb(s[i]-'0');}
```

6.23.   Sum of Subsets

$$F[mask] = \sum_{i \subseteq mask} A[i]$$

$$O(N2^N)$$

```
for(int i = 0; i<(1<<N); ++i)
  F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
  if(mask & (1<<i))
    F[mask] += F[mask^(1<<i)];
}
```

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

Factoriales

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

max signed tint = 9.223.372.036.854.775.807
max unsigned tint = 18.446.744.073.709.551.615

Primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109
113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227
229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347
349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461
463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599
601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727
733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859
863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009
1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103
1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229
1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327
1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471
1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579
1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697
1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951
1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

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

Cantidad de primos menores que 10^n

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

Divisores

Cantidad de divisores (σ_0) para *algunos* $n/\neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$
$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$
Suma de divisores (σ_1) para *algunos* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

7. Grafos

7.1. Dijkstra

```
#define INF 1e9
int N;
#define MAX_V 250001
vector<ii> G[MAX_V];
//To add an edge use
```

```
6   #define add(a, b, w) G[a].pb(make_pair(w, b))
7   ll dijkstra(int s, int t){//O(|E| log |V|)
8     priority_queue<ii, vector<ii>, greater<ii> > Q;
9     vector<ll> dist(N, INF); vector<int> dad(N, -1);
10    Q.push(make_pair(0, s)); dist[s] = 0;
11    while(sz(Q)){
12      ii p = Q.top(); Q.pop();
13      if(p.snd == t) break;
14      forall(it, G[p.snd])
15        if(dist[p.snd]+it->first < dist[it->snd]){
16          dist[it->snd] = dist[p.snd] + it->fst;
17          dad[it->snd] = p.snd;
18          Q.push(make_pair(dist[it->snd], it->snd));  }
19    }
20    return dist[t];
21    if(dist[t]<INF)//path generator
22      for(int i=t; i!=-1; i=dad[i])
23        printf("%d%c", i, (i==s?'\n':' '));}
```

7.2.   Bellman-Ford

```
1   #define INF 1e9
2   #define MAX_N 1001
3   vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
4   //To add an edge use
5   #define add(a, b, w) G[a].pb(make_pair(w, b))
6   int dist[MAX_N];
7   int N; //cantidad de vertices -- setear!!
8   void bford(int src){//O(VE)
9     memset(dist,INF,sizeof dist);
10    dist[src]=0;
11    forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
12      dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
13  }
14
15  bool hasNegCycle(){
16    forn(j, N) if(dist[j]!=INF) forall(it, G[j])
17      if(dist[it->snd]>dist[j]+it->fst) return true;
18    //inside if: all points reachable from it->snd will have -INF distance(do
          bfs) ?
19    return false;
20  }
```

7.3.   Floyd-Warshall

```
1   //G[i][j] contains weight of edge (i, j) or INF
2   //G[i][i]=0
3   int G[MAX_N][MAX_N];
4   void floyd(){//O(N^3)
5   forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
6     G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
7   }
8   bool inNegCycle(int v){
9     return G[v][v]<0;}
10  //checks if there's a neg. cycle in path from a to b
11  bool hasNegCycle(int a, int b){
12    forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
13      return true;
14    return false;
15  }
```

7.4.   Kruskal

```
1   const int MAXN=100000;
2   vector<ii> G[MAXN];
3   int n;
4
5   struct Ar{int a,b,w;}; //w y cost deberian tener el mismo tipo
6   bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
7   vector<Ar> E;
8   ll kruskal(){ //no hace falta agregar las aristas en las dos direcciones! (
        en prim si)
9       ll cost=0;
10      sort(E.begin(), E.end());//ordenar aristas de menor a mayor -- OJO
            cuando ordena algo no necesariamente las cosas del mismo valor
            quedan en el mismo orden!!
11      uf.init(n);
12      forall(it, E){
13          if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
14              uf.join(it->a, it->b);//conectar
15              cost+=it->w;
16          }
17      }
18      return cost;
19  }
```

7.5.   Prim

