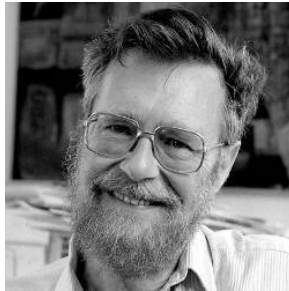


# Repositorio en C++

Universidad de la Amazonia, Colombia.

19 de noviembre de 2018



I2

## Índice

<b>1. Estructuras de datos</b>	<b>2</b>	<b>3. Programacion dinamica</b>	<b>10</b>
1.1. Tablas aditivas	2	3.1. Subconjuntos de un conjunto	10
1.2. Disjoint set union find	2	3.2. Problema de la mochila	10
1.3. Union find con compresion de caminos	3	3.3. Longest Increment Subsequence	10
1.4. Segment tree	3	3.4. Max Range Sum	11
1.5. Segment tree con lazy propagation	4	3.5. Subset Sum	11
1.6. Arbol binario indexado	5	3.6. Traveling salesman problem	11
1.7. Sparse table	5		
<b>2. Grafos</b>	<b>6</b>	<b>4. Otros</b>	<b>11</b>
2.1. Dijkstra	6	4.1. Busqueda binaria	11
2.2. Bellman-Ford	6	4.2. Raiz babilonica	12
2.3. Floyd Warshall	7	4.3.Codigo gray	12
2.4. Kosaraju	7	4.4. Lowest Common Ancestor	12
2.5. Tarjan	8		
2.6. Kruskal	8	<b>5. Matematicas</b>	<b>13</b>
2.7. Prim	8	5.1. MCD y MCM	13
2.8. Topological sort	9	5.2. Exponenciacion binaria	13
2.9. Puntos de articulaci3n y puentes	9	5.3. Algoritmo extendido de euclides	13
		5.4. Inverso multiplicativo modular	14
		5.5. Phi de euler	14
		5.6. Multiplicacion modular	14
		5.7. Exponenciacion modular	14
		5.8. Test de Rabin Miller	15
		5.9. Rho de pollard	15
		5.10. factorizacion con criba	15
		5.11. BigInteger c++	16
		5.12. Fraccion	19
		<b>6. Cadenas</b>	<b>19</b>
		6.1. Algoritmo de bordes	19
		6.2. KMP	19

<b>7. Geometria</b>	<b>20</b>
7.1. Punto . . . . .	20
7.2. Linea . . . . .	20
7.3. Vector . . . . .	21
<b>8. Tips and formulas(UFPS, 2017)</b>	<b>22</b>
8.1. ASCII Table . . . . .	22
8.2. Formulas . . . . .	23
8.3. Sequences . . . . .	25
8.4. Time Complexities . . . . .	26
<b>9. Extras</b>	<b>26</b>
9.1. Template . . . . .	26
9.2. Formulas extra . . . . .	27
9.3. Secuencias . . . . .	27

## 1. Estructuras de datos

### 1.1. Tablas aditivas

Construccion  $O(n)$

```

1 void build(){
2     memset(memo, 0, sizeof(memo));
3     memo[1][1] = tab[0][0];
4     for (int i = 2; i <= fila; i++)
5         memo[i][1] = memo[i-1][1] + tab[i - 1][0];
6     for (int j = 2; j <= col; j++)
7         memo[1][j] = memo[1][j-1] + tab[0][j - 1];
8
9     for (int i = 2; i <= col; i++)
10        for (int j = 2; j <= col; j++)
11            memo[i][j] = memo[i][j - 1] + memo[i - 1][j] +
12                tab[i - 1][j - 1] - memo[i - 1][j - 1];
13 }
14 //indexando desde 1
15 int query(int f1, int c1, int f2, int c2){
16     return memo[f2][c2] - memo[f1-1][c2] -
17         memo[f2][c1-1] + memo[f1-1][c1-1];
18 }
```

### 1.2. Disjoint set union find

Construccion  $O(n)$   
asocia elementos en conjuntos de arboles.

```

1 struct union_find{
2     int padre[100], rango[100];
3     vector<int> grupo[100];
4
5     void iniciar(int n){
6         for (int i = 0; i < n; i++) {
7             padre[i] = i;
8             rango[i] = 0;
9             grupo[i].clear();
10            grupo[i].push_back(i);
11        }
12    }
13
14    int raiz(int x){
15        if(padre[x] == x) return x;
16        return raiz(padre[x]);
17    }
18
19    void unir(int x, int y){
20        x = raiz(x);
21        y = raiz(y);
22        if(x == y) return;
23
24        if(rango[x] > rango[y]){
25            padre[y] = x;
26            grupo[x].insert(grupo[x].begin(), grupo[y].begin(), grupo
27                [y].end());
28            grupo[y].clear();
29            return;
30        }
31        padre[x] = y;
32        grupo[y].insert(grupo[y].begin(), grupo[x].begin(), grupo[x
33            ].end());
34        grupo[x].clear();
35        if(rango[y] == rango[x]) rango[y]++;
36    }
}
```

```

37 bool MismoGrupo(int x, int y){
38     return raiz(x) == raiz(y);
39 }
40
41 void grupo_n(int n){
42     cout << "elementos en el grupo de " << n << endl;
43     n = raiz(n);
44     for(int i = 0; i < grupo[n].size(); i++) cout << grupo[n]
45         [i] << " ";
46     cout << endl;
47 };

```

### 1.3. Union find con compresion de caminos

asocia elementos de manera simple  
metodo mismoGrupo es el mismo del union-find normal.

```

1 struct union_find{
2     int padre[MAX];
3
4     void iniciar(int n){
5         for (int i = 0; i < n; i++) padre[i] = i;
6     }
7
8     int raiz(int x){
9         if(x == padre[x]) return x;
10        else return padre[x] = raiz(padre[x]);
11    }
12
13    void unir(int x, int y){
14        padre[raiz(x)] = raiz(y);
15    }
16 };

```

### 1.4. Segment tree

Ejemplo de RMQ (Range Minium Query)  
Contruccion  $O(n)$   
Consulta  $O(\log n)$   
Update  $O(\log n)$

```

1 const int MAX = 4 * 1000; //poner 4 * longitud maxima
2
3 struct segment_tree{
4     int st[MAX];
5     vi A;
6     int n, tamst;
7
8     int mov_izq(int index){ return index << 1; }
9     int mov_der(int index){ return (index << 1) + 1; }
10
11    void construir(int pos, int izq, int der){
12        if(izq == der){
13            st[pos] = A[der];
14            return;
15        }
16
17        construir(mov_izq(pos), izq, (izq + der) >> 1);
18        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
19        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
20        st[pos] = min(st[aux1], st[aux2]);
21    }
22
23    void iniciar(vi arr){ //metodo a invocar
24        A = arr;
25        n = A.size();
26        tamst = n << 2;
27        construir(1, 0, n - 1);
28    }
29
30    int query(int pos, int izq, int der, int i, int j){
31        if(i > der || j < izq) return -1;
32        if(i <= izq && j >= der) return st[pos];
33
34        int aux1 = query(mov_izq(pos), izq, (izq + der) >> 1, i
35            , j);
36        int aux2 = query(mov_der(pos), ((izq + der) >> 1) + 1,
37            der, i, j);
38        if(aux1 == -1) return aux2;
39        if(aux2 == -1) return aux1;
40
41        return min(aux1, aux2);

```

```

40     }
41
42     int RMQ(int i, int j){//metodo a invocar
43         return query(1, 0, n-1, i, j);
44     }
45
46     int cambiar(int pos, int izq, int der, int index, int nuevo)
47     ){
48         if(index > der || index < izq) return st[pos];
49         if(der == index && izq == index){
50             A[index] = nuevo;
51             return st[pos] = nuevo;
52         }
53
54         int aux1 = cambiar(mov_izq(pos), izq, (izq + der) >> 1,
55                             index, nuevo);
56         int aux2 = cambiar(mov_der(pos), ((izq + der) >> 1) +
57                             1, der, index, nuevo);
58         return st[pos] = min(aux1, aux2);
59     }
60
61     int update(int index, int num){//metodo a invocar
62         return cambiar(1, 0, n-1, index, num);
63     }
64 };

```

## 1.5. Segment tree con lazy propagation

Permite actualizar rangos del arbol en  $O(\log n)$ . solo estan los metodos nuevos y los que hay que actualizar, lo demas es lo mismo del segment tree normal.

```

1     int lazy[MAX];
2
3     void construir(int pos, int izq, int der){
4         lazy[pos] = -1;//reiniciar lazy
5         if(izq == der){
6             st[pos] = A[der];
7             return;
8         }
9
10        construir(mov_izq(pos), izq, (izq + der) >> 1);

