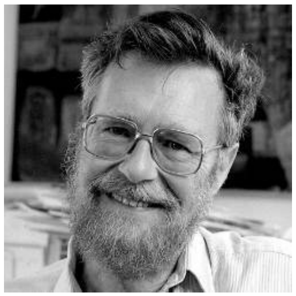
Repositorio en C++

Universidad de la Amazonia, Colombia.

12 de abril de 2021



I2

Índice

1. Estructuras de datos

1.1. Tablas aditivas

Construcción O(n), Consulta O(1)

```
void build(){
    memset(memo, 0, sizeof(memo));
    memo[1][1] = tab[0][0];
    for (int i = 2; i <= fila; i++)
        memo[i][1] = memo[i-1][1] + tab[i - 1][0];
    for (int j = 2; j <= col; j++)
        memo[1][j] = memo[1][j-1] + tab[0][j - 1];

    for (int i = 2; i <= fila; i++)
    for (int j = 2; j <= col; j++)
            memo[i][j] = memo[i][j - 1] + memo[i - 1][j] +
                    tab[i - 1][j - 1] - memo[i - 1][j - 1];
}
//indexando desde 1
int query(int f1, int c1, int f2, int c2){
    return memo[f2][c2] - memo[f1-1][c2] -
                memo[f2][c1-1] + memo[f1-1][c1-1];
}
```

1.2. Disjoint set union find

Asocia elementos en conjuntos de arboles.
Construcción O(n)

```
struct union_find{
    int padre[MAX], rango[MAX];//Rango opcional

    void iniciar(int n){
        for (int i = 0; i < n; i++){
            padre[i] = i;  rango[i] = 0;
        }
    }

    int raiz(int x){
        if(x == padre[x]) return x;
        else return padre[x] = raiz(padre[x]);
    }
```

```

    void unir(int x, int y){
        x = raiz(x);  y = raiz(y);
        if(x == y) return;

        if(rango[x] > rango[y]){
            padre[y] = x;
            return;
        }
        padre[x] = y;
        if(rango[y] == rango[x]) rango[y]++;
    }

    bool MismoGrupo(int x, int y){return raiz(x) == raiz(y);}

    //Usar este para compresion de caminos
    int raiz_compresion(int x){
        if(x == padre[x]) return x;
        else return padre[x] = raiz(padre[x]);
    }
    //Usar este para compresion de caminos
    void unir_compresion(int x, int y){
        padre[raiz(x)] = raiz(y);
    }
};
```

1.3. Segment tree

Ejemplo de RMQ (Range Minium Query)
Construcción O(n), Consulta O(log n), Update O(log n)

```
const int MAX = 4 * 1000;//poner 4 * longitud maxima

struct segment_tree{
    int st[MAX];
    vi A;
    int n, tamst;

    int mov_izq(int index){ return index << 1; }
    int mov_der(int index){ return (index << 1) + 1; }

    void construir(int pos, int izq, int der){
        if(izq == der){
            st[pos] = A[der];
            return;
        }

        construir(mov_izq(pos), izq, (izq + der) >> 1);
        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
        st[pos] = min(st[aux1], st[aux2]);
    }

    void iniciar(vi arr){//metodo a invocar
        A = arr;
        n = A.size();
        tamst = n << 2;
        construir(1, 0, n - 1);
    }

    int query(int pos, int izq, int der, int i, int j){
        if(i > der || j < izq) return -1;
        if(i <= izq && j >= der) return st[pos];

        int aux1 = query(mov_izq(pos), izq, (izq + der) >> 1, i, j);
        int aux2 = query(mov_der(pos), ((izq + der) >> 1) + 1, der, i, j);
        if(aux1 == -1) return aux2;
        if(aux2 == -1) return aux1;

        return min(aux1, aux2);
    }

    int RMQ(int i, int j){//metodo a invocar
        return query(1, 0, n-1, i, j);
    }

    int cambiar(int pos, int izq, int der, int index, int nuevo){
        if(index > der || index < izq) return st[pos];
        if(der == index && izq == index){
            A[index] =  nuevo;
            return st[pos] = nuevo;
```

```
        }

        int aux1 = cambiar(mov_izq(pos), izq, (izq + der) >> 1, index, nuevo);
        int aux2 = cambiar(mov_der(pos), ((izq + der) >> 1) + 1, der, index, nuevo);
        return st[pos] = min(aux1, aux2);
    }

    int update(int index, int num){//metodo a invocar
        return cambiar(1, 0, n-1, index, num);
    }
};
```

1.4. Segment tree con lazy propagation

Permite actualizar rangos del árbol en O(log n).
solo están los métodos nuevos y los que hay que actualizar, lo demás es lo mismo del segment tree normal.

```
    int lazy[MAX];

    void construir(int pos, int izq, int der){
        lazy[pos] = -1;//reiniciar lazy
        if(izq == der){
            st[pos] = A[der];
            return;
        }

        construir(mov_izq(pos), izq, (izq + der) >> 1);
        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
        st[pos] = min(st[aux1], st[aux2]);
    }

    int query(int pos, int izq, int der, int i, int j){
        if(i > der || j < izq) return -1;
        solve_lazy(pos, izq, der);//resolver algun lazy pendiente
        if(i <= izq && j >= der) return st[pos];

        int aux1 = query(mov_izq(pos), izq, (izq + der) >> 1, i, j);
        int aux2 = query(mov_der(pos), ((izq + der) >> 1) + 1, der, i, j);
        if(aux1 == -1) return aux2;
        if(aux2 == -1) return aux1;

        return min(aux1, aux2);
    }

    void solve_lazy(int pos, int izq, int der){//resolver lazy
        if(lazy[pos] == -1) return;

        st[pos] = lazy[pos];
        if(izq != der){
            lazy[mov_izq(pos)]=lazy[mov_der(pos)]=lazy[pos];
        }
        lazy[pos] = -1;
    }

    int lazy_propagation(int pos, int izq, int der, int i, int j, int nuevo){
        solve_lazy(pos, izq, der);
        if(i > der || j < izq) return st[pos];

        if(i <= izq && j >= der){
            lazy[pos] = nuevo;
            solve_lazy(pos, izq, der);
            return st[pos];
        }

        int aux1 = lazy_propagation(mov_izq(pos), izq, (izq + der) >> 1, i, j, nuevo);
        int aux2 = lazy_propagation(mov_der(pos), ((izq + der) >> 1) + 1, der, i, j, nuevo);
        return st[pos] = min(aux1, aux2);
    }

    int update(int i, int j, int nuevo){//metodo a invocar
        return lazy_propagation(1, 0, n-1, i, j, nuevo);
    }
```

1.5. Arbol de Fenwick

Estructura para el RSM(Range Sum Query)
Construcción O(n log n), Consulta O(log k), Update O(log n)

```
struct FenwickTree{
    vi ft;
    //indexamos desde 1
    void iniciar(int n){ ft.assign(n + 1, 0); }

    void iniciar(vi &v){
        ft.assign(v.size() + 1, 0);
        for(int i = 1; i <= v.size(); i++)
            actualizar(i, v[i - 1]);
    }
    //bit menos significativo en 1
    int lsOne(int n){ return n & (-n); }

    int rsq(int i){//suma de 1 hasta i
        int acum = 0;
        for(; i; i -= lsOne(i)) acum+=ft[i];
        return acum;
    }

    int rsq(int i, int j){//suma de i hasta j
        return rsq(j) - ((i==1)? 0: rsq(i - 1));
    }

    void actualizar(int pos, int n){//n = nuevo - anterior
        for(; pos < ft.size(); pos += lsOne(pos))
            ft[pos] += n;
    }
};
```

1.6. Sparse table

Para RMQ (Range Minium Query) en arreglos estaticos
Construcción O(n log n), Consulta O(1)

```
#define MAX 1000 //n
#define Log2 10 //2^10 > n
int arr[MAX], spt[MAX][Log2];

struct sparseTable{
    sparseTable(){}

    sparseTable(int n, int a[]){
        memset(spt, 0, sizeof(spt));
        for(int i = 0; i < n; i++){
            arr[i] = a[i];  spt[i][0] = i;
        }

        for(int j = 1; (1<<j) <= n; j++){
        for(int i=0; i+(1<<j)-1 < n; i++)
            if(arr[spt[i][j-1]] < arr[spt[i+(1<<(j-1))][j-1]])
                spt[i][j] = spt[i][j-1];
            else spt[i][j] = spt[i+(1<<(j-1))][j-1];
        }
    }

    int query(int i, int j){//de i hasta j, index desde 0
        int k = (int) floor(log(((j-i+1)*1.0))/log(2.0));
        if(arr[spt[i][k]] <= arr[spt[j-(1<<k)+1][k]])
            return spt[i][k];
        else return spt[j-(1<<k)+1][k];
    }
};
```

1.7. Set extendido

Set indexado.