```
1  bool taken[MAXN];
2  priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
3  void process(int v){
4      taken[v]=true;
5      forall(e, G[v])
6          if(!taken[e->second]) pq.push(*e);
7  }
8
9  ll prim(){
10     zero(taken);
11     process(0);
12     ll cost=0;
13     while(sz(pq)){
14         ii e=pq.top(); pq.pop();
15         if(!taken[e.second]) cost+=e.first, process(e.second);
16     }
17     return cost;
18 }
```

7.6.   2-SAT + Tarjan SCC

```
1  //We have a vertex representing a var and other for his negation.
2  //Every edge stored in G represents an implication. To add an equation of
       the form a||b, use addor(a, b)
3  //MAX=max cant var, n=cant var
4  #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
5  vector<int> G[MAX*2];
6  //idx[i]=index assigned in the dfs
7  //lw[i]=lowest index(closer from the root) reachable from i
8  int lw[MAX*2], idx[MAX*2], qidx;
9  stack<int> q;
10 int qcmp, cmp[MAX*2];
11 //verdad[cmp[i]]=valor de la variable i
12 bool verdad[MAX*2+1];
13
14 int neg(int x) { return x>=n? x-n : x+n;}
15 void tjn(int v){
16   lw[v]=idx[v]=++qidx;
17   q.push(v), cmp[v]=-2;
18   forall(it, G[v]){
19     if(!idx[*it] || cmp[*it]==-2){
20       if(!idx[*it]) tjn(*it);
21       lw[v]=min(lw[v], lw[*it]);
22     }
23   }
24   if(lw[v]==idx[v]){
25     int x;
26     do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
27     verdad[qcmp]=(cmp[neg(v)]<0);
28     qcmp++;
29   }
30 }
31 //remember to CLEAR G!!!
32 bool satisf(){//O(n)
33   memset(idx, 0, sizeof(idx)), qidx=0;
34   memset(cmp, -1, sizeof(cmp)), qcmp=0;
35   forn(i, n){
36     if(!idx[i]) tjn(i);
37     if(!idx[neg(i)]) tjn(neg(i));
38   }
39   forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
40   return true;
41 }
```

7.7.   Articulation Points

```
1  int N;
2  vector<int> G[1000000];
3  //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4  int qV, V[1000000], L[1000000], P[1000000];
5  void dfs(int v, int f){
6    L[v]=V[v]=++qV;
7    forall(it, G[v])
8      if(!V[*it]){
9        dfs(*it, v);
10       L[v] = min(L[v], L[*it]); //a todo lo que pueden llegar mis hijos yo
               tmb puede llegar
11       P[v]+= L[*it]>=V[v]; // no puede llegar a ningun vertice u / V[u] < V
               [v] => si saco v quedan desconectados => v punto de articulacion
12     }
13     else if(*it!=f) //backedge
14       L[v]=min(L[v], V[*it]);
15 }
16 int cantart(int N){ //O(n)
```

```
17    qV=0;
18    zero(V), zero(P);
19    dfs(0, -1);
20      P[0]--; //la raiz debe tener al menos dos hijos para ser punto de
              articulazion
21    int q=0;
22    forn(i, N) if(P[i]) q++;
23  return q;
24  }
```

7.8.   Comp. Biconexas y Puentes

```
1   const int MAXN=1010;
2   int n, m;
3   vector<int> G[MAXN];
4
5   struct edge {
6     int u,v, comp;
7     bool bridge;
8   };
9   vector<edge> e;
10  void addEdge(int u, int v) {
11    G[u].pb(sz(e)), G[v].pb(sz(e));
12    e.pb((edge){u,v,-1,false});
13  }
14  //V[i]=id de la dfs
15  //L[i]=lowest id reachable from i
16  int V[MAXN], L[MAXN], qV;
17  int nbc;//cant componentes
18  int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
19  void initDfs(int n) {
20    zero(G), zero(comp);
21    e.clear();
22    forn(i,n) V[i]=-1;
23    nbc = qV = 0;
24  }
25  stack<int> st;
26  void dfs(int u, int pe) {//O(n + m)
27      L[u] = V[u] = qV++;
28    comp[u] = (pe != -1);
29      for(auto &ne: G[u]) if (ne != pe){
30      int v = e[ne].u ^ e[ne].v ^ u; // x ^ y ^ x = y!
31      if (V[v] == -1) { // todavia no se lo visito
```