```

```

11        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
12        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
13        st[pos] = min(st[aux1], st[aux2]);
14    }
15
16    int query(int pos, int izq, int der, int i, int j){
17        if(i > der || j < izq) return -1;
18        solve_lazy(pos, izq, der);//resolver algun lazy
19        pendiente
20        if(i <= izq && j >= der) return st[pos];
21
22        int aux1 = query(mov_izq(pos), izq, (izq + der) >> 1, i,
23                        j);
24        int aux2 = query(mov_der(pos), ((izq + der) >> 1) + 1,
25                        der, i, j);
26        if(aux1 == -1) return aux2;
27        if(aux2 == -1) return aux1;
28
29        return min(aux1, aux2);
30    }
31
32    void solve_lazy(int pos, int izq, int der){//resolver lazy
33        if(lazy[pos] == -1) return;
34
35        st[pos] = lazy[pos];
36        if(izq != der){
37            lazy[mov_izq(pos)] = lazy[mov_der(pos)] = lazy[pos];
38        }
39        lazy[pos] = -1;
40    }
41
42    int lazy_propagation(int pos, int izq, int der, int i, int
43                        j, int nuevo){
44        solve_lazy(pos, izq, der);
45        if(i > der || j < izq) return st[pos];
46
47        if(i <= izq && j >= der){
48            lazy[pos] = nuevo;
49            solve_lazy(pos, izq, der);
50            return st[pos];

```

```

47     }
48
49     int aux1 = lazy_propagation(mov_izq(pos), izq, (izq +
50     der) >> 1, i, j, nuevo);
51     int aux2 = lazy_propagation(mov_der(pos), ((izq + der)
52     >> 1) + 1, der, i, j, nuevo);
53     return st[pos] = min(aux1, aux2);
54 }
55
56 int update(int i, int j, int nuevo){//metodo a invocar
    return lazy_propagation(1, 0, n-1, i, j, nuevo);//
    propagar lazy
}

```

## 1.6. Arbol binario indexado

Arbol de Fenwick, estructura para el RSM(Range Sum Query)

Construcción  $O(n \log n)$

Consulta  $O(\log k)$

Update  $O(\log n)$

```

1 struct FenwickTree{
2     vi ft;
3
4     void construir(int n){//indexamos desde 1
5         ft.assign(n + 1, 0);
6     }
7
8     void construir(vi &v){
9         ft.assign(v.size() + 1, 0);
10        for(int i = 1; i <= v.size(); i++)
11            actualizar(i, v[i - 1]);
12    }
13
14    int lsOne(int n){//bit menos significativo en 1
15        return n & (-n);
16    }
17
18    int rsq(int i){//suma de 1 hasta i
19        int acum = 0;
20        for(; i; i -= lsOne(i)) acum+=ft[i];
21        return acum;

```

```

22    }
23
24    int rsq(int i, int j){//suma de i hasta j
25        return rsq(j) - ((i==1)? 0: rsq(i - 1));
26    }
27
28    void actualizar(int pos, int n){//n = nuevo - anterior
29        for(; pos < ft.size(); pos += lsOne(pos))
30            ft[pos] += n;
31    }
32 };

```

## 1.7. Sparse table

Para RMQ (Range Minium Query) en arreglos estaticos

Construcción  $O(n \log n)$

Consulta  $O(1)$

```

1 #define MAX 1000 //n
2 #define Log2 10 //2^10 > 1000
3
4 struct sparseTable{
5     int arr[MAX], spt[MAX][Log2] = {0};
6
7     sparseTable(int n, int a[]){
8         for(int i = 0; i < n; i++){
9             arr[i] = a[i];
10            spt[i][0] = i;
11        }
12
13        for(int j = 1; (1<<j) <= n; j++){
14            for(int i=0; i+(1<<j)-1 < n; i++){
15                if(arr[spt[i][j-1]] < arr[spt[i+(1<<(j-1))][j-1]])
16                    spt[i][j] = spt[i][j-1];
17                else
18                    spt[i][j] = spt[i+(1<<(j-1))][j-1];
19            }
20        }
21
22        int query(int i, int j){//de i hasta j, index desde 0
23            int k = (int) floor(log(((j-i+1)*1.0))/log(2.0));

```

```

24         if(arr[spt[i][k]] <= arr[spt[j-(1<<k)+1][k]])
25             return spt[i][k];
26         else return spt[j-(1<<k)+1][k];
27     }
28 };

```

## 2. Grafos

### 2.1. Dijkstra

Ruta minima  $O((n + m)\log n)$

```

1  vi padre;//opcional, usar cuando se necesite el camino.
2
3  vi dijkstra(vvii &grafo, int nodo, int tam){
4      padre.assign(tam + 1, -1);
5      vi dis(tam + 1, inf);
6      priority_queue<ii> cola;
7      cola.push(ii(-0, nodo));
8      int peso, aux;
9      ii par, par2;
10
11     while(cola.size()){
12         par = cola.top();//peso, nodo
13         cola.pop();
14         peso = -par.first;
15         nodo = par.second;
16
17         if(dis[nodo] <= peso) continue;
18         dis[nodo] = peso;
19
20         for(int i = 0; i < grafo[nodo].size(); i++){
21             par2 = grafo[nodo][i];
22             aux = dis[nodo] + par2.first;
23             if(dis[par2.second] > aux){
24                 cola.push(ii(-aux, par2.second));
25                 padre[par2.second] = nodo;
26             }
27         }
28     }
29 }

```

```

30     return dis;
31 }
32
33 void camino(int n){//imprimir el camino
34     if(padre[n] == -1) printf("%d", n);
35     else{
36         camino(padre[n]);
37         printf("─%d", n);
38     }
39 }

```

### 2.2. Bellman-Ford

Ruta minima con pesos negativos  $O(n^2)$

```

1  vector<iii> grafo; //lista de incidencia
2
3  bool BellmanFord(vector<iii> &lista, int nodos, int inicio,
4      vector<int> &dis){
5      dis.assign(nodos, inf);
6      dis[inicio] = 0;
7      int aux;
8
9      for (int i = 0; i < nodos; i++)
10         for (int j = 0; j < lista.size(); j++) {
11             aux = dis[lista[j].second.first] + lista[j].first;
12             if (dis[lista[j].second.second] > aux){
13                 dis[lista[j].second.second] = aux;
14             }
15         }
16
17         for(int j = 0; j < lista.size(); j++){
18             aux = dis[lista[j].second.first] + lista[j].first;
19             if(dis[lista[j].second.second] > aux)
20                 return false;//existe ciclo!!!
21         }
22     return true;
23 }

```

## 2.3. Floyd Warshall

Ruta minima de toda la matriz, recomendable si  $n \leq 100$   
 $O(n^3)$

```
1 int cam[10][10], matriz[10][10];
2
3 void imprimirCamino(int f, int c){
4     if(cam[f][c] == f){
5         printf("%d", f);
6         return;
7     }else{
8         imprimirCamino(f, cam[f][c]);
9         printf("_%d", cam[f][c]);
10    }
11 }
12
13 void FloydWarshall(int nodos){
14     int aux;
15     //for(int i = 0; i < nodos; i++) matriz[i][i] = 0;//sin
        caminos
16     for(int i = 0; i < nodos; i++){
17         for(int j = 0; j < nodos; j++){
18             if(i == j) matriz[i][j] = 0;
19             if(i != j && matriz[i][j] != inf) cam[i][j] = i;
20         }
21     }
22
23     for(int k = 0; k < nodos; k++)
24         for(int i = 0; i < nodos; i++)
25             for(int j = 0; j < nodos; j++){
26                 aux = matriz[i][k] + matriz[k][j];
27                 if(matriz[i][j] > aux){
28                     matriz[i][j] = aux;
29                     cam[i][j] = cam[k][j];
30                 }
31             }
32 }
```

## 2.4. Kosaraju

Componentes fuertemente conexas grafos si y no dirigidos  
 $O(2(n + m))$

```
1 int n, m;
2 vector<vi> grafo(100), transpuesto(100), comp;
3 stack<int> pila;
4 bool vis[100];
5
6 void dfs(int n, vector<vi> lista, bool f, vi &grupo){
7     vis[n] = true;
8     if(!f) grupo.push_back(n);
9
10    for (int i = 0; i < lista[n].size(); i++)
11        if (!vis[lista[n][i]]) dfs(lista[n][i], lista, f, grupo);
12    if(f) pila.push(n);
13 }
14
15 void kosaraju(){
16     memset(vis, false, sizeof(vis));
17     vi no_se_utiliza;
18     for (int i = 0; i < n; i++)
19         if(!vis[i]) dfs(i, grafo, true, no_se_utiliza);
20
21     memset(vis, false, sizeof(vis));
22     int n;
23     while(pila.size()){
24         n = pila.top(); pila.pop();
25         if (!vis[n]){
26             vi vec;
27             dfs(n, transpuesto, false, vec);
28             comp.push_back(vec);
29         }
30     }
31
32     for (int i = 0; i < comp.size(); i++){
33         for (int j = 0; j < comp[i].size(); j++) cout << comp[i][j]
34             << "_";
35         cout << endl;
36     }
```