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

typedef tree<int,null_type,less<int>,
    rb_tree_tag,tree_order_statistics_node_update> set_t;
//it = s.find_by_order(k); iterador al k-esimo elemento
//x = s.order_of_key(k); posicion del lower_bound de k
```

2. Grafos

2.1. Dijkstra

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

```
vi padre;//opcional, usar cuando se necesite el camino.

vi dijkstra(vvii &grafo, int nodo, int tam){
    padre.assign(tam + 1, -1);
    priority_queue<ii> cola;
    cola.push(ii(-0, nodo));
    vi dis(tam + 1, inf);  dis[nodo] = 0;
    int peso, aux;  ii par, par2;

    while(cola.size()){
        par = cola.top();//peso, nodo
        cola.pop();
        peso = -par.first;  nodo = par.second;

        for(int i = 0; i < grafo[nodo].size(); i++){
            par2 = grafo[nodo][i];
            aux = dis[nodo] + par2.first;
            if(dis[par2.second] > aux){
                dis[par2.second] = aux;
                cola.push(ii(-aux, par2.second));
                padre[par2.second] = nodo;
            }
        }
    }

    return dis;
}

void camino(int n){//imprimir el camino
    if(padre[n] == -1) printf("%d", n);
    else{
        camino(padre[n]);
        printf(" %d", n);
    }
}
```

2.2. Bellman-Ford

Ruta minima con pesos negativos O(n^2)

```
vector<iii> grafo; //lista de incidencia

bool BellmanFord(vector<iii> &lista, int nodos, int inicio,
        vector<int> &dis){
    dis.assign(nodos + 1, inf);
    dis[inicio] = 0;
    int aux;

    for (int i = 0; i < nodos; i++)
        for (int j = 0; j < lista.size(); j++) {
            aux = dis[lista[j].second.first] + lista[j].first;
            if(dis[lista[j].second.second] > aux)
                dis[lista[j].second.second] = aux;
        }

    for(int j = 0; j < lista.size(); j++){
        aux = dis[lista[j].second.first] + lista[j].first;
        if(dis[lista[j].second.second] > aux)
            return false;//existe ciclo!!!
    }
    return true;
}
```

2.3. Floyd Warshall

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

```
int cam[10][10], matriz[10][10];

void imprimirCamino(int f, int c){
    if(cam[f][c] == f){
        printf("%d", f);  return;
    }else{
        imprimirCamino(f, cam[f][c]);
        printf(" %d", cam[f][c]);
    }
}

```

```
void FloydWarshall(int nodos){
    int aux;
    //si no necesita caminos, solo hacer la diagonal cero
    for(int i = 0; i < nodos; i++)
        for(int j = 0; j < nodos; j++){
            if(i == j) matriz[i][j] = 0;
            if(i != j && matriz[i][j] != inf) cam[i][j] = i;
        }

    for(int k = 0; k < nodos; k++)
    for(int i = 0; i < nodos; i++)
    for(int j = 0; j < nodos; j++){
        aux = matriz[i][k] + matriz[k][j];
        if(matriz[i][j] > aux){
            matriz[i][j] = aux;  cam[i][j] = cam[k][j];
        }
    }
}
```

2.4. kosaraju

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

```
int n, m;
vector<vi> grafo(100), grafoT(100);
vector<int> ts;  bool vis[100];

void dfs(int u, int pass){
    vis[u] = true;  vi vecino;
    if(pass == 1) vecino = grafo[u];
    else vecino = grafoT[u];

    for(int i = 0; i < vecino.size(); i++)
        if(!vis[vecino[i]]) dfs(vecino[i], pass);
    ts.push_back(u);
}

int kosaraju(){
    ts.clear();
    memset(vis, 0, sizeof(vis));
    for(int i = 0; i < n; i++)
        if(!vis[i]) dfs(i, 1);

    int num_comp = 0;
    memset(vis, 0, sizeof(vis));
    for(int i = ts.size()-1; i >= 0; i--)
        if(!vis[ts[i]]){
            num_comp++;  dfs(ts[i], 2);
        }
    return num_comp;
}
```

2.5. Tarjan

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

```
vi dfs_low, dfs_num, s; vector<bool> vis;
int dfsCont;

void dfs(int u){
    dfs_low[u] = dfs_num[u] = dfsCont++;
    s.push_back(u);  vis[u] = true;

    for(int i = 0; i < lista[u].size(); i++){
        int aux = lista[u][i];
        if(dfs_num[aux] == -1) dfs(aux);
        if(vis[aux])
            dfs_low[u] = min(dfs_low[u], dfs_low[aux]);
    }

    if(dfs_low[u] == dfs_num[u]){
        printf("comp:\n");
        while(true){
            int v = s.back(); s.pop_back();
            printf(" %d\n", v);  vis[v] = false;
            if(v == u) break;
        }
        printf("\n");
    }
}

```

```
void tarjan(){
    dfs_num.assign(n+1,-1); dfs_low.assign(n+1,0);
    vis.assign(n+1, false);  dfsCont = 0;
    for(int i = 0; i < n; i++)
        if(dfs_num[i] == -1) dfs(i);
}
```

2.6. Kruskal

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

```
typedef pair<int, ii> iii;//peso, origen y destino
vector<iii> listaInc;//lista de incidencia
union_find arbol;

int kruskal(vector<iii> lista, int nodos, union_find &uf){
  sort(lista.begin(), lista.end());
  uf.iniciar(nodos);
  int acum = 0, ejes = 0, n = nodos - 1;

  for (int i = 0; i < lista.size(); i++) {
    if (!uf.MismoGrupo(lista[i].second.first,
                      lista[i].second.second)) {
      ejes++;
      uf.unir(lista[i].second.first, lista[i].second.second);
      acum += lista[i].first;
      if(ejes == n) return acum;
    }
  }
  return -1;
}
```

2.7. Prim

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

```
priority_queue<ii> cola;
vector<bool> vis;

void vecinos(vvii &lista, int nodo){
    vis[nodo] = true;
    for(int i = 0; i < lista[nodo].size(); i++){
        ii par = lista[nodo][i];//peso - destino
        if(!vis[par.second])
            cola.push(ii(-par.first, -par.second));
    }
}

int prim(vvii &lista, int n){
    vis.assign(n + 1, false);
    vecinos(lista, 1);
    int acum = 0; ii par;

    while(cola.size()){
        par = cola.top(); cola.pop();
        if(vis[-par.second]) continue;
        acum += -par.first;
        vecinos(lista, -par.second);
    }
    return acum;
}
```

2.8. Topological sort

O(n + m), algoritmo de kahn.