```
32        st.push(ne);
33        dfs(v,ne);
34        if (L[v] > V[u]){// bridge => no pertenece a ninguna comp biconexa
35          e[ne].bridge = true;
36        }
37        if (L[v] >= V[u]){ // art
38          int last;
39          do { //todas las aristas que estan entre dos puntos de articulacion
                  pertenecen a la misma componente biconexa
40            last = st.top(); st.pop();
41            e[last].comp = nbc;
42          } while (last != ne);
43          nbc++;
44          comp[u]++;
45        }
46        L[u] = min(L[u], L[v]);
47      }
48      else if (V[v] < V[u]) { // back edge
49        st.push(ne);
50        L[u] = min(L[u], V[v]);
51      }
52    }
53  }
54
55  set<int> C[2*MAXN];
56  int compnodo[MAXN];
57  int ptoart;
58  void blockcuttree(){
59    ptoart = 0;
60    forn(i,2*MAXN) C[i].clear();
61      for(auto &it: e){
62      int u = it.u, v = it.v;
63      if(comp[u] == 1) compnodo[u] = it.comp;
64        else{
65        if(compnodo[u] == 0){ compnodo[u] = nbc+ptoart; ptoart++;}
66        C[it.comp].insert(compnodo[u]);
67        C[compnodo[u]].insert(it.comp);
68      }
69      if(comp[v] == 1) compnodo[v] = it.comp;
70        else{
71        if(compnodo[v] == 0){ compnodo[v] = nbc+ptoart; ptoart++;}
72        C[it.comp].insert(compnodo[v]);
73        C[compnodo[v]].insert(it.comp);
```

```
74        }
75      }
76  }
77
78  int main() {
79    while(cin >> n >> m){
80      initDfs(n);
81      forn(i, m){
82        int a,b; cin >> a >> b;
83        addEdge(a,b);
84      }
85        dfs(0,-1);
86        forn(i, n) cout << "comp[" << i << "] = " <<  comp[i] << endl;
87      for(auto &ne: e) cout << ne.u << "->" << ne.v << " en la comp. " << ne.
            comp << endl;
88      cout << "Cant. de componentes biconexas = " << nbc << endl;
89    }
90      return 0;
91  }
```

7.9.   LCA + Climb

```
1   const int MAXN=100001;
2   const int LOGN=20;
3   //f[v][k] holds the 2^k father of v
4   //L[v] holds the level of v
5   int f[MAXN][LOGN], L[MAXN];
6   //call before build:
7   void dfs(int v, int fa=-1, int lvl=0){//generate required data
8     f[v][0]=fa, L[v]=lvl;
9     forall(it, G[v])if(*it!=fa)
10      dfs(*it, v, lvl+1);
11  }
12  void build(int N){//f[i][0] must be filled previously, O(nlgn)
13    forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
14
15  #define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
16
17  int climb(int a, int d){//O(lgn)
18    if(!d) return a;
19    dforn(i, lg(L[a])+1)
20      if(1<<i<=d)
21        a=f[a][i], d-=1<<i;
```

```
22      return a;
23  }
24  int lca(int a, int b){//O(lgn)
25    if(L[a]<L[b]) swap(a, b);
26    a=climb(a, L[a]-L[b]);
27    if(a==b) return a;
28    dforn(i, lg(L[a])+1)
29      if(f[a][i]!=f[b][i])
30        a=f[a][i], b=f[b][i];
31    return f[a][0];
32  }
33  int dist(int a, int b) {//returns distance between nodes
34    return L[a]+L[b]-2*L[lca(a, b)];}
```

7.10.   Heavy Light Decomposition