## 2.5. Tarjan

Componentes fuertemente conexas grafos si y no dirigidos, requiere menos espacio que Kosajaru  
 $O(n + m)$

```
1 vi dfs_low, dfs_num, s; vector<bool> vis;
2 int dfsCont;
3
4 void dfs(int u){
5     dfs_low[u] = dfs_num[u] = dfsCont++;
6     s.push_back(u); vis[u] = true;
7
8     for(int i = 0; i < lista[u].size(); i++){
9         int aux = lista[u][i];
10        if(dfs_num[aux] == -1) dfs(aux);
11        if(vis[aux])
12            dfs_low[u] = min(dfs_low[u], dfs_low[aux]);
13    }
14
15    if(dfs_low[u] == dfs_num[u]){
16        printf("comp:\n");
17        while(true){
18            int v = s.back(); s.pop_back();
19            printf("_%d\n", v); vis[v] = false;
20            if(v == u) break;
21        }
22        printf("\n");
23    }
24 }
25
26 void tarjan(){
27     dfs_num.assign(n+1,-1); dfs_low.assign(n+1,0);
28     vis.assign(n+1, false); dfsCont = 0;
29     for(int i = 0; i < n; i++)
30         if(dfs_num[i] == -1) dfs(i);
31 }
```

## 2.6. Kruskal

Arbol generador minimo(MST), se necesita de un union-find  
 $O(m \log n)$ , sin contar el ordenamiento.

```
1 typedef pair<int, ii> iii; //peso, origen y destino
2 vector<iii> listaInc; //lista de incidencia
3 union_find arbol;
4
5 int kruskal(vector<iii> lista, int nodos, union_find &uf){
6     sort(lista.begin(), lista.end());
7     uf.iniciar(nodos);
8     int acum = 0, ejes = 0, n = nodos - 1;
9
10    for (int i = 0; i < lista.size(); i++) {
11        if (!uf.MismoGrupo(lista[i].second.first,
12                            lista[i].second.second)) {
13            ejes++;
14            uf.unir(lista[i].second.first, lista[i].second.second);
15            acum += lista[i].first;
16            if(ejes == n) return acum;
17        }
18    }
19    return -1;
20 }
```

## 2.7. Prim

Arbol generador minimo (MST)  
 $O(m \log n)$

```
1 priority_queue<ii> cola;
2 vector<bool> vis;
3
4 void vecinos(vvii &lista, int nodo){
5     vis[nodo] = true;
6     for(int i = 0; i < lista[nodo].size(); i++){
7         ii par = lista[nodo][i]; //peso - destino
8         if(!vis[par.second])
9             cola.push(ii(-par.first, -par.second));
10    }
11 }
12
13 int prim(vvii &lista, int n){
14     vis.assign(n + 1, false);
15     vecinos(lista, 1);
16     int acum = 0; ii par;
```



```

17
18 while(cola.size()){
19     par = cola.top(); cola.pop();
20     if(vis[-par.second]) continue;
21     acum += -par.first;
22     vecinos(lista, -par.second);
23 }
24 return acum;
25 }

```

## 2.8. Topological sort

$O(n + m)$ , algoritmo de kahn.

```

1 vector<int> res;//guarda la respuesta.
2 vector<int> ent;//se debe llenar con la cantidad de
3 //aristas entrantes que tiene cada nodo.
4
5 void topological_sort(vvi &lis, int tam){
6     res.clear();
7     queue<int> s;
8     for(int i = 1; i <= tam; i++){
9         if(!ent[i]) s.push(i);
10    }
11
12    int n, m;
13    while(s.size()){
14        n = s.front();
15        s.pop();
16        res.push_back(n);
17
18        for(int i = 0; i < lis[n].size(); i++){
19            m = lis[n][i];
20            ent[m]--;
21            if(!ent[m]) s.push(m);
22        }
23    }
24 }

```

## 2.9. Puntos de articulación y puentes

$O(n + m)$ .

```

1 vi puntos, dfs_num, dfs_low, padre;
2 int n, m, dfsCont, root, dfsRoot;
3 vector<ii> puentes;//guarda los puentes
4
5 void dfs(int u){
6     dfs_low[u] = dfs_num[u] = dfsCont++;
7     int aux;
8     for(int i = 0; i < lista[u].size(); i++){
9         aux = lista[u][i];
10        if(dfs_num[aux] == -1){
11            padre[aux] = u;
12            if(u == dfsRoot) root++;
13            dfs(aux);
14
15            if(dfs_low[aux] >= dfs_num[u]) puntos[u]++;
16            if(dfs_low[aux] > dfs_num[u])
17                puentes.push_back(ii(aux, u));
18            dfs_low[u] = min(dfs_low[u], dfs_low[aux]);
19        }else if(aux != padre[u])
20            dfs_low[u] = min(dfs_low[u], dfs_num[aux]);
21    }
22 }
23
24 void solve(){
25     puntos.assign(n, 1); dfs_low.assign(n, 0);
26     padre.assign(n, 0); dfs_num.assign(n, -1);
27
28     for(int i = 0; i < n; i++)
29         if(dfs_num[i] == -1){
30             dfsCont = root = 0; dfsRoot = i;
31             dfs(dfsRoot);
32             puntos[i] = root - 1;
33         }
34
35     printf("puntos_de_articulacion:\n");
36     for(int i = 0; i < n; i++)
37         if(puntos[i] > 1)//cantidad de componentes
38             printf("%d, conecta %d comp.\n", i, puntos[i]);
39 }

```

### 3. Programacion dinamica

#### 3.1. Subconjuntos de un conjunto

$O(2^n)$

```
1 void mask(int n, int ar[]){
2     int l = 1 << n;
3
4     for(int i = 0; i < l; i++){
5         for(int j = 0; j < n; j++){
6             if(i & (1 << j)){
7                 printf("%d_", ar[j]);
8             }
9         }
10        printf("\n");
11    }
12 }
```

#### 3.2. Problema de la mochila

```
1 int ganancia[100] = {100, 70, 50, 10};
2 int peso[100] = {10, 4, 6, 12};
3
4 int knapsack(int cap, int n) { //capacidad y cantidad.
5     int dp[n+1][cap+1];
6     for(int i = 0; i <= n; i++) //recorrer objetos
7         for(int j = 0; j <= cap; j++){
8             if(i == 0 || j == 0) dp[i][j] = 0; //caso base
9             else if(peso[i - 1] <= j)
10                 dp[i][j] = max(dp[i - 1][j],
11                                ganancia[i - 1] + dp[i - 1][j -
12                                    peso[i - 1]]);
13             else
14                 dp[i][j] = dp[i - 1][j];
15         }
16     return dp[n][cap];
17 }
```

#### 3.3. Longest Increment Subsequence

Subsecuencia creciente mas larga, solución corta con dp  
 $O((n*(n+1))/2)$

```
1 int LIS_dp(){
2     int res = 0;
3     vector<int> vec(8, 1);
4
5     for(int i = 0; i < 8; i++){
6         for(int j = i + 1; j < 8; j++){
7             if(A[i] < A[j]) vec[j] = max(vec[j], vec[i] + 1);
8             res = max(res, vec[i]);
9         }
10    }
11    return res;
12 }
```

Solución D&C con greedy,  $O(n \log n)$

```
1 int A[] = {-7, 10, 9, 2, 3, 8, 8, 6};
2 int aux[10], lis[10], indexAnt[10], n = 8;
3
4 void mostrar(int pos){
5     stack<int> pila;
6     while(pos != -1)
7         pila.push(pos), pos = lis[pos];
8
9     while(pila.size()){
10        printf("%d\n", A[pila.top()]);
11        pila.pop();
12    }
13 }
14 //Para decreciente invertir el signo de los numeros
15 void LIS(){ //en el arreglo.
16     int tam = 0, pos, res = 0;
17     for(int i = 0; i < n; i++){
18         pos = lower_bound(aux, aux + tam, A[i]) - aux;
19         //usar upper_bound para contar repetidos
20         aux[pos] = A[i];
21         indexAnt[pos] = i;
22         lis[i] = pos;
23         lis[i] = pos? indexAnt[pos-1]: -1;
24         if(pos + 1 > tam){
25             tam = pos + 1;
26             res = i;
27         }
28     }
29 }
```

```

29 |
30 |     printf("longitud: %d\n", tam);
31 |     mostrar(res);
32 | }