```
vector<int> res;//guarda la respuesta.
vector<int> ent;//se debe llenar con la cantidad de
                //aristas entrantes que tiene cada nodo.
void topological_sort(vvi &lis, int tam){
    res.clear();
    queue<int> s;
    for(int i = 1; i < tam; i++)
        if(!ent[i]) s.push(i);

    int n, m;
    while(s.size()){
        n = s.front();  s.pop();
        res.push_back(n);
        for(int i = 0; i < lis[n].size(); i++){
            m = lis[n][i];
            ent[m]--;
```

```
            if(!ent[m]) s.push(m);
        }
    }
}
```

2.9. Puntos de articulación y puentes

O(n + m).

```
vi puntos, dfs_num, dfs_low, padre;
int n, m, dfsCont, root, dfsRoot;
vector<ii> puentes;//guarda los puentes

void dfs(int u){
    dfs_low[u] = dfs_num[u] = dfsCont++;
    int aux;
    for(int i = 0; i < lista[u].size(); i++){
        aux = lista[u][i];
        if(dfs_num[aux] == -1){
            padre[aux] = u;
            if(u == dfsRoot) root++;
            dfs(aux);

            if(dfs_low[aux]>=dfs_num[u]) puntos[u]++;
            if(dfs_low[aux] > dfs_num[u])
                puentes.push_back(ii(aux, u));
            dfs_low[u] = min(dfs_low[u], dfs_low[aux]);
        }else if(aux != padre[u])
            dfs_low[u] = min(dfs_low[u], dfs_num[aux]);
    }
}

void solve(){
    puntos.assign(n, 1); dfs_low.assign(n, 0);
    padre.assign(n, 0); dfs_num.assign(n, -1);

    for(int i = 0; i < n; i++)
        if(dfs_num[i] == -1){
            dfsCont = root = 0; dfsRoot = i;
            dfs(dfsRoot);
            puntos[i] = root - 1;
        }

    printf("puntos␣de␣articulacion:\n");
    for(int i = 0; i < n; i++)
        if(puntos[i] > 1)//cantidad de componentes
            printf("%d,␣conecta␣%d␣comp.\n",i,puntos[i]);
}
```

2.10. Maximum flow

Flujo maximo en un grafo. Algoritmo de Edmonds Karp O(VE^2)

```
int start, target, MAX=110, mf, f, matriz[110][110];
vi p;  vvi grafo;//matriz inicialmente se debe llenar de ceros

void augment(int v, int minEdge){
    if(v == start){ f = minEdge; return; }
    else if(p[v] != -1){
        augment(p[v], min(minEdge, matriz[p[v]][v]));
        matriz[p[v]][v] -= f;  matriz[v][p[v]] += f;
} }

int EdmondsKarp(){
    mf = 0;
    while(true){
        f = 0;
        vector<bool> vis(MAX, false); vis[start] = true;
        queue<int> cola;  cola.push(start);
        p.assign(MAX, -1);
        while(cola.size()){
            int u = cola.front();  cola.pop();
            if(u == target) break;

            for(int j = 0; j < grafo[u].size(); j++){
                int v = grafo[u][j];
                if(matriz[u][v] > 0 && !vis[v]){
                    vis[v] = true;
                    cola.push(v);  p[v] = u;
        }}}
        augment(target, inf);
        if(f == 0) break;
        mf += f;
    }
```

```
    return mf;
}

void addEdgeUndirected(int x, int y, int z){
    grafo[x].push_back(y);  grafo[y].push_back(x);
    matriz[x][y] += z;  matriz[y][x] += z;
}
void addEdgeDirected(int x, int y, int z){
    grafo[x].push_back(y);  grafo[y].push_back(x);
    matriz[x][y] += z;  matriz[y][x] += 0;
}
```

2.11. Min cost Max flow

Flujo maximo manteniendo minimo costo.

```
const lli INFFLUJO=1e14, MAXN = 100010;
lli dist[MAXN], min_cost, cap[MAXN], max_flow;
int pre[MAXN];  bool en_cola[MAXN];//int n;

struct edge {
    int u, v;  lli cap, flow, cost;
    lli rem(){return cap-flow;}
};
vector<edge> aristas;  vector<int> grafo[MAXN];

void add_edge(int u, int v, lli cap, lli cost) {
    grafo[u].push_back(aristas.size());
    aristas.push_back(edge{u,v,cap,0,cost});
    grafo[v].push_back(aristas.size());
    aristas.push_back(edge{v,u,0,0,-cost});
}//Agrega aristas unidireccionales

void flow(int s, int t){
    memset(en_cola,0,sizeof(en_cola));
    max_flow = min_cost = 0;

    while(1){
        memset(dist, 3586, sizeof(dist));//inf 10^17
        memset(pre, -1, sizeof(pre));
        memset(cap, 0, sizeof(cap));
        pre[s] = dist[s] = 0;
        cap[s] = INFFLUJO;
        queue<int> cola;
        cola.push(s);  en_cola[s]=1;

        while(cola.size()){
            int u = cola.front();  cola.pop();  en_cola[u]=0;
            for(int j = 0; j < grafo[u].size(); j++){
                int i = grafo[u][j];
                edge &E = aristas[i];
                if(E.rem() && dist[E.v]>dist[u]+E.cost+1e-9){
                    dist[E.v] = dist[u]+E.cost;
                    pre[E.v]=i;
                    cap[E.v]= min(cap[u], E.rem());
                    if(!en_cola[E.v]){
                        cola.push(E.v);  en_cola[E.v] = 1;
                    }
                }
            }
        }

        if(pre[t] < 0) break;
        max_flow+=cap[t];  min_cost+=cap[t]*dist[t];
        for(int v = t; v != s; v = aristas[pre[v]].u){
            aristas[pre[v]].flow += cap[t];
            aristas[pre[v]^1].flow -= cap[t];
        }
    }
}
```

2.12. Ruta minima en un DAG

O(V+2E)

```
bool vis[110];  vi ts;

void dfs(vvii &lista, int nodo){
    vis[nodo] = 1;  ii par;
    for(int i = 0; i < lista[nodo].size(); i++){
        par = lista[nodo][i];
        if(!vis[par.second]) dfs(lista, par.second);
    }
    ts.push_back(nodo);
```

```
}

void topological_sort(vvii &lis, int tam){
    for(int i = 0; i < tam; i++)
        if(!vis[i]) dfs(lis, i);
    reverse(ts.begin(), ts.end());
}

vi sp_DAG(vvii &lista, int n){
    topological_sort(lista, n);
    vi dist(n, inf);
    ii par;  int aux;
    dist[ts[0]] = 0;

    for(int i = 0; i < n; i++)
        for(int j = 0; j < lista[ts[i]].size(); j++){
            par = lista[ts[i]][j];
            aux = dist[ts[i]] + par.first;
            if(dist[par.second] > aux){
                dist[par.second] = aux;
            }
        }
    return dist;
}
```

2.13. Tour de Euler

```
typedef list<int>::iterator lii;
int degree[100];
vector<vector<pair<int, bool>>> lista;//destino, visitado
list<int> cyc;

void EulerTour(lii i, int u){
    for(int j = 0; j < lista[u].size(); j++){
        ib v = lista[u][j];
        if(v.second){
            v.second = false;
            lista[u][j].second = false;
            for(int k = 0; k < lista[v.first].size(); k++){
                ib uu = lista[v.first][k];
                if(uu.first==u && uu.second){
                    uu.second = false;
                    lista[v.first][k].second = false;
                    break;
                }
            }
            //inserta conexion (v.first,u)
            EulerTour(cyc.insert(i, u), v.first);
        }
    }
}
```

2.14. 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).