```
1   int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
2   int dad[MAXN];//dad[v]=padre del nodo v
3   void dfs1(int v, int p=-1){//pre-dfs
4     dad[v]=p;
5     treesz[v]=1;
6     forall(it, G[v]) if(*it!=p){
7       dfs1(*it, v);
8       treesz[v]+=treesz[*it];
9     }
10  }
11  //PONER Q EN 0  !!!!!
12  int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
13  //Las cadenas aparecen continuas en el recorrido!
14  int cantcad;
15  int homecad[MAXN];//dada una cadena devuelve su nodo inicial
16  int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
17  void heavylight(int v, int cur=-1){
18    if(cur==-1) homecad[cur=cantcad++]=v;
19    pos[v]=q++;
20    cad[v]=cur;
21    int mx=-1;
22    forn(i, sz(G[v])) if(G[v][i]!=dad[v])
23      if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
24    if(mx!=-1) heavylight(G[v][mx], cur);
25    forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
26      heavylight(G[v][i], -1);
27  }
```

```
28  //ejemplo de obtener el maximo numero en el camino entre dos nodos
29  //RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
30  //esta funcion va trepando por las cadenas
31  int query(int an, int v){//O(logn)
32    //si estan en la misma cadena:
33    if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
34    return max(query(an, dad[homecad[cad[v]]]),
35          rmq.get(pos[homecad[cad[v]]], pos[v]+1));
36  }
```

7.11.　Centroid Decomposition

```
1   int n;
2   vector<int> G[MAXN];
3   bool taken[MAXN];//poner todos en FALSE al principio!!
4   int padre[MAXN];//padre de cada nodo en el centroid tree
5
6   int szt[MAXN];
7   void calcsz(int v, int p) {
8     szt[v] = 1;
9     forall(it,G[v]) if (*it!=p && !taken[*it])
10      calcsz(*it,v), szt[v]+=szt[*it];
11  }
12  void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
13    if(tam==-1) calcsz(v, -1), tam=szt[v];
14    forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
15      {szt[v]=0; centroid(*it, f, lvl, tam); return;}
16    taken[v]=true;
17    padre[v]=f;
18    /*Analizar todos los caminos que pasan por este nodo:
19     * Agregar la informacion de cada subarbol
20     * Para cada subarbol:
21     * -sacar la informacion
22     * -analizar
23     * -agregar de nuevo la informacion
24     */
25    forall(it, G[v]) if(!taken[*it])
26      centroid(*it, v, lvl+1, -1);
27  }
```

7.12.　Euler Cycle

```
1   #define MAXN 1005
2   #define MAXE 1005005
```

```
3
4   int n,ars[MAXE], eq;
5   vector<int> G[MAXN];//fill G,ars,eq
6   list<int> path;
7   int used[MAXN]; //used[v] = i => para todo j<=i la arista v-G[v][j] fue
        usada y la arista v-G[v][i+1] no se uso
8   bool usede[MAXE];
9
10  //encuentra el ciclo euleriano, el grafo debe ser conexo y todos los nodos
        tener grado par para que exista
11  //para encontrar el camino euleriano conectar los dos vertices de grado
        impar y empezar de uno de ellos.
12
13  queue<list<int>::iterator> q;
14  int get(int v){
15    while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
16    return used[v];
17  }
18  void explore(int v, int r, list<int>::iterator it){
19    int ar=G[v][get(v)]; int u=v^ars[ar];
20    usede[ar]=true;
21    list<int>::iterator it2=path.insert(it, u);
22    if(u!=r) explore(u, r, it2);
23    if(get(v)<sz(G[v])) q.push(it);
24  }
25  void euler(int a){
26    zero(used), zero(usede);
27    path.clear();
28    q=queue<list<int>::iterator>();
29    path.push_back(a); q.push(path.begin());
30    while(sz(q)){
31      list<int>::iterator it=q.front(); q.pop();
32      if(used[*it]<sz(G[*it])) explore(*it, *it, it);
33    }
34    reverse(path.begin(), path.end());
35  }
36  void addEdge(int u, int v){
37    G[u].pb(eq), G[v].pb(eq);
38    ars[eq++]=u^v;
39  }
```

7.13.　Diametro árbol