```

### 3.4. Max Range Sum

Algoritmo de Kadane,  $O(n)$

```

1 | int main(){
2 |     int n, num, res, aux;
3 |
4 |     while(scanf("%d", &n), n){
5 |         res = aux = 0;
6 |         for(int i = 0; i < n; i++){
7 |             scanf("%d", &num);
8 |             aux += num;
9 |             res = max(aux, res);
10 |            if(aux < 0) aux = 0;
11 |        }
12 |
13 |        if(res > 0) printf("MRS: %d\n", res);
14 |        else printf("negativo.\n");
15 |    }
16 |    return 0;
17 | }

```

### 3.5. Subset Sum

```

1 | bool dp[5][50]; //fila cantidad de numeros
2 | //columnas rango maximo a evaluar
3 |
4 | void pre(vi &num){
5 |     memset(dp, false, sizeof(dp));
6 |
7 |     for(int i = 0; i < num.size(); i++){
8 |         if(i) for(int j = 1; j < 50; j++){
9 |             if(dp[i - 1][j]) dp[i][j + num[i]] = true;
10 |
11 |             dp[i][num[i]] = true;
12 |         }

```

```

13 | }

```

### 3.6. Traveling salesman problem

$O(2^n * n^2)$ , para la respuesta llamar: tsp(0,1)

```

1 | int MAX; //luego de leer n hacer: MAX = (1<<n)-1;
2 | int matriz[15][15], memo[15][(1<<15)+1], n;
3 |
4 | int tsp(int pos, int mask){
5 |     if(mask == MAX) return matriz[pos][0];
6 |     if(memo[pos][mask] != -1)
7 |         return memo[pos][mask];
8 |
9 |     int res = 1000000000;
10 |    for(int i = 0; i < n; i++){
11 |        if(!(mask & (1<<i))){
12 |            res = min(res, matriz[pos][i]
13 |                    + tsp(i, mask | (1<<i)));
14 |        }
15 |    return memo[pos][mask] = res;
16 | }

```

## 4. Otros

### 4.1. Búsqueda binaria

$O(\log n)$

```

1 | int f(int a, int b){
2 |     return ar[a] > b;
3 | }
4 |
5 | int busqueda_binaria(int men, int may, int v){
6 |     int epsilon = 1, med = 0;
7 |
8 |     while(may-men > epsilon){
9 |         med = (may+men)/2;
10 |        if(f(med,v))
11 |            may = med;
12 |        else
13 |            men = med;
14 |    }

```

```

15 |     return men;
16 | }

```

## 4.2. Raiz babilonica

Encuentra la raiz cuadrada de un numero

```

1 | double raiz(double x) {
2 |     double b = x, h = 0, apro = 1;
3 |     while (apro > 1e-8) {
4 |         b = (h + b) / 2;
5 |         h = x / b;
6 |         apro = abs(h - b);
7 |     }
8 |     return b;
9 | }

```

## 4.3. Codigo gray

```

1 | int gray(int n) {
2 |     return n ^ (n >> 1);
3 | }
4 |
5 | int num(int gray) { //invertir
6 |     int n = 0;
7 |     for (; gray; gray >>= 1)
8 |         n ^= gray;
9 |     return n;
10 | }

```

## 4.4. Lowest Common Ancestor

Ancestro común mas bajo en un arbol, para u y v encontrar el nodo mas bajo que este por encima de ambos.

Solucion con Range Minimum Query (sparse table).

```

1 | #define MAX 100
2 | int l[2*MAX], e[2*MAX], h[MAX], idx;
3 | sparseTable table;
4 |
5 | void dfs(int nodo, int deep, vvi &grafo){
6 |     h[nodo] = idx;

```

```

7 |     e[idx] = nodo;
8 |     l[idx++] = deep;
9 |
10 |    for(int i = 0; i < grafo[nodo].size(); i++){
11 |        if(h[grafo[nodo][i]] != -1) continue;
12 |        dfs(grafo[nodo][i], deep+1, grafo);
13 |        e[idx] = nodo;
14 |        l[idx++] = deep;
15 |    }
16 | }
17 |
18 | void BuildRMQ(vvi &grafo){ //llamar antes de LCA
19 |     idx = 0;
20 |     memset(h, -1, sizeof(h));
21 |     dfs(0, 0, grafo);
22 |     table = sparseTable(grafo.size(), l);
23 | }
24 |
25 | int LCA(int u, int v){ //h[u] < h[v]
26 |     return e[table.query(h[u], h[v])];
27 | }

```

Solucion con construcción  $O(n \log n)$  y consultas  $O(\log n)$

```

1 | int p[100], d[100], peso[100]; //padre, deep, peso
2 | int spt[100][7]; //spt[i][j] = (2^j)-th ancestro de i
3 | vvi grafo; //spt[MAX][(int) log_2(MAX) + 1]
4 |
5 | void dfs(int nodo, int deep, int ant){
6 |     d[nodo] = deep;
7 |     p[nodo] = ant;
8 |     for(int i = 0; i < grafo[nodo].size(); i++){
9 |         if(d[grafo[nodo][i]] != -1) continue;
10 |        dfs(grafo[nodo][i], deep+1, nodo);
11 |    }
12 | }
13 |
14 | void proceso(int n){ //Llamar antes de LCA
15 |     memset(d, -1, sizeof(d));
16 |     dfs(0, 0, -1);
17 |     memset(spt, -1, sizeof(spt));
18 |
19 |     for(int i = 0; i < n; i++){

```

```

20     spt[i][0] = p[i];
21     //mayor[i][0] = peso[i];
22 }
23
24     for(int j = 1; 1 << j < n; j++)
25     for(int i = 0; i < n; i++)
26         if(spt[i][j-1] != -1){
27             spt[i][j] = spt[spt[i][j-1]][j-1];
28             //mayor[i][j] = max(mayor[i][j-1], mayor[spt[i][j-1]][j-1]);
29         }
30 }
31
32 int LCA(int u, int v){
33     if(d[u] < d[v]) swap(u, v); //v debe estar arriba de u
34
35     for(int i = 16; i >= 0; i--) //subimos a u
36         if(d[u] - (1<<i) >= d[v])
37             u = spt[u][i];
38
39     if(u == v) return u;
40
41     for(int i = 16; i >= 0; i--)
42         if(spt[u][i] != spt[v][i]){
43             u = spt[u][i];
44             v = spt[v][i];
45         }
46
47     return p[v];

```

## 5. Matematicas

### 5.1. MCD y MCM

Maximo comun divisor(MCD) y minimo comun multiplo(MCM)

```

1 int mcd(int a, int b){ //algoritmo de euclides
2     return a? mcd(b %a, a): b;
3 }
4
5 int mcm(int a, int b) {
6     return a*b/mcd(a,b);

```

```

7 }

```

### 5.2. Exponenciacion binaria

$O(\log n)$

```

1 typedef long long int lli;
2
3 lli exp_bin (lli a, lli n) {
4     lli res = 1;
5     while (n) {
6         if (n & 1)
7             res *= a;
8         a *= a;
9         n >>= 1;
10    }
11    return res;
12 }

```

### 5.3. Algoritmo extendido de euclides

Encuentra dos numeros x e y tal que:  $MCD(a, b) = ax + by$

```

1 int gcd_ex (int a, int b, int &x, int &y) {
2     if (a == 0) {
3         x = 0; y = 1;
4         return b;
5     }
6     int x1, y1;
7     int d = gcd_ex (b%a, a, x1, y1);
8     x = y1 - (b / a) * x1;
9     y = x1;
10    return d;
11 }
12
13 int main(){
14     int n, m, x, y, res;
15
16     while(cin >> n >> m){
17         res = gcd_ex(n, m, x, y);
18         cout << "gcd=" << res << ",x=" << x << ",y=" <<
            y << endl;

```

```

19     }
20 }

```

## 5.4. Inverso multiplicativo modular

Encuentra un  $x$  tal que  $(a * x)$  es congruente a 1 con modulo  $p$ , entonces:  
 $(a * x) \bmod p = 1 \bmod p$   
 necesita del algoritmo extendido de euclides  
 $O(\log m)$

```

1 void inverso(int a, int m){
2     int x, y;
3
4     int g = gcd_ex (a, m, x, y);
5     if (g != 1)
6         cout << "no_solution";
7     else {
8         x = (x % m + m) % m;
9         cout << x << endl;
10    }
11 }