```
#define MAX 100
int l[2*MAX], e[2*MAX], h[MAX], idx;
sparseTable table;

void dfs(int nodo, int deep, vvi &grafo){
    h[nodo] = idx;
    e[idx] = nodo;
    l[idx++] = deep;

    for(int i = 0; i < grafo[nodo].size(); i++){
        if(h[grafo[nodo][i]] != -1) continue;
        dfs(grafo[nodo][i], deep+1, grafo);
        e[idx] = nodo;  l[idx++] = deep;
    }
}

void BuildRMQ(vvi &grafo){//llamar antes de LCA
    idx = 0;
    memset(h, -1, sizeof(h));
    memset(l, -1, sizeof(l));
    dfs(0, 0, grafo);
    table = sparseTable(grafo.size()<<1, l);
}

int LCA(int u, int v){//h[u] < h[v]
    if(h[u] > h[v]) swap(u, v);
```

```
    return e[table.query(h[u], h[v])];
}
```

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

```
#define log2 16

int p[MAX], d[MAX], spt[MAX][log2];
int mayor[MAX][log2], peso[MAX];
vvii grafo;

void dfs(int nodo, int deep, int ant){
    d[nodo] = deep;
    p[nodo] = ant;
    ii par;

    for(int i = 0; i < grafo[nodo].size(); i++){
        par = grafo[nodo][i];
        if(d[par.second] != -1) continue;
        peso[par.second] = par.first;
        dfs(par.second, deep+1, nodo);
    }
}

void proceso(int n){//spt[i][j] = (2^j)-th ancestro de i
    memset(d, -1, sizeof(d));
    memset(mayor, 0, sizeof(mayor));
    memset(peso, 0, sizeof(peso));
    dfs(0, 0, -1);
    memset(spt, -1, sizeof(spt));
    for(int i = 0; i < n; i++){
        spt[i][0] = p[i];
        mayor[i][0] = peso[i];
    }

    for(int j = 1; 1 << j < n; j++)
    for(int i = 0; i < n; i++)
        if(spt[i][j-1] != -1){
            spt[i][j] = spt[spt[i][j-1]][j-1];
            //opcional, para obtener suma o mayor arista
            mayor[i][j] = max(mayor[i][j-1], mayor[spt[i][j-1]][j-1]);
            //mayor[i][j] = mayor[i][j-1] + mayor[spt[i][j-1]][j-1];
        }
}

//Opcional, para recorrer los nodos entre u y V
int maxEdge(int u, int v){
    if(d[u] < d[v]) swap(u, v);
    int may = 0, suma = 0;
    for(int i = 16; i >= 0; i--)
        if(d[u] - (1<<i) >= d[v]){
            may = max(may, mayor[u][i]);
      //suma += mayor[u][i];//en caso de requerir suma
            u = spt[u][i];
        }
    return may;
}

int LCA(int u, int v){
    if(d[u] < d[v]) swap(u, v);//v debe estar arriba de u
    for(int i = 16; i >= 0; i--)//subimos a u
        if(d[u] - (1<<i) >= d[v])
            u = spt[u][i];
    if(u == v) return u;

    for(int i = 16; i >= 0; i--)
        if(spt[u][i] != spt[v][i]){
            u = spt[u][i];
            v = spt[v][i];
        }
    return p[v];
}
```

2.15. Conectividad dinámica

```
struct UnionFind {
  int n,comp;
  vector<int> uf,si,c;
  UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1){
    for(int i=0; i<n; ++i) uf[i]=i;
    }
  int find(int x){return x==uf[x]?x:find(uf[x]);}
```

```
  bool join(int x, int y){
    if((x=find(x))==(y=find(y)))return false;
    if(si[x]<si[y])swap(x,y);
    si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
    return true;
  }
  int snap(){return c.size();}
  void rollback(int snap){
    while(c.size()>snap){
      int x=c.back();c.pop_back();
      si[uf[x]]-=si[x];uf[x]=x;comp++;
    }
  }
};
enum {ADD,DEL,QUERY};
struct Query {int type,x,y;};
struct DynCon {
  vector<Query> q;
  UnionFind dsu;
  vector<int> mt;
  map<pair<int,int>,int> last;
  DynCon(int n):dsu(n){}
  void add(int x, int y){
    if(x>y) swap(x,y);
    q.pb((Query){ADD,x,y}),mt.pb(-1);
    last[{x,y}]=q.size()-1;
  }
  void remove(int x, int y){
    if(x>y)swap(x,y);
    q.pb((Query){DEL,x,y});
    int pr=last[{x,y}];
    mt[pr]=q.size()-1;
    mt.pb(pr);
  }
  void query(){
    q.pb((Query){QUERY,-1,-1});mt.pb(-1);}
  void process(){ // answers all queries in order
    if(!q.size()) return;
    for(int i=0; i<q.size(); ++i)
            if(q[i].type==ADD&&mt[i]<0)mt[i]=q.size();
    go(0,q.size());
  }
  void go(int s, int e){
    if(s+1==e){
      if(q[s].type==QUERY) // answer query using DSU
        printf("%d\n",dsu.comp);
      return;
    }
    int k=dsu.snap(),m=(s+e)/2;
    for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s)dsu.join(q[i].x,q[i].y);
    go(s,m);
    dsu.rollback(k);
    for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join(q[i].x,q[i].y);
    go(m,e);
    dsu.rollback(k);
  }
};
```

3. Programacion dinamica

3.1. Subconjuntos de un conjunto

$O(2^n)$

```
void mask(int n, int ar[]){
    int l = 1 << n;

    for(int i = 0; i < l; i++){
        for(int j = 0; j < n; j++)
            if(i & (1 << j))
                printf("%d ", ar[j]);
        printf("\n");
    }
}
```

3.2. Problema de la mochila

```
int ganancia[100] = {100, 70, 50, 10};
int peso[100] = {10, 4, 6, 12};

int knapsack(int cap, int n) {//capacidad y cantidad.
    int dp[n+1][cap+1];
```

```
    for(int i = 0; i <= n; i++)//recorrer objetos
        for(int j = 0; j <= cap; j++){
            if(i == 0 || j == 0) dp[i][j] = 0;//caso base
            else if(peso[i - 1] <= j)
                dp[i][j] = max(dp[i - 1][j],
                            ganancia[i - 1] + dp[i - 1][j - peso[i - 1]]);
            else
                dp[i][j] = dp[i - 1][j];
        }
    return dp[n][cap];
}
```

3.3. Longest Increment Subsecuence

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

```
int LIS_dp(){
    int res = 0;
    vector<int> vec(8, 1);

    for(int i = 0; i < 8; i++)
        for(int j = i + 1; j < 8; j++)
            if(A[i] < A[j]) vec[j] = max(vec[j], vec[i] + 1);
        res = max(res, vec[i]);
    return res;
}
```

Solución D&C con gredy, O(n log n)

```
int A[] = {-7, 10, 9, 2, 3, 8, 8, 6};
int aux[10], lis[10], indexAnt[10], n = 8;

void mostrar(int pos){
    stack<int> pila;
    while(pos != -1)
        pila.push(pos), pos = lis[pos];

    while(pila.size()){
        printf("%d\n", A[pila.top()]);
        pila.pop();
    }
}

void imp(int ar[], int v){
    for(int i = 0; i < v; ++i) printf("%3d",ar[i]);
    printf("\n");
}

//Para decreciente invertir el signo de los numeros
void LIS(){                         //en el arreglo.
    int tam = 0, pos, res = 0;
    for(int i = 0; i < n; i++){
        pos = lower_bound(aux, aux + tam, A[i]) - aux;
        imp(aux,tam);
        printf("pos = %d hasta %d, buscando %d\n", pos,tam, A[i]);
        //usar upper_bound para contar repetidos
        aux[pos] = A[i];
        indexAnt[pos] = i;
        lis[i] = pos;
        lis[i] = pos? indexAnt[pos-1]: -1;
        if(pos + 1 > tam){
            tam = pos + 1;
            res = i;
        }
    }

    printf("longitud: %d\n", tam);
    mostrar(res);
}
```

3.4. Max Range Sum

Algoritmo de Kadane, O(n)