```cpp
vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
int bfs(int r, int *d) {
  queue<int> q;
  d[r]=0; q.push(r);
  int v;
  while(sz(q)) { v=q.front(); q.pop();
    forall(it,G[v]) if (d[*it]==-1)
      d[*it]=d[v]+1, p[*it]=v, q.push(*it);
  }
  return v;//ultimo nodo visitado
}
vector<int> diams; vector<ii> centros;
void diametros(){
  memset(d,-1,sizeof(d));
  memset(d2,-1,sizeof(d2));
  diams.clear(), centros.clear();
  forn(i, n) if(d[i]==-1){
    int v,c;
    c=v=bfs(bfs(i, d2), d);
    forn(_,d[v]/2) c=p[c];
    diams.pb(d[v]);
    if(d[v]&1) centros.pb(ii(c, p[c]));
    else centros.pb(ii(c, c));
  }
}

int main() {
  freopen("in", "r", stdin);
  while(cin >> n >> m){
    forn(i,m) { int a,b; cin >> a >> b; a--, b--;
      G[a].pb(b);
      G[b].pb(a);
```

7.14.   Chu-liu

```cpp
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 {
```

```cpp
      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) {
```

```
52      forn(u,n) if (prev[u] >= 0) cost += mcost[u];
53      return cost;
54    }
55  }
56 }
```

7.15.   Hungarian

```
1  //Dado un grafo bipartito completo con costos no negativos, encuentra el
       matching perfecto de minimo costo.
2  tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar: cost=matriz de
       adyacencia
3  int n, max_match, xy[N], yx[N], slackx[N],prev2[N];//n=cantidad de nodos
4  bool S[N], T[N]; //sets S and T in algorithm
5  void add_to_tree(int x, int prevx) {
6    S[x] = true, prev2[x] = prevx;
7    forn(y, n) if (lx[x] + ly[y] - cost[x][y] < slack[y] - EPS)
8      slack[y] = lx[x] + ly[y] - cost[x][y], slackx[y] = x;
9  }
10 void update_labels(){
11   tipo delta = INF;
12   forn (y, n) if (!T[y]) delta = min(delta, slack[y]);
13   forn (x, n) if (S[x]) lx[x] -= delta;
14   forn (y, n) if (T[y]) ly[y] += delta; else slack[y] -= delta;
15 }
16 void init_labels(){
17   zero(lx), zero(ly);
18   forn (x,n) forn(y,n) lx[x] = max(lx[x], cost[x][y]);
19 }
20 void augment() {
21   if (max_match == n) return;
22   int x, y, root, q[N], wr = 0, rd = 0;
23   memset(S, false, sizeof(S)), memset(T, false, sizeof(T));
24   memset(prev2, -1, sizeof(prev2));
25   forn (x, n) if (xy[x] == -1){
26     q[wr++] = root = x, prev2[x] = -2;
27     S[x] = true; break; }
28   forn (y, n) slack[y] = lx[root] + ly[y] - cost[root][y], slackx[y] = root
       ;
29   while (true){
30     while (rd < wr){
31       x = q[rd++];
32       for (y = 0; y < n; y++) if (cost[x][y] == lx[x] + ly[y] && !T[y]){
33         if (yx[y] == -1) break; T[y] = true;
34         q[wr++] = yx[y], add_to_tree(yx[y], x); }
35       if (y < n) break; }
36     if (y < n) break;
37     update_labels(), wr = rd = 0;
38     for (y = 0; y < n; y++) if (!T[y] && slack[y] == 0){
39       if (yx[y] == -1){x = slackx[y]; break;}
40       else{
41         T[y] = true;
42         if (!S[yx[y]]) q[wr++] = yx[y], add_to_tree(yx[y], slackx[y]);
43       }}
44     if (y < n) break; }
45   if (y < n){
46     max_match++;
47     for (int cx = x, cy = y, ty; cx != -2; cx = prev2[cx], cy = ty)
48       ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
49     augment(); }
50 }
51 tipo hungarian(){
52   tipo ret = 0; max_match = 0, memset(xy, -1, sizeof(xy));
53   memset(yx, -1, sizeof(yx)), init_labels(), augment(); //steps 1-3
54   forn (x,n) ret += cost[x][xy[x]]; return ret;
55 }
```

7.16.   Dynamic Conectivity