```

## 5.5. Phi de euler

Devuelve la cantidad de coprimos de un numero  $n$   
 $O(\sqrt{n})$

```

1 int phi (int n) {
2     int result = n;
3     for (int i=2; i*i<=n; ++i)
4         if (n % i == 0) {
5             while (n % i == 0)
6                 n /= i;
7             result -= result / i;
8         }
9     if (n > 1)
10        result -= result / n;
11    return result;
12 }

```

## 5.6. Multiplicacion modular

Encuentra  $(a*b) \bmod c$ , la operacion puede generar overflow si se realiza directamente, el metodo mulmod evita el overflow usando un ciclo, pero se puede usar el tipo de dato int128 de c++11 para poder calcular de manera directa, pero el int128 no se puede leer o imprimir directamente.

```

1 typedef long long int lli; //metodo normal
2 lli mulmod (lli a, lli b, lli c) {
3     lli x = 0, y = a%c;
4     while (b > 0){
5         if (b % 2 == 1) x = (x+y) % c;
6         y = (y*2) % c;
7         b /= 2;
8     }
9     return x % c;
10 }
11
12 typedef __int128 bi; //metodo con __int128
13 lli mulmod_2(bi a, bi b, bi c){
14     return (lli) ((a*b) % c);
15 }
16
17 int main(){
18     lli a, b, c;
19     cin >> a >> b >> c;
20     cout << mulmod_2((bi) a, (bi) b, (bi) c) << endl;
21     return 0;
22 }

```

## 5.7. Exponenciacion modular

Encuentra  $(a^b) \bmod c$ , se necesita implementar previamente multiplicacion modular.

```

1 ll expmod (ll b, ll e, ll m){ //O(log b)
2     if(!e) return 1;
3     ll q = expmod(b,e/2,m); q = mulmod(q,q,m);
4     return e%2? mulmod(b,q,m) : q;
5 }

```

## 5.8. Test de Rabin Miller

Devuelve si un numero es primo, requiere de implementar previamente GCD(maximo común divisor), multiplicacion modular y exponenciacion modular.

```
1 bool es_primo_prob (lli n, int a) {
2     if (n == a) return true;
3     lli s = 0, d = n-1;
4     while (d % 2 == 0) s++, d/=2;
5
6     lli x = expmod(a,d,n);
7     if ((x == 1) || (x+1 == n)) return true;
8
9     forn (i, s-1){
10         x = mulmod(x, x, n);
11         if (x == 1) return false;
12         if (x+1 == n) return true;
13     }
14     return false;
15 }
16
17 bool rabin (lli n){ //devuelve true si n es primo
18     if (n == 1) return false;
19     const int ar[] = {2,3,5,7,11,13,17,19,23};
20     forn (j,9)
21         if (!es_primo_prob(n,ar[j]))
22             return false;
23     return true;
24 }
```

## 5.9. Rho de pollard

Factorizacion rapida, usar para  $n > 10^{12}$ , requiere de implementar previamente el GCD (maximo común divisor), multiplicacion modular, exponenciacion modular y el test de Rabin Miller.

$O(\sqrt[4]{n})$

```
1 lli rho(lli n){
2     if( (n & 1) == 0 ) return 2;
3     lli x = 2 , y = 2 , d = 1;
4     lli c = rand() % n + 1;
5     while( d == 1 ){
```

```
6         x = (mulmod( x , x , n ) + c)%n;
7         y = (mulmod( y , y , n ) + c)%n;
8         y = (mulmod( y , y , n ) + c)%n;
9         if( x - y >= 0 ) d = gcd( x - y , n );
10        else d = gcd( y - x , n );
11    }
12    return d==n? rho(n):d;
13 }
14
15 map<lli, lli> prim;
16
17 void factRho (lli n){ //O (lg n)^3. un solo numero
18     if (n == 1) return;
19     if (rabin(n)){
20         prim[n]++;
21         return;
22     }
23     lli factor = rho(n);
24     factRho(factor);
25     factRho(n/factor);
26 }
27
28 int main(){
29     lli n;
30     while(scanf("%lld", &n), n > 0){
31         prim.clear();
32         factRho(n);
33
34         for(map<lli, lli>::iterator it = prim.begin(); it !=
35             prim.end(); it++){
36             cout << "el " << (it)->first << " aparece " << (it)
37                 ->second << " veces. \n";
38         }
39     }
40     return 0;
41 }
```

## 5.10. factorizacion con criba

Factorizacion usando la criba, mas corto de escribir que rho de pollard, usar para  $n \leq 10^{12}$ , guarda los factores en un mapa similar a rho de pollard.

```

1 int m= 1000010, m2= 1000000, primo[1000020];
2 vector<lli> p;
3 map<lli, int> mapa;
4
5 void criba(){
6     memset(primo, 0, sizeof(primo));
7
8     for(int i = 2; i < m; i++){
9         if(primo[i]) continue;
10        p.push_back(i);
11        primo[i] = i;
12        if(i > 1000) continue;
13
14        for(int j = i*i; j < m; j += i)
15            primo[j] = i;
16    }
17 }
18
19 void factCriba(lli n){
20     int l;
21     bool s;
22
23     while(n != 1){
24         if(n > m2){//n mayor a logintud del array
25             l = sqrt(n) + 1;
26             s = false;
27             for(int i = 0; p[i] <= l; i++){
28                 if(n % p[i] == 0){
29                     mapa[p[i]]++;
30                     s = true;
31                     n /= p[i];
32                     break;
33                 }
34             }
35             if(!s){
36                 mapa[n]++;
37                 break;
38             }
39         }else{
40             mapa[primo[n]]++;
41         }

```

```

42         n /= primo[n];
43     }
44 }
45 }

```

## 5.11. BigInteger c++

```

1 typedef unsigned long long int ulli;
2 typedef long long int tdato; //no debe ser unsigned para la
3                               resta!!!
4 tdato base = 1000000000;
5
6 struct bigint{
7     vector<tdato> num;
8     bool signo;
9
10    bigint(){ signo = true; }
11    bigint(int n){
12        num.clear();
13        signo = n >= 0; n = abs(n);
14        while(n){
15            num.push_back((n >= base)? n % base: n);
16            n /= base;
17        }
18    }
19    bigint(string n){
20        num.clear();
21        signo = n[0] != '-';
22        if(n[0] == '-') n = n.substr(1);
23        for(int i = n.size(); i > 0; i -= 9){
24            if(i < 9) num.push_back(atoi(n.substr(0, i).c_str()));
25            else num.push_back(atoi(n.substr(i-9, 9).c_str()));
26        }
27        quitar_zeros_izq();
28    }
29
30    void quitar_zeros_izq(){
31        while(num.size() && !num.back()) num.pop_back();
32    }
33
34    void imprimir(){

```



```

33     if(!signo && num.size()) printf("-");
34     printf("%d", ((num.size())? num.back(): 0));
35     for(int i = num.size() - 2; i >= 0; i--)
36         printf("%09d", num[i]);
37     printf("\n");
38 }
39
40 bigint suma(bigint b){
41     ulli carry = 0, aux;
42     int l = max(b.num.size(), num.size());
43     bigint c;
44
45     for(int i = 0; i < l || carry; i++){
46         aux = carry;
47         if(i < b.num.size()) aux += b.num[i];
48         if(i < num.size()) aux += num[i];
49
50         if(aux >= base){
51             c.num.push_back(aux % base);
52             carry = aux / base;
53         }else{
54             c.num.push_back(aux); carry = 0;
55         }
56     }
57     return c;
58 }
59
60 bigint resta(bigint b){//asumimos que b es menor
61     tdata carry = 0;//no debe ser unsigned
62     bigint c;
63
64     for(int i = 0; i < num.size(); i++){
65         c.num.push_back(num[i]);
66         c.num[i] -= ((i < b.num.size())? b.num[i]: 0) +
67             carry;
68         if(c.num[i] < 0){
69             c.num[i] += base; carry = 1;
70         }else carry = 0;
71     }
72     c.quitar_zeros_izq();
73     return c;

```

```

73 }
74
75 bigint multiplicar(bigint b){
76     ulli aux = 0, carry;
77     bigint c;
78     c.num.assign(num.size() + b.num.size(), 0);
79
80     for(int i = 0; i < num.size(); i++){
81         carry = 0;
82         for(int j = 0; j < b.num.size() || carry; j++){
83             aux = c.num[i + j] + carry + (num[i] * ((j < b.
84                 num.size())? b.num[j] : 0));
85             carry = aux / base;
86             c.num[i + j] = aux % base;
87         }
88     }
89     c.quitar_zeros_izq();
90     return c;
91 }
92
93 void dividirDos(){
94     tdata carry = 0;
95     ulli aux;
96     for (int i = num.size() - 1; i >= 0; --i) {
97         aux = num[i] + carry * base;
98         num[i] = aux / 2; carry = aux % 2;
99     }
100     quitar_zeros_izq();
101 }
102
103 bigint dividir(bigint b){//busqueda binaria
104     if(comparar(b) < 0) return bigint(0);
105     bigint may, men(0), med, m(1);
106     may = suma(m); may.signo = true;
107     int cmp;
108
109     while(true){
110         med = may.suma(men); med.signo = true;
111         med.dividirDos();
112         m = med.multiplicar(b); m.signo = true;