```
int main(){
    int n, num, res, aux;

    while(scanf("%d", &n), n){
        res = aux = 0;
        for(int i = 0; i < n; i++){
            scanf("%d", &num);
            aux += num;
            res = max(aux, res);
            if(aux < 0) aux = 0;
```

```
        }

        if(res > 0) printf("MRS = %d\n", res);
        else printf("negativo.\n");
    }
    return 0;
}
```

3.5. Subset Sum

```
bool dp[10][50];//fila cantidad de numeros
//columas rango maximo a evaluar

void pre(vi &num){
    memset(dp, false, sizeof(dp));

    for(int i = 0; i < num.size(); i++){
        if(i) for(int j = 1; j < 50; j++)
            if(dp[i - 1][j]) dp[i][j + num[i]] = true;

        dp[i][num[i]] = true;
    }
}
```

3.6. Traveling salesman problem

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

```
int MAX;//luego de leer n hacer: MAX = (1<<n)-1;
int matriz[15][15], memo[15][(1<<15)+1], n;

int tsp(int pos, int mask){
    if(mask == MAX) return matriz[pos][0];
    if(memo[pos][mask] != -1)
        return memo[pos][mask];

    int res = 1000000000;
    for(int i = 0; i < n; i++)
        if(!(mask & (1<<i))){
            res = min(res, matriz[pos][i]
                    + tsp(i, mask | (1<<i)));
        }
    return memo[pos][mask] = res;
}
```

4. Otros

4.1. Busqueda binaria

O(log n)

```
int f(int a, int b){return ar[a] > b;}
int busqueda_binaria(int men, int may, int v){
    int epsilon = 1, med = 0;
    while(may-men > epsilon){
        med = (may+men)/2;
        if(f(med,v)) may = med;
        else men = med;
    }
    return men;
}
//lower_bound -> cambiar f por: f(a,b){return ar[a] < b;}
//  cambiar "return men" por: return (f(med,v))? may: men;
//  y por ultimo invertir "may=med;" con "men = med;"
```

4.2. Raiz babilonica

Encuentra la raiz cuadrada de un numero

```
double raiz(double x) {
    double b = x, h = 0, apro = 1;
    while (apro > 1e-8) {
        b = (h + b) / 2;
        h = x / b;
        apro = abs(h - b);
    }
    return b;
}
```

4.3. Codigo gray

```
int gray(int n) { return n ^ (n >> 1); }

```

```
int num(int gray) {//invertir
  int n = 0;
  for (; gray; gray >>= 1)
    n ^= gray;
  return n;
}
```

4.4. Método de Wilson

Encuentra n! mod p, con p primo, requiere el inverso modular

```
int metodoWilson(int n, int p)  {
  if (p <= n) return 0;
  int res = (p - 1);

  for (int i = n + 1; i < p; i++)
    res = (res * inverso(i, p)) % p;
  return res;
}
```

5. Matematicas

5.1. MCD y MCM

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

```
int mcd(int a,int b){return a? mcd(b%a,a): b;}
int mcm(int a,int b){return a*(b/mcd(a,b));}
```

5.2. Phi de euler

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

```
int phi(int n) {
    int result = n;
    for(int i=2; i*i<=n; ++i)
        if(n % i == 0) {
            while(n % i == 0) n /= i;
            result -= result / i;
        }
    if(n > 1) result -= result / n;
    return result;
}
```

5.3. Algoritmo extendido de euclides

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

```
int gcd_ex(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
  int x1, y1;
  int d = gcd_ex(b, a%b, x1, y1);
  x = y1;
  y = x1 - (a/b)*y1;
  return d;//Maximo comun divisor
}
```

5.4. Ecuaciones diofanticas

Resuelve ecuaciones de la forma aX+bY=c

```
void solve(lli a, lli b, lli c){
    double q, w;
    lli x, y, d, xx, yy, men, may;
    d = gcd_ex(a,b,x,y);
    q = (double) x*(c/b);
    w = (double) y*(c/a);
    men = (lli) ceil(-1.0*q);
    may = (lli) floor(w);

    if(c%d || may < men){
        printf("Sin solucion\n");
        return;
    }
    for(lli i=men; i<=may; ++i){
        xx = x*(c/d)+((b/d)*i);
        yy = y*(c/d)-((a/d)*i);
        printf("solucion (%lld , %lld)\n",xx,yy);
    }
}
```

5.5. Exponenciacion binaria

O(log n)

```
lli exp_bin(lli a, lli n) {
    lli res = 1;
    while (n) {
        if (n & 1) res *= a;
        a *= a;  n >>= 1;
    }
    return res;
}
```

5.6. Multiplicacion modular

Encuentra (a*b) mod 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.

```
typedef long long int lli;//metodo normal
lli mulmod (lli a, lli b, lli c) {
  lli 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;
}

typedef __int128 bi; //metodo con __int128
lli mulmod_2(bi a, bi b, bi c){
    return (lli) ((a*b) % c);
}

int main(){
    lli a, b, c;
    cin >> a >> b >> c;
    cout << mulmod_2((bi) a, (bi) b, (bi) c) << endl;
    return 0;
}
```

5.7. Exponenciacion modular

Encuentra (a^b) mod c, se nesecita implementar previamente multiplicacion modular.

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

5.8. Inverso modular

Encontrar x talque $a * x \equiv 1 \% m$, el primer método requiere algoritmo extendido de euclides y el segundo exponenciacion modular

```
int inverso1(int a, int m){
    int x, y;
    int g = gcd_ex (a, m, x, y);
    if (g != 1) return -1;
    else  return (x % m + m) % m;
}

int inverso2(int a, int m) {
    return expmod(a, m - 2, m);
}
```

5.9. Logaritmo discreto

Encuentra una solución para $a^x \equiv 1(mod(p))$.
O($\sqrt{m}log(m)$)

```
int solve(int a, int b, int m){
    int n = (int) sqrt(m + 0.0) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i) an = (an * a) % m;

    map<int, int> vals;
    for (int p = 1, cur = an; p <= n; ++p) {
        if (!vals.count(cur)) vals[cur] = p;
        cur = (cur * an) % m;
    }
    for (int q = 0, cur = b; q <= n; ++q) {
```

```
        if (vals.count(cur)) {
            int ans = vals[cur] * n - q;
            return ans;
        }
        cur = (cur * a) % m;
    }
    return -1;
}
```

5.10. Test de Rabin Miller

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

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

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

  for(int i = 0; i < s-1; i++){
    x = mulmod(x, x, n);
    if (x == 1) return false;
    if (x+1 == n) return true;
  }
  return false;
}

bool rabin (lli n){ //devuelve true si n es primo
  if (n == 1) return false;
  const int ar[] = {2,3,5,7,11,13,17,19,23};
  for(int j = 0; j < 9; j++)
    if (!es_primo_prob(n,ar[j]))
      return false;
  return true;
}
```

5.11. 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.
$O(\sqrt[4]{n})$

```
lli rho(lli n){
    if( (n & 1) == 0 ) return 2;
    lli x = 2 , y = 2 , d = 1;
    lli 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 = mcd( x - y , n );
        else d = mcd( y - x , n );
    }
    return d==n? rho(n):d;
}

map<lli, lli> prim;

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

5.12. Factorizacion con criba

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

```
int m = 1000010, primo[1000020];
vector<lli> p;  int lim = sqrt(m)+1;
map<lli, int> mapa;

```