```
1  struct UnionFind {
2      int n, comp;
3      vector<int> pre,si,c;
4      UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
5          forn(i,n) pre[i] = i; }
6      int find(int u){return u==pre[u]?u:find(pre[u]);}
7      bool merge(int u, int v) {
8          if((u=find(u))==(v=find(v))) return false;
9          if(si[u]<si[v]) swap(u, v);
10         si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
11         return true;
12     }
13     int snap(){return sz(c);}
14     void rollback(int snap){
15         while(sz(c)>snap){
16             int v = c.back(); c.pop_back();
17             si[pre[v]] -= si[v], pre[v] = v, comp++;
```

```
18            }
19        }
20  };
21  enum {ADD,DEL,QUERY};
22  struct Query {int type,u,v;};
23  struct DynCon {
24      vector<Query> q;
25      UnionFind dsu;
26      vector<int> match,res;
27      map<ii,int> last;//se puede no usar cuando hay identificador para cada
                arista (mejora poco)
28      DynCon(int n=0):dsu(n){}
29      void add(int u, int v) {
30          if(u>v) swap(u,v);
31          q.pb((Query){ADD, u, v}), match.pb(-1);
32          last[ii(u,v)] = sz(q)-1;
33      }
34      void remove(int u, int v) {
35          if(u>v) swap(u,v);
36          q.pb((Query){DEL, u, v});
37          int prev = last[ii(u,v)];
38          match[prev] = sz(q)-1;
39          match.pb(prev);
40      }
41      void query() {//podria pasarle un puntero donde guardar la respuesta
42          q.pb((Query){QUERY, -1, -1}), match.pb(-1);}
43      void process() {
44          forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz
                (q);
45          go(0,sz(q));
46      }
47      void go(int l, int r) {
48          if(l+1==r){
49              if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
50                  res.pb(dsu.comp);//aqui query=cantidad de componentes
                        conexas
51              return;
52          }
53          int s=dsu.snap(), m = (l+r) / 2;
54          forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v
                );
55          go(l,m);
56          dsu.rollback(s);
```

```
57          s = dsu.snap();
58          forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].
                v);
59          go(m,r);
60          dsu.rollback(s);
61      }
62  }dc;
```

8.   Network Flow

8.1.   Dinic

```
1
2   const int MAX = 300;
3   // Corte minimo: vertices con dist[v]>=0 (del lado de src) VS.  dist[v]==-1
        (del lado del dst)
4   // Para el caso de la red de Bipartite Matching (Sean V1 y V2 los conjuntos
        mas proximos a src y dst respectivamente):
5   // Reconstruir matching: para todo v1 en V1 ver las aristas a vertices de
        V2 con it->f>0, es arista del Matching
6   // Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con
        dist[v]>0
7   // Max Independent Set: tomar los vertices NO tomados por el Min Vertex
        Cover
8   // Max Clique: construir la red de G complemento (debe ser bipartito!) y
        encontrar un Max Independet Set
9   // Min Edge Cover: tomar las aristas del matching + para todo vertices no
        cubierto hasta el momento, tomar cualquier arista de el
10  //Complejidad:
11  //Peor caso: O(V^2E)
12  //Si todas las capacidades son 1: O(min(E^1/2,V^2/3)E)
13  //Para matching bipartito es: O(sqrt(V)E)
14
15  int nodes, src, dst; // Setear estos
16  vector<int> dist, q, work; // inicializar de tamaño n
17  struct Edge {
18      int to, rev;
19      tint f, cap;
20      Edge(int to, int rev, tint f, tint cap) : to(to), rev(rev), f(f), cap(
            cap) {}
21  };
22  vector<vector<Edge>> G; // inicializar de tamaño n
23  void addEdge(int s, int t, tint cap){
```