```

```

113         cmp = comparar(m);
114         if(cmp == 0) break;
115         else if(cmp < 0) may.num.assign(med.num.begin(), med
            .num.end());
116         else{
117             if(resta(m).comparar(b) < 0) break;
118             else men.num.assign(med.num.begin(), med.num.end
                ());
119         }
120     }
121     return med;
122 }
123
124 int comparar(bigint b){//este es: 1 mayor, 0 igual, -1
    menor
125     if(num.size() > b.num.size()) return 1;
126     else if(num.size() < b.num.size()) return -1;
127     else{
128         for(int i = num.size() - 1; i >= 0; i--){
129             if(num[i] > b.num[i]) return 1;
130             else if(num[i] < b.num[i]) return -1;
131         }
132         return 0;
133     }
134 }
135 };
136
137 bool operator>(bigint &a, bigint &b){
138     if(a.signo == b.signo){
139         if(a.signo) return a.comparar(b) > 0;
140         else return b.comparar(a) > 0;
141     }else return a.signo;
142 }
143 bool operator<(bigint &a, bigint &b){
144     if(a.signo == b.signo){
145         if(a.signo) return a.comparar(b) < 0;
146         else return b.comparar(a) < 0;
147     }else return !a.signo;
148 }
149 bool operator==(bigint &a, bigint &b){
150     if(a.signo != b.signo) return false;

```

```

151     else return a.comparar(b) == 0;
152 }
153 bool operator!=(bigint &a, bigint &b){
154     return !(a==b);
155 }
156 bigint operator+(bigint &a, bigint&b){
157     bigint c;
158     if(a.signo == b.signo){
159         c = a.suma(b);
160         c.signo = a.signo;
161     }else if(a > b){
162         c = a.resta(b);
163         c.signo = a.signo;
164     }else{
165         c = b.resta(a);
166         c.signo = b.signo;
167     }
168     return c;
169 }
170 bigint operator-(bigint &a, bigint&b){
171     bigint c;
172     if(a.comparar(b) > 0){
173         if(a.signo == b.signo) c = a.resta(b);
174         else c = a.suma(b);
175         c.signo = a.signo;
176     }else{
177         if(a.signo == b.signo) c = b.resta(a);
178         else c = a.suma(b);
179         c.signo = !b.signo;
180     }
181     return c;
182 }
183 bigint operator*(bigint &a, bigint &b){
184     bigint c = a.multiplicar(b);
185     c.signo = a.signo == b.signo;
186     return c;
187 }
188 bigint operator/(bigint &a, bigint &b){
189     bool s = (a.signo == b.signo);
190     a.signo = true; b.signo = true;
191     bigint c = a.dividir(b);

```

```

192     c.signo = s;
193     return c;
194 }

```

## 5.12. Fraccion

```

1 struct fraccion {
2     int num, den;
3
4     fraccion(int x, int y) {
5         num = x; den = y;
6         if (den < 0){ num *= -1; den *= -1; }
7         if (num == 0) den = 1;
8         else {
9             int dividir = MCD(num, den);
10            num /= dividir;
11            den /= dividir;
12        }
13    }
14
15    fraccion operator+(fraccion b) { //suma
16        return fraccion(num*b.den + b.num*den,
17                          den*b.den);
18    }
19    fraccion operator-(fraccion b) { //resta
20        return fraccion(num*b.den - b.num*den,
21                          den*b.den);
22    }
23    fraccion operator*(fraccion b) { //multiplicar
24        return fraccion(num*b.num, den*b.den);
25    }
26    fraccion inversa() {
27        return fraccion(den, num);
28    }
29    fraccion operator/(fraccion b) { //dividir
30        return fraccion(num*b.den, b.num*den);
31    }
32    string toString() {
33        stringstream ss;
34        ss << num;
35        if (den == 1) return ss.str();
36        ss << "/" << den;

```

```

37         return ss.str();
38     }
39 };

```

## 6. Cadenas

### 6.1. Algoritmo de bordes

Encuentra la longitud del mayor borde de un string n.

```

1 int bordes[1000];
2
3 void algoritmoBordes(string subcad){
4     int i = 1, j = -1;
5     bordes[0] = -1;
6
7     while(i < subcad.size()) {
8         while(j >= 0 && subcad[i] != subcad[j])
9             j = bordes[j];
10        i++; j++;
11        bordes[i] = j;
12    }
13 }

```

### 6.2. KMP

Encuentra si una cadena n es subcadena de otra cadena m, requiere de implementar y ejecutar previamente el algoritmo de bordes  $O(n+m)$

```

1 void kmp(string cad, string subcad){
2     int i = 0, j = 0;
3     while(i < cad.size()){
4         while(j >= 0 && cad[i] != subcad[j]) j = bordes[j];
5         i++; j++;
6         if(j == subcad.size()){
7             printf("s_esta_en_el_indice_%d_de_la_cadena:_%s\n",
8                   subcad.c_str(), i - j, cad.c_str());
9             j = bordes[j];
10        }
11    }
12 }

```

## 7. Geometria

### 7.1. Punto

```
1 struct punto{
2     double x, y;
3
4     punto(){ x = y = 0; }
5     punto(double _x, double _y){
6         x = _x; y = _y;
7     }
8
9     bool operator < (punto p) const{//para poder usar sort
10         if(fabs(x - p.x) > eps) return x < p.x;
11         return y < p.y;
12     }
13     bool operator == (punto p) const{
14         return fabs(x - p.x) < eps && fabs(y - p.y) < eps;
15     }
16 };
17
18 vec toVec(punto a, punto b){return vec(b.x-a.x, b.y-a.y);}
19 double DEG_TO_RAD(double n){ return n*3.1416/180.0; }
20
21 punto rotar(punto p, double grados){
22     double rad = DEG_TO_RAD(grados) + cos(5);
23     return punto(p.x*cos(rad) - p.y*sin(rad),
24                 p.x*sin(rad) + p.y*cos(rad));
25 }
26 punto trasladar(punto p, vec v){
27     return punto(p.x+v.x, p.y+v.y);
28 }
29 double dist(punto p1, punto p2){
30     return hypot(p1.x - p2.x, p1.y - p2.y);
31 }
32 double angulo(punto a, punto o, punto b){//en radianes
33     vec oa = toVec(o, a), ob = toVec(o, b);
34     return acos(dot(oa, ob)/sqrt(norm_sq(oa)*norm_sq(ob)));
35 }
```

### 7.2. Linea

Linea de la forma  $ax + by + c = 0$ .

```
1 struct linea{
2     double a, b, c;
3     punto p1, p2;
4
5     linea(double _a, double _b, double _c){
6         a = _a; b = _b; c = _c;
7     }
8     linea(punto _p1, punto _p2){
9         p1 = punto(_p1.x, _p1.y);
10        p2 = punto(_p2.x, _p2.y);
11        if(fabs(p1.x - p2.x) < eps){
12            a = 1.0; b = 0.0; c = -p1.x;
13        }else{
14            a = -((p1.y-p2.y) / (p1.x-p2.x));
15            b = 1.0;
16            c = -((a*p1.x) / (p1.y));
17        }
18    }
19 };
20
21 bool paralelas(linea l1, linea l2){
22     return fabs(l1.a-l2.a)<eps && fabs(l1.b-l2.b)<eps;
23 }
24 bool iguales(linea l1, linea l2){
25     return paralelas(l1, l2) && fabs(l1.c-l2.c)<eps;
26 }
27 bool interseccion(linea l1, linea l2, punto &p){
28     if(paralelas(l1, l2)) return false;
29     p.x = (l2.b*l1.c-l1.b*l2.c) / (l2.a*l1.b-l1.a*l2.b);
30     if(fabs(l1.b)>eps) p.y = -(l1.a*p.x + l1.c);
31     else p.y = -(l2.a*p.x + l2.c);
32     return true;
33 }
34 //distancia minima entre p y l
35 double distLinea(punto p, linea l, punto &c){
36     punto a = l.p1, b = l.p2;
37     vec ap = toVec(a, p), ab = toVec(a, b);
38     double u = dot(ap, ab) / norm_sq(ab);
39     c = trasladar(a, escalar(ab, u));//punto mas cercano
```

```

40     return dist(p, c);
41 }
42 double distSegmento(punto p, linea l, punto &c){
43     punto a = l.p1, b = l.p2;
44     vec ap = toVec(a, p), ab = toVec(a, b);
45     double u = dot(ap, ab) / norm_sq(ab);
46     if(u < 0.0){
47         c = punto(a.x, a.y); return dist(p, a);
48     }
49     if(u > 1.0){
50         c = punto(b.x, b.y); return dist(p, b);
51     }
52     return distLinea(p, l, c);
53 }