```
void criba(){
    memset(primo, 0, sizeof(primo));

    for(int i = 2; i < m; i++){
        if(primo[i]) continue;
        p.push_back(i);
        primo[i] = i;
        if(i > lim) continue;

        for(int j = i*i; j < m; j += i)
            primo[j] = i;
    }
}

void factCriba(int n){
    int l, pos;
    while(n != 1){
        if(n >= m){//n mayor a logintud del array
            l = sqrt(n) + 1;  pos = -1;
            while(p[++pos] <= l)
                if(n % p[pos] == 0){
                    mapa[p[pos]]++;
                    n /= p[pos];
                    break;
                }
            if(p[pos] > l){
                mapa[n]++;
                break;
            }
        }else{
            mapa[primo[n]]++;
            n /= primo[n];
        }
    }
}
```

5.13. Divisores

Encontrar los divisores de un numero según su factorización

```
void divisores(){
    set<lli> num;  num.insert(1);
    for(auto it=prim.begin(); it!=prim.end(); ++it){
        lli n = it->first, x;
        int c = it->second;
        for(int i=0; i<c; ++i){
            vector<lli> v;
            for(auto it2=num.begin();it2!=num.end();++it2){
                x = *it2;
                v.push_back(x*n);
            }
            num.insert(v.begin(), v.end());
        }
    }

    for(auto it2=num.begin(); it2!=num.end(); ++it2){
        printf("%lld\n", *it2);
    }
}
```

5.14. Fraccion

```
struct fraccion {
    int num, den;

    fraccion(int x, int y) {
        num = x; den = y;
        if (num == 0) den = 1;
        else {
            int dividir = mcd(num, den);
            num /= dividir;
            den /= dividir;
        }
        if (den < 0){ num *= -1;  den *= -1; }
    }

    fraccion operator+(fraccion b) {//suma
        return fraccion(num*b.den + b.num*den, den*b.den);
    }
    fraccion operator-(fraccion b) {//resta
        return fraccion(num*b.den - b.num*den, den*b.den);
    }
    fraccion operator*(fraccion b) {//multiplicar
```

```
        return fraccion(num*b.num, den*b.den);
    }
    fraccion inversa() {
        return fraccion(den, num);
    }
    fraccion operator/(fraccion b) {//dividir
        return fraccion(num*b.den, b.num*den);
    }
    string toString() {
        stringstream ss;
        ss << num;
        if (den == 1) return ss.str();
        ss << "/"; ss << den;
        return ss.str();
    }
};
```

5.15. Matrices

Exponenciación de matrices: M^b en $O(n^3 log(b))$

```
struct matrix{  lli mat[maxn][maxn];  };

matrix matmul(matrix &a, matrix &b){//multiplicar
    matrix ans;
    int i, j, k;

    for(i = 0; i < maxn; i++)
    for(j = 0; j < maxn; j++)
        for(ans.mat[i][j] = k = 0; k < maxn; k++)
            ans.mat[i][j] += (a.mat[i][k] * b.mat[k][j]);

    return ans;
}

matrix matpow(matrix base, int p){//exp binaria
    matrix ans;
    int i, j;

    for(i = 0; i < maxn; i++)
        for(j = 0; j < maxn; j++)
            ans.mat[i][j] = (i == j);

    while(p){
        if(p&1) ans = matmul(ans, base);
        base = matmul(base, base);
        p >>= 1;
    }
    return ans;
}
```

5.16. FFT

Multiplicación rápida de polinomios y enteros, complejidad O(n*log(n))

```
typedef complex<double> base;
int rev[600010];//Longitun >= n

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

  for(int i=0; i<n; ++i)
    if (i < rev[i])
      swap (a[i], a[rev[i]]);

  for(int len=2; len<=n; len<<=1) {
    double ang = 2*PI/len * (invert ? -1 : 1);
    base wlen (cos(ang), sin(ang));
    for (int i=0; i<n; i+=len) {
      base w(1);
      for (int j=0; j<len/2; ++j) {
        base u = a[i+j],  v = a[i+j+len/2] * w;
        a[i+j] = u + v;
        a[i+j+len/2] = u - v;
        w *= wlen;
      }
    }
  }
  if(invert)
    for(int i=0; i<n; ++i)
      a[i] /= n;
}
//Llamar este metodo antes del primer llamado a multiply
void calc_rev(int n, int log_n){
  for(int i=0; i<n; ++i) {
```

```
    rev[i] = 0;
    for(int j=0; j<log_n; ++j)
      if(i & (1<<j))
        rev[i] |= 1<<(log_n-1-j);
  }
}
//n >= a.size()+b.size(), y debe ser potencia de 2
void multiply(vector<int> &a, vector<int> &b, int &n, vector<int>
    &res) {
    vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    fa.resize(n);  fb.resize(n);

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

    res.resize(n);
    for(int i=0; i<n; ++i)
        res[i] = int(fa[i].real() + 0.5);

    //Omitir lo de abajo para multiplicar polinomios
    //Acarreos para la multiplicacion de numeros
    int carry = 0;
    for(int i=0; i<n; ++i) {
        res[i] += carry;
        carry = res[i] / 10;
        res[i] %= 10;
    }
}
```

6. Cadenas

6.1. Algoritmo de bordes

Encuentra la longitud del mayor borde de un string n.

```
int bordes[1000];

void algoritmoBordes(string subcad){
    int i = 0, j = -1;
    bordes[0] = -1;

    while(i < subcad.size()) {
        while(j >= 0 && subcad[i] != subcad[j])
            j = bordes[j];
        i++; j++;
        bordes[i] = j;
    }
}
```

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)

```
void kmp(string cad, string subcad){
    int i = 0, j = 0;
    while(i < cad.size()){
        while(j >= 0 && cad[i] != subcad[j]) j = bordes[j];
        i++; j++;
        if(j == subcad.size()){
            printf("%s esta en indice %d de la cadena: %s\n",
                    subcad.c_str(), i - j, cad.c_str());
            j = bordes[j];
        }
    }
}
```

6.3. Tablas hash

```
lli prime = 131, tab[100005], pot[100005];
string v;

void hashing(){
    tab[0] = 0;  tab[1] = v[0];
    for(int i = 1; i < v.size(); i++)
        tab[i+1] = ((tab[i]*prime)) + v[i];
}//tab[v.size()] = hash_total

lli query(int p1, int p2){//pot[i] = prime^i
    p1--;
```

```
    return tab[p2] - tab[p1]*pot[p2-p1];
}
```

6.4. String alignment

Mínima distancia entre dos string.
O(s1*s2)

```
int memo[100][100];
char dummy = (char) 1;
int costo(char a, char b){return (a==b)? 0: 1;}

int dp(string &cad1, string &cad2){
    int ans;
    for(int i = 0; i <= cad2.size(); ++i) memo[0][i] = i;
    for(int i = 0; i <= cad1.size(); ++i) memo[i][0] = i;

    for(int i = 0; i < cad1.size(); ++i)//f
        for(int j = 0; j < cad2.size(); ++j){//c
            ans = costo(cad1[i],cad2[j]) + memo[i][j];
            ans = min(ans,costo(cad2[i],dummy) + memo[i+1][j]);
            ans = min(ans,costo(cad1[i],dummy) + memo[i][j+1]);
            memo[i+1][j+1] = ans;
        }
    return memo[cad1.size()][cad2.size()];
}
```

6.5. Arreglo de sufijos

```
#define maxn 100100
char cad[maxn];
int c[maxn];
int len_cad, len_subcad;//len t, len p
int ra[maxn], temp_ra[maxn], sa[maxn], temp_sa[maxn];

void countingSort(int k){
    int aux, may = max(300,len_cad);
    memset(c,0,sizeof(c));
    for(int i = 0; i < len_cad; ++i)
        c[(i+k<len_cad)? ra[i+k] : 0]++;
    for(int i = 0, sum = 0; i < may; ++i){
        aux = c[i];
        c[i] = sum;
        sum += aux;
    }
    for(int i = 0; i < len_cad; ++i)
        temp_sa[c[(sa[i]+k<len_cad)?
            ra[sa[i]+k] : 0]++] = sa[i];
    for(int i = 0; i < len_cad; ++i) sa[i] = temp_sa[i];
}

void construir_SA(){//hasta 10^5 caracteres
    strcat(cad,"$");
    len_cad = strlen(cad);

    int r;
    for(int i = 0; i < len_cad; ++i){
        ra[i] = cad[i];
        sa[i] = i;
    }
    for(int k = 1; k < len_cad; k<<=1){
        countingSort(k);
        countingSort(0);
        temp_ra[sa[0]] = r = 0;
        for(int i = 1; i < len_cad; ++i)
            temp_ra[sa[i]]=(ra[sa[i]]==ra[sa[i-1]] && ra[sa[i]+k
                ]==ra[sa[i-1]+k])?
                r: ++r;
        for(int i = 0; i < len_cad; ++i) ra[i] = temp_ra[i];
        if(ra[sa[len_cad-1]]==len_cad-1) break;
    }
}
```

6.6. Longest common prefix

Requiere implementar arreglo de sufijos.