```
24    G[s].pb(Edge(t, G[t].size(), 0, cap)); G[t].pb(Edge(s, G[s].size(), 0,
          0));}
25  bool dinic_bfs(){
26      for(auto & c : dist) c = -1;
27      dist[src]=0;
28      int qt=0; q[qt++]=src;
29      for(int qh=0; qh<qt; qh++){
30          int u =q[qh];
31          for(auto &e : G[u]){
32              int v=e.to;
33              if(dist[v]<0 && e.f < e.cap)
34                  dist[v]=dist[u]+1, q[qt++]=v;
35          }
36      }
37      return dist[dst]>=0;
38  }
39  tint dinic_dfs(int u, tint f){
40      if(u==dst) return f;
41      for(int &i=work[u]; i<G[u].size(); i++){
42          Edge &e = G[u][i];
43          if(e.cap<=e.f) continue;
44          int v=e.to;
45          if(dist[v]==dist[u]+1){
46              tint df=dinic_dfs(v, min(f, e.cap-e.f));
47              if(df>0){
48                  e.f+=df; G[v][e.rev].f-= df;
49                  return df;  }
50          }
51      }
52      return 0;
53  }
54  tint maxFlow(int _src, int _dst){
55      src=_src, dst=_dst;
56      tint result=0;
57      while(dinic_bfs()){
58          for(auto & c : work) c = 0;
59          while(tint delta=dinic_dfs(src,INF))
60              result+=delta;
61      }
62      // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1 forman
          el min-cut
63      return result; }
```

8.2. Konig

```
1  // asume que el dinic YA ESTA tirado
2  // asume que nodes-1 y nodes-2 son la fuente y destino
3  int match[maxnodes]; // match[v]=u si u-v esta en el matching, -1 si v no
       esta matcheado
4  int s[maxnodes]; // numero de la bfs del koning
5  queue<int> kq;
6  // s[e] %2==1  o si e esta en V1 y s[e]==-1-> lo agarras
7  void koning() {//O(n)
8    forn(v,nodes-2) s[v] = match[v] = -1;
9    forn(v,nodes-2) forall(it,g[v]) if (it->to < nodes-2 && it->f>0)
10     { match[v]=it->to; match[it->to]=v;}
11   forn(v,nodes-2) if (match[v]==-1) {s[v]=0;kq.push(v);}
12   while(!kq.empty()) {
13     int e = kq.front(); kq.pop();
14     if (s[e] %2==1) {
15       s[match[e]] = s[e]+1;
16       kq.push(match[e]);
17     } else {
18
19       forall(it,g[e]) if (it->to < nodes-2 && s[it->to]==-1) {
20         s[it->to] = s[e]+1;
21         kq.push(it->to);
22       }
23     }
24   }
25  }
```

8.3. Edmonds Karp's

```
1  #define MAX_V 1000
2  #define INF 1e9
3  //special nodes
4  #define SRC 0
5  #define SNK 1
6  const tint N = !!!;
7  vector<map<int, int>> G;//inicializar de tamaño n
8  //To add an edge use
9  void add(int a, int b, int w) {G[a][b]=w;}
10 int f; vector<int> p;
11 void augment(int v, int minE){
12   if(v==SRC) f=minE;
```