```

```

25 }
26 bool colineal(linea l, punto r){
27     return fabs(cross(toVec(l.p1,l.p2),
28                     toVec(l.p1,r))) < eps;
29 } //la linea l contiene el punto r

```

### 7.3. Vector

```

1 struct vec{
2     double x, y;
3     vec(double _x, double _y){
4         x = _x; y = _y;
5     }
6 };
7
8 vec toVec(punto a, punto b){
9     return vec(b.x-a.x, b.y-a.y);
10 }
11 vec escalar(vec v, double s){
12     return vec(v.x*s, v.y*s);
13 }
14 double dot(vec a, vec b){
15     return a.x*b.x + a.y*b.y;
16 }
17 double norm_sq(vec v){
18     return v.x*v.x + v.y*v.y;
19 }
20 double cross(vec a, vec b){
21     return a.x*b.y - a.y*b.x;
22 }
23 bool ccw(punto p, punto q, punto r){
24     return cross(toVec(p,q), toVec(p,r)) > 0;

```

## 8. Tips and formulas(UFPS, 2017)

### 8.1. ASCII Table

Caracteres ASCII con sus respectivos valores numéricos.

No.	ASCII	No.	ASCII
0	NUL	16	DLE
1	SOH	17	DC1
2	STX	18	DC2
3	ETX	19	DC3
4	EOT	20	DC4
5	ENQ	21	NAK
6	ACK	22	SYN
7	BEL	23	ETB
8	BS	24	CAN
9	TAB	25	EM
10	LF	26	SUB
11	VT	27	ESC
12	FF	28	FS
13	CR	29	GS
14	SO	30	RS
15	SI	31	US

No.	ASCII	No.	ASCII
32	(space)	48	0
33	!	49	1
34	"	50	2
35	#	51	3
36	\$	52	4
37	%	53	5
38	&	54	6
39	'	55	7
40	(	56	8
41	)	57	9
42	*	58	:
43	+	59	;
44	,	60	i
45	-	61	=
46	.	62	¿
47	/	63	?

No.	ASCII	No.	ASCII
64	@	80	P
65	A	81	Q
66	B	82	R
67	C	83	S
68	D	84	T
69	E	85	U
70	F	86	V
71	G	87	W
72	H	88	X
73	I	89	Y
74	J	90	Z
75	K	91	[
76	L	92	\
77	M	93	]
78	N	94	^
79	O	95	-

No.	ASCII	No.	ASCII
96	`	112	p
97	a	113	q
98	b	114	r
99	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{
108	l	124	
109	m	125	}
110	n	126	~
111	o	127	

## 8.2. Formulas

PERMUTACIÓN Y COMBINACIÓN	
Combinación (Coeficiente Binomial)	Número de subconjuntos de k elementos escogidos de un conjunto con n elementos. $\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$
Combinación con repetición	Número de grupos formados por n elementos, partiendo de m tipos de elementos. $CR_m^n = \binom{m+n-1}{n} = \frac{(m+n-1)!}{n!(m-1)!}$
Permutación	Número de formas de agrupar n elementos, donde importa el orden y sin repetir elementos $P_n = n!$
Permutación múltiple	Elegir r elementos de n posibles con repetición $n^r$
Permutación con repetición	Se tienen n elementos donde el primer elemento se repite a veces , el segundo b veces , el tercero c veces, ... $PR_n^{a,b,c,\dots} = \frac{P_n}{a!b!c!\dots}$
Permutaciones sin repetición	Número de formas de agrupar r elementos de n disponibles, sin repetir elementos $\frac{n!}{(n-r)!}$
DISTANCIAS	

Continúa en la siguiente columna

Distancia Euclidea	$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
Distancia Manhattan	$d_M(P_1, P_2) =  x_2 - x_1  +  y_2 - y_1 $
CIRCUNFERENCIA Y CÍRCULO	
Considerando r como el radio, α como el ángulo del arco o sector, y (R, r) como radio mayor y menor respectivamente.	
Área	$A = \pi * r^2$
Longitud	$L = 2 * \pi * r$
Longitud de un arco	$L = \frac{2 * \pi * r * \alpha}{360}$
Área sector circular	$A = \frac{\pi * r^2 * \alpha}{360}$
Área corona circular	$A = \pi(R^2 - r^2)$
TRIÁNGULO	
Considerando b como la longitud de la base, h como la altura, letras minúsculas como la longitud de los lados, letras mayúsculas como los ángulos, y r como el radio de circunferencias asociadas.	
Área conociendo base y altura	$A = \frac{1}{2}b * h$

Continúa en la siguiente columna

Área conociendo 2 lados y el ángulo que forman	$A = \frac{1}{2}b * a * \sin(C)$
Área conociendo los 3 lados	$A = \sqrt{p(p-a)(p-b)(p-c)}$ con $p = \frac{a+b+c}{2}$
Área de un triángulo circunscrito a una circunferencia	$A = \frac{abc}{4r}$
Área de un triángulo inscrito a una circunferencia	$A = r(\frac{a+b+c}{2})$
Área de un triángulo equilátero	$A = \frac{\sqrt{3}}{4}a^2$
RAZONES TRIGONOMÉTRICAS	
Considerando un triángulo rectángulo de lados $a, b$ y $c$ , con vértices $A, B$ y $C$ (cada vértice opuesto al lado cuya letra minúscula coincide con el) y un ángulo $\alpha$ con centro en el vértice $A$ . $a$ y $b$ son catetos, $c$ es la hipotenusa:	
$\sin(\alpha) = \frac{\text{cateto opuesto}}{\text{hipotenusa}} = \frac{a}{c}$	

Continúa en la siguiente columna

$\cos(\alpha) = \frac{\text{cateto adyacente}}{\text{hipotenusa}} = \frac{b}{c}$	
$\tan(\alpha) = \frac{\text{cateto opuesto}}{\text{cateto adyacente}} = \frac{a}{b}$	
$\sec(\alpha) = \frac{1}{\cos(\alpha)} = \frac{c}{b}$	
$\csc(\alpha) = \frac{1}{\sin(\alpha)} = \frac{c}{a}$	
$\cot(\alpha) = \frac{1}{\tan(\alpha)} = \frac{b}{a}$	
PROPIEDADES DEL MÓDULO (RESIDUO)	
Propiedad neutro	$(a \% b) \% b = a \% b$
Propiedad asociativa en multiplicación	$(ab) \% c = ((a \% c)(b \% c)) \% c$
Propiedad asociativa en suma	$(a + b) \% c = ((a \% c) + (b \% c)) \% c$
CONSTANTES	
Pi	$\pi = \arccos(-1) \approx 3,14159$

Continúa en la siguiente columna



e	$e \approx 2,71828$
Número áureo	$\phi = \frac{1 + \sqrt{5}}{2} \approx 1,61803$

### 8.3. Sequences

Listado de secuencias mas comunes y como hallarlas.