```
int phi[maxn], plcp[maxn], lcp[maxn];
void LCP(){
    phi[sa[0]] = -1;
    for(int i = 1; i < len_cad; ++i) phi[sa[i]] = sa[i-1];
    for(int i = 0, l = 0; i < len_cad; ++i){
        if(phi[i]==-1){
            plcp[i] = 0;
            continue;
```

```
        }
        while(cad[i+l]==cad[phi[i]+l] && cad[i+l]!='$') l++;
        plcp[i] = l;
        l = max(l-1,0);
    }
    for(int i = 0; i < len_cad; ++i)
        lcp[i] = plcp[sa[i]];
}//lcp[i] = maximo prefijo en sa[i].
```

6.7. String matching

Requiere implementar arreglo de sufijos con la cadena principal.

```
ii stringMatching() {
    len_subcad = strlen(subcad);
    int lo = 0, hi = len_cad-1, mid = lo;
    while (lo < hi) {
        mid = (lo + hi) / 2;
        int res = strncmp(cad + sa[mid], subcad, len_subcad);
        if (res >= 0) hi = mid;
        else lo = mid + 1;
    }
    if (strncmp(cad + sa[lo], subcad, len_subcad) != 0)
        return ii(-1, -1);
    ii ans; ans.first = lo;
    lo = 0;  hi = len_cad - 1;  mid = lo;
    while (lo < hi) {
        mid = (lo + hi) / 2;
        int res = strncmp(cad + sa[mid], subcad, len_subcad);
        if (res > 0) hi = mid;
        else lo = mid + 1;
    }
    if (strncmp(cad + sa[hi], subcad, len_subcad) != 0) hi--;
    ans.second = hi;
    return ans;
}
```

6.8. Subcadena común mas larga

Requiere implementar arreglo de sufijos y Longest common prefix.

```
char subcad[maxn];
vector<string> LCS(){
    int l = strlen(cad), best = -1;
    strcat(cad,"."); strcat(cad,subcad); strcat(cad,"$");
    construir_SA();  LCP();
    string aux;  set<string> ss;  vector<string> res;

    for(int i = 0; i < len_cad; ++i){
        if(lcp[i] < best) continue;
        if((sa[i]>l && sa[i-1]>l) || (sa[i]<l&&sa[i-1]<l) || lcp[i
            ]<1)
            continue;

        aux = "";
        for(int j=0; j<lcp[i]; ++j)
            aux.push_back(cad[sa[i]+j]);
        if(lcp[i]>best){
            best = lcp[i];
            ss.clear();
            ss.insert(aux);
        }else if(lcp[i]==best) ss.insert(aux);
    }
    res.assign(ss.begin(),ss.end());
    return res;
}
```

7. Geometria

7.1. Punto

```
struct punto{
    double x, y;

    punto(){ x = y = 0; }
    punto(double _x, double _y){
        x = _x;  y = _y;
    }

    bool operator < (punto p) const{//para poder usar sort
        if(fabs(x - p.x) > eps) return x < p.x;
        return y < p.y;
```

```
    }
    bool operator == (punto p) const{
        return fabs(x - p.x) < eps && fabs(y - p.y) < eps;
    }
};

vec toVec(punto a, punto b){return vec(b.x-a.x, b.y-a.y);}
double DEG_TO_RAD(double n){ return n*3.1416/180.0; }

punto rotar(punto p, double grados){//en sentido antihorario
    double rad = DEG_TO_RAD(grados);
    return punto(p.x*cos(rad) - p.y*sin(rad),
                 p.x*sin(rad) + p.y*cos(rad));
}
punto transladar(punto p, vec v){
    return punto(p.x+v.x, p.y+v.y);
}
double dist(punto p1, punto p2){
    return hypot(p1.x - p2.x, p1.y - p2.y);
}
double angulo(punto a, punto o, punto b){//en radianes
    vec oa = toVec(o, a), ob = toVec(o, b);
    return acos(dot(oa, ob)/sqrt(norm_sq(oa)*norm_sq(ob)));
}
bool colineal(punto p, punto q, punto r){
    return fabs(cross(toVec(p,q),toVec(p,r))) < eps;
}//r esta en la misma linea que p-q
```

7.2. Linea y segmento

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

```
struct linea{
    double a, b, c;
    punto p1, p2;

    linea(double _a, double _b, double _c){
        a = _a;  b = _b;  c = _c;
    }
    linea(punto _p1, punto _p2){
        p1 = punto(_p1.x, _p1.y);
        p2 = punto(_p2.x, _p2.y);
        if(fabs(p1.x - p2.x) < eps){
            a = 1.0;  b = 0.0;  c = -p1.x;
        }else{
            a = -((p1.y-p2.y) / (p1.x-p2.x));
            b = 1.0;
            c = -a*p1.x-p1.y;
        }
    }
};

bool paralelas(linea l1, linea l2){
    return fabs(l1.a-l2.a)<eps && fabs(l1.b-l2.b)<eps;
}
bool iguales(linea l1, linea l2){
    return paralelas(l1, l2) && fabs(l1.c-l2.c)<eps;
}
bool interseccion(linea l1, linea l2, punto &p){
    if(paralelas(l1, l2)) return false;
    p.x = (l2.b*l1.c-l1.b*l2.c) / (l2.a*l1.b-l1.a*l2.b);
    if(fabs(l1.b)>eps) p.y = -(l1.a*p.x + l1.c);
    else p.y = -(l2.a*p.x + l2.c);
    return true;
}
bool intersecSegmentos(linea l1, linea l2, punto &p){
    punto pp, c;
    if(interseccion(l1,l2,pp)){
        if(distSegmento(pp,l1,c)<eps &&
           distSegmento(pp,l2,c)<eps){
            p.x = pp.x;  p.y = pp.y;
            return true;
        }
    }
    return false;
}
//distancia minima entre p y l
double distLinea(punto p, linea l, punto &c){
    punto a = l.p1, b = l.p2;
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    c = transladar(a, escalar(ab, u));//punto mas cercano
    return dist(p, c);
}
```

```
double distSegmento(punto p, linea l, punto &c){
    punto a = l.p1, b = l.p2;
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    if(u < 0.0){
        c = punto(a.x, a.y); return dist(p, a);
    }
    if(u > 1.0){
        c = punto(b.x, b.y); return dist(p, b);
    }
    return distLinea(p, l, c);
}
```

7.3. Vector

```
struct vec{
    double x, y;
    vec(double _x, double _y){
        x = _x;  y = _y;
    }
    vec operator - (const vec& q) const{
        return vec(x-q.x, y-q.y);
    }
};

vec toVec(punto a, punto b){
    return vec(b.x-a.x, b.y-a.y);
}
vec escalar(vec v, double s){
    return vec(v.x*s, v.y*s);
}
double dot(vec a, vec b){
    return a.x*b.x + a.y*b.y;
}
double norm_sq(vec v){
    return v.x*v.x + v.y*v.y;
}
double cross(vec a, vec b){
    return a.x*b.y - a.y*b.x;
}
bool ccw(punto p, punto q, punto r){
    return cross(toVec(p,q), toVec(p,r)) > 0;
}
```

7.4. Convex Hull

Polígono convexo con perímetro mínimo que cubre todos los puntos, verificar que no hallan puntos repetidos, O(n log n)

```
punto pivote;

bool angleCmp(punto a, punto b){
    if(colineal(pivote,a,b))
        return dist(pivote,a) < dist(pivote,b);
    double d1x = a.x-pivote.x, d1y = a.y-pivote.y;
    double d2x = b.x-pivote.x, d2y = b.y-pivote.y;
    return (atan2(d1y,d1x) - atan2(d2y,d2x)) < 0;
}

vector<punto> ConvexHull(vector<punto> p){
    pivote = punto(0,0);
    int i, j, n = p.size(), k = 0;
    if(n <= 3){
        if(!(p[0]==p[n-1])) p.push_back(p[0]);
        return p;
    }

    sort(p.begin(), p.end());
    vector<punto> s(p.size()*2);
    for(int i = 0; i < p.size(); i++){
        while(k>=2 && !ccw(s[k-2],s[k-1],p[i])) k--;
        s[k++] = p[i];
    }

    for(int i=p.size()-2, t=k+1; i>=0; i--){
        while(k>=t && !ccw(s[k-2],s[k-1],p[i])) k--;
        s[k++] = p[i];
    }
    s.resize(k);
    return s;
}
```

7.5. Área de un polígono

Área de un polígono, resultado con coordenadas en sentido horario: positivo, anti-horario: negativo, y cero si están desordenados, O(n)

```
double AreaPoligono(vector<punto> &p){
    double s = 0.0, d = 0.0;
    int n = p.size();
    forr(i, n){
        s += p[i].x*p[(i+1)%n].y;
        d += p[(i+1)%n].x*p[i].y;
    }
    return fabs(0.5*(s-d));
}
```

7.6. Punto en poligono

Verifica si un punto esta dentro de un polígono, O(n)

```
//Metodo general
bool enPoligono(punto pt, vector<punto> &p){
    if(p.size() == 0) return false;
    double sum = 0.0;
    for(int i = 1; i < p.size(); i++){
        if(ccw(pt, p[i-1], p[i]))
            sum+= angulo(p[i-1],pt,p[i]);
        else sum -= angulo(p[i-1],pt,p[i]);
    }
    return fabs(fabs(sum) - 2*PI) < eps;
}
//Metodo mas preciso para poligonos convexos
double areaTriangulo(punto &pa, punto &pb, punto &pc){
    vec a(pa.x,pa.y), b(pb.x,pb.y), c(pc.x,pc.y);
    return (cross((c-a),(b-a)))/2.0;
}
//considerar el caso de todos los puntos iguales o colineales
bool enPoligonoConvexo(punto pt, vector<punto> &p) {
    double area = AreaPoligono(p), sum = 0.0;
    for(int i = 0; i < p.size()-1; i++) {
        if(p[i] == pt) return true;
        sum += fabs(areaTriangulo(p[i],pt,p[i+1]));
    }
    return fabs(area - sum) < eps;
}
```

7.7. Minimum covering circle

Mínimo circulo que puede cubrir todos los puntos

```
double distpow2(punto p1, punto p2){
    return (p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y);
}

double CircumCircle(punto &A, punto &B, punto &C, punto &c){
    double bx = B.x-A.x, by = B.y-A.y;
    double cx = C.x-A.x, cy = C.y-A.y;
    double _B = bx * bx + by * by;
    double _C = cx * cx + cy * cy;
    double _D = bx * cy - by * cx;
    c = punto((cy*_B - by*_C) / (2*_D),
              (bx*_C - cx*_B) / (2*_D));
    c.x += A.x;
    c.y += A.y;
    return distpow2(c, A);
}

double minCoverCircle(vector<punto> &p){
    double maxlen = 0.0, aux;
    punto c = p[0];
    double r = 0.0;

    forab(i,1,p.size()){
        if(distpow2(c, p[i]) >= r+eps){
            r = 0.0;
            c = p[i];
            forr(j, i){
                if(distpow2(c, p[j]) >= r+eps){
                    c.x = (p[i].x+p[j].x)/2.0;
                    c.y = (p[i].y+p[j].y)/2.0;
                    r = distpow2(c, p[i]);
                    forr(k, j){
                        if(distpow2(p[k], c) >= r+eps)
                            r = CircumCircle(p[i],p[j],p[k],c);
                    }
                }
```

```
            }
        }
    }
    return sqrt(r);
}
```

7.8. Nearest pair of points

Mínima distancia entre dos puntos O(n)

```
bool cmp_x(const punto & a, const punto & b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool cmp_y(const punto & a, const punto & b) {
    return a.y < b.y;
}

vector<punto> a, t;
double mindist;
pair<int, int> best_pair;

void upd_ans(const punto & a, const punto & b) {
    double dist = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
    if (dist < mindist) {
        mindist = dist;
        best_pair = {a.id, b.id};
    }
}

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

    int m = (l + r) >> 1, midx = a[m].x;
    rec(l, m);  rec(m, r);

    merge(a.begin() + l, a.begin() + m, a.begin() + m, a.begin() + r, t.begin(), cmp_y);
    copy(t.begin(), t.begin() + r - l, a.begin() + l);
    int tsz = 0;
    for (int i = l; i < r; ++i) {
        if (abs(a[i].x - midx) < mindist) {
            for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y < mindist; --j)
                upd_ans(a[i], t[j]);
            t[tsz++] = a[i];
        }
    }
}

double solve(){
    sort(a.begin(), a.end(), cmp_x);
    t.resize(a.size());
    mindist = 1E20;
    rec(0, a.size());
    return mindist;
}
```

8. Formulas

8.1. Tabla ASCII

Caracteres ASCII con sus respectivos valores numéricos.

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

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

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

8.2. Formulas generales

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.	$\binom{m+n-1}{n} = \frac{(m+n-1)!}{n!(m-1)!}$
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 primero se repite a veces, el segundo b veces, etc.	$PR_n^{a,b,c\ldots} = \frac{P_n}{a!b!c!\ldots}$
Permutaciones sin repetición: Número de formas de agrupar r elementos de n disponibles, sin repetir elementos	$\frac{n!}{(n-r)!}$

DISTANCIAS	
Distancia Euclideana	$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
Distancia Manhattan	$d_M(P_1, P_2) = \lvert x_2 - x_1\rvert + \lvert y_2 - y_1\rvert$

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	$A = \pi(R^2 - r^2)$	Formula general	$(X - P_x)^2 + (Y - P_y)^2 = r^2$

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 círcunferencias asociadas.			
Área con base y altura	$A = \frac{1}{2} b * h$	Área con 2 lados y su ángulo	$A = \frac{1}{2} b*a*sin(C)$
Área con los 3 lados	$A = \sqrt{p(p-a)(p-b)(p-c)}$ con $p = \frac{a+b+c}{2}$		
Triángulo circunscrito a circunferencia	$A = \frac{abc}{4r}$	Triángulo inscrito a circunferencia	$A = r(\frac{a+b+c}{2})$
Triangulo equilátero	$A = \frac{\sqrt{3}}{4} a^2$		

TRIGONOMÉTRIA		
$sin(\alpha) = \frac{opuesto}{hipotenusa}$	$cos(\alpha) = \frac{adyacente}{hipotenusa}$	$tan(\alpha) = \frac{opuesto}{adyacente}$
$sec(\alpha) = \frac{1}{cos(\alpha)}$	$csc(