```
13      else if(p[v]!=-1){
14        augment(p[v], min(minE, G[p[v]][v]));
15        G[p[v]][v]-=f, G[v][p[v]]+=f;
16      }
17  }
18  tint maxflow(){//O(VE^2)
19    tint Mf=0;
20    do{
21      f=0;
22      vector<char> used(N); queue<int> q; q.push(SRC);
23      p = vector<char>(N, -1);
24      while(sz(q)){
25        int u=q.front(); q.pop();
26        if(u==SNK) break;
27        for(pp : G[u])
28          if(pp.snd>0 && !used[pp.fst])
29            used[pp.fst]=true, q.push(pp.fst), p[pp.fst]=u;
30      }
31      augment(SNK, INF);
32      Mf+=f;
33    }while(f);
34    return Mf;
35  }
```

8.4. Push-Relabel O(N3)

```
1   #define MAX_V 1000
2   int N;//valid nodes are [0...N-1]
3   #define INF 1e9
4   //special nodes
5   #define SRC 0
6   #define SNK 1
7   map<int, int> G[MAX_V];
8   //To add an edge use
9   #define add(a, b, w) G[a][b]=w
10  ll excess[MAX_V];
11  int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
12  queue<int> Q;
13  void enqueue(int v) {
14    if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
15  void push(int a, int b) {
16    int amt = min(excess[a], ll(G[a][b]));
17    if(height[a] <= height[b] || amt == 0) return;
```

```
18    G[a][b]-=amt, G[b][a]+=amt;
19    excess[b] += amt, excess[a] -= amt;
20    enqueue(b);
21  }
22  void gap(int k) {
23    forn(v, N){
24      if (height[v] < k) continue;
25      count[height[v]]--;
26      height[v] = max(height[v], N+1);
27      count[height[v]]++;
28      enqueue(v);
29    }
30  }
31  void relabel(int v) {
32    count[height[v]]--;
33    height[v] = 2*N;
34    forall(it, G[v])
35      if(it->snd)
36        height[v] = min(height[v], height[it->fst] + 1);
37    count[height[v]]++;
38    enqueue(v);
39  }
40  ll maxflow() {//O(V^3)
41    zero(height), zero(active), zero(count), zero(excess);
42    count[0] = N-1;
43    count[N] = 1;
44    height[SRC] = N;
45    active[SRC] = active[SNK] = true;
46    forall(it, G[SRC]){
47      excess[SRC] += it->snd;
48      push(SRC, it->fst);
49    }
50    while(sz(Q)) {
51      int v = Q.front(); Q.pop();
52      active[v]=false;
53      forall(it, G[v]) push(v, it->fst);
54      if(excess[v] > 0)
55        count[height[v]] == 1? gap(height[v]):relabel(v);
56    }
57    ll mf=0;
58    forall(it, G[SRC]) mf+=G[it->fst][SRC];
59    return mf;
60  }
```

8.5. Min-cost Max-flow

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

9. Template

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forsn(i,s,n) for(tint i=(tint)(s); i < (tint)(n); i++)
#define forn(i,n) forsn(i,0,n)
#define dforsn(i,s,n) for(tint i=(tint)(n)-1; i >= (tint)(s); i--)
#define dforn(i,n) dforsn(i,0,n)
#define pb push_back
#define mp make_pair
#define fst first
#define snd second
typedef long long tint;
#define sz(C) ((tint) C.size())

#ifdef DEBUG
#define debug(v) cerr << #v << " = " << (v) << endl;
#else
#define debug(v)
#endif

int main() {
    ios::sync_with_stdio(0); cin.tie(0);



    return 0;
}
```

10.   Ayudamemoria

Cant. decimales

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

Rellenar con espacios(para justificar)

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

Leer hasta fin de linea

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

Aleatorios

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

Doubles Comp.

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

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

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

Mejorar velocidad 2

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

Expandir pila

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

C++0x / C++11

```
g++ -std=c++0x o g++ -std=c++11
```

Leer del teclado

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

Iterar subconjunto

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

File setup

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

Excepción en fuera de indices

```
1  g++ -g -D_GLIBCXX_DEBUG(_PEDANTIC)
```