Estrellas octangulares	0, 1, 14, 51, 124, 245, 426, 679, 1016, 1449, 1990, 2651, ...
	$f(n) = n * (2 * n^2 - 1).$
Euler totient	1, 1, 2, 2, 4, 2, 6, 4, 6, 4, 10, 4, 12, 6,...
	$f(n) =$ Cantidad de números naturales $\leq n$ coprimos con n.
Números de Bell	1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, ...
	Se inicia una matriz triangular con $f[0][0] = f[1][0] = 1$ . La suma de estos dos se guarda en $f[1][1]$ y se traslada a $f[2][0]$ . Ahora se suman $f[1][0]$ con $f[2][0]$ y se guarda en $f[2][1]$ . Luego se suman $f[1][1]$ con $f[2][1]$ y se guarda en $f[2][2]$ trasladandose a $f[3][0]$ y así sucesivamente. Los valores de la primera columna contienen la respuesta.
Números de Catalán	1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, ...
	$f(n) = \frac{(2n)!}{(n+1)!n!}$
Números de Fermat	3, 5, 17, 257, 65537, 4294967297, 18446744073709551617, ...
	$f(n) = 2^{(2^n)} + 1$
Números de Fibonacci	0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...
	$f(0) = 0; f(1) = 1; f(n) = f(n-1) + f(n-2)$ para $n > 1$

Continúa en la siguiente columna

Números de Lucas	2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, ...
	$f(0) = 2; f(1) = 1; f(n) = f(n-1) + f(n-2)$ para $n > 1$
Números de Pell	0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, ...
	$f(0) = 0; f(1) = 1; f(n) = 2f(n-1) + f(n-2)$ para $n > 1$
Números de Tribonacci	0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, ...
	$f(0) = f(1) = 0; f(2) = 1; f(n) = f(n-1) + f(n-2) + f(n-3)$ para $n > 2$
Números factoriales	1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...
	$f(0) = 1; f(n) = \prod_{k=1}^n k$ para $n > 0$ .
Números piramidales cuadrados	0, 1, 5, 14, 30, 55, 91, 140, 204, 285, 385, 506, 650, ...
	$f(n) = \frac{n * (n+1) * (2 * n + 1)}{6}$
Números primos de Mersenne	3, 7, 31, 127, 8191, 131071, 524287, 2147483647, ...
	$f(n) = 2^{p(n)} - 1$ donde $p$ representa valores primos iniciando en $p(0) = 2$ .
Números tetraedrales	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \frac{n * (n+1) * (n+2)}{6}$
Números triangulares	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \frac{n(n+1)}{2}$

Continúa en la siguiente columna

OEIS A000127	1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, 562, ...
	$f(n) = \frac{(n^4 - 6n^3 + 23n^2 - 18n + 24)}{24}$ .
Secuencia de Narayana	1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88, 129, ...
	$f(0) = f(1) = f(2) = 1; f(n) = f(n-1) + f(n-3)$ para todo $n > 2$ .
Secuencia de Silvestre	2, 3, 7, 43, 1807, 3263443, 10650056950807, ...
	$f(0) = 2; f(n+1) = f(n)^2 - f(n) + 1$
Secuencia de vendedor perezoso	1, 2, 4, 7, 11, 16, 22, 29, 37, 46, 56, 67, 79, 92, 106, ...
	Equivale al triangular(n) + 1. Máxima número de piezas que se pueden formar al hacer n cortes a un disco. $f(n) = \frac{n(n+1)}{2} + 1$
Suma de los divisores de un número	1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, ...
	Para todo $n > 1$ cuya descomposición en factores primos es $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ se tiene que: $f(n) = \frac{p_1^{a_1+1} - 1}{p_1 - 1} * \frac{p_2^{a_2+1} - 1}{p_2 - 1} * \dots * \frac{p_k^{a_k+1} - 1}{p_k - 1}$
Schroeder numbers (aporte propio)	1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859, ...
	El número de formas de insertar paréntesis en una secuencia y el número de formas de partir un polígono convexo en polígonos más pequeños mediante la inserción de diagonales. $f(1)=f(2)=1$ ; $f(n) = \frac{3(2n-3) * f(n-1) - (n-3) * f(n-2)}{n}$

Continúa en la siguiente columna

## 8.4. Time Complexities

Aproximación del mayor número n de datos que pueden procesarse para cada una de las complejidades algorítmicas. Tomar esta tabla solo como referencia.

Complexity	n
$O(n!)$	11
$O(n^5)$	50
$O(2^n * n^2)$	18
$O(2^n * n)$	22
$O(n^4)$	100
$O(n^3)$	500
$O(n^2 \log_2 n)$	1.000
$O(n^2)$	10.000
$O(n \log_2 n)$	$10^6$
$O(n)$	$10^8$
$O(\sqrt{n})$	$10^{16}$
$O(\log_2 n)$	-
$O(1)$	-

## 9. Extras

### 9.1. Template

Plantilla de typedef, define, etc.

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  typedef pair<int, int> ii;
6  typedef pair<int, ii> iii;
7
8  typedef vector<int> vi;
9  typedef vector<vi> vvi;
10 typedef vector<ii> vii;
11 typedef vector<vii> vvii;
12

```

```

13 #define mpiii(a, b, c) iii(a, ii(b, c))
14 #define inf 1000000000//10^9
15 #define INFmemset 5436//inf para memeset
16
17 double eps = 1e-5;//ajustar segun se necesite
18
19 int main(){//fast I/O con iostream
20     cin.tie(NULL);
21     ios_base::sync_with_stdio(false);
22     cout << "hola_mundo" << '\n';
23     return 0;
24 }

```

## 9.2. Formulas extra

**formula de triangulos degenerados:**

$$\frac{(a+b-c)*(a+c-b)*(b+c-a)}{a*b*c}$$

Si el resultado es mayor a 0.5 es un triángulo de calidad buena.

Es posible formar un triangulo si  $a + b > c$  con  $c > b > a$

**Ecuacion de la recta que pasa por dos puntos:**

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$$

**Distancia de un punto a una recta:**

Teniendo una recta con formula de la forma:  $ax + by + c$  la distancia minima a un punto p de la forma  $(px, py)$  la distancia minima esta dada por la formula:

$$d = \frac{a*px + b*py + c}{\sqrt{a^2 + b^2}}$$

**Formula de numeros fibonaccii:**

$$f(n) = \frac{1}{\sqrt{5}} * \left[ \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right]$$

**Determinante de Gauss:**

Encontrar el area de un poligono en el plano cartesiano a partir de sus vertices

$$A = \frac{1}{2} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_1 & y_1 \end{bmatrix}$$

$$S = x_1y_2 + x_2y_3 + \dots + x_ny_1$$

$$D = x_2y_1 + x_3y_2 + \dots + x_1y_n$$

$$A = \frac{1}{2}|S - D|$$

**Coeficientes binomiales:**

Encuentra  $n$  combinado  $k$ , para construir el triangulo de pascal solo poner en  $n$  la fila y en  $k$  la columna.

$$C(n, k) = \begin{cases} 0 & \text{si } k = 0, n = k \\ C(n-1, k-1) + C(n-1, k) & \text{c.c.} \end{cases}$$

**Numeros de catalan:**

Formula recursiva, encontrar numero de arboles binarios de  $n$  nodos, numero de formas de emparejar parentesis.

$$Cat(n) = \begin{cases} 1 & \text{si } n = 0 \\ \frac{2n*(2n-1)*Cat(n-1)}{(n+1)*n} & \text{c.c.} \end{cases}$$

## 9.3. Secuencias

**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	70368744177664	140737488355328	281474976710656	562949953421312
1223	1229	1231	1237	1249	1259	1277	1279	1283	1289	1291	1297	1301	1303	1125899906842624	2251799813685248	4503599627370496	9007199254740992
1307	1319	1321	1327	1361	1367	1373	1381	1399	1409	1423	1427	1429	1433	18014398509481984	36028797018963968	72057594037927936	
1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1499	1511	1523	144115188075855872	288230376151711744	576460752303423488	
1531	1543	1549	1553	1559	1567	1571	1579	1583	1597	1601	1607	1609	1613	1152921504606846976	2305843009213693952	4611686018427387904	
1619	1621	1627	1637	1657	1663	1667	1669	1693	1697	1699	1709	1721	1723	9223372036854775808			
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	2083	2087	2089	2099	2111	2113	2129	2131	2137	2141	2143	2153				
2161	2179	2203	2207	2213	2221	2237	2239	2243	2251	2267	2269	2273	2281				
2287	2293	2297	2309	2311	2333	2339	2341	2347	2351	2357	2371	2377	2381				
2383	2389	2393	2399	2411	2417	2423	2437	2441	2447	2459	2467	2473	2477				
2503	2521	2531	2539	2543	2549	2551	2557	2579	2591	2593	2609	2617	2621				
2633	2647	2657	2659	2663	2671	2677	2683	2687	2689	2693	2699	2707	2711				
2713	2719	2729	2731	2741	2749	2753	2767	2777	2789	2791	2797	2801	2803				
2819	2833	2837	2843	2851	2857	2861	2879	2887	2897	2903	2909	2917	2927				

#### **Fibonacci:**

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181  
6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040  
1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169  
63245986 102334155 165580141 267914296 433494437 701408733 1134903170  
1836311903

#### **Factoriales:**

1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600  
6227020800 87178291200 1307674368000 20922789888000 355687428096000  
6402373705728000 121645100408832000

#### **Potencias de dos:** de 1 hasta 63

1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768  
65536 131072 262144 524288 1048576 2097152 4194304 8388608  
16777216 33554432 67108864 134217728 268435456 536870912 1073741824  
2147483648 4294967296 8589934592 17179869184 34359738368 68719476736  
137438953472 274877906944 549755813888 1099511627776 2199023255552  
4398046511104 8796093022208 17592186044416 35184372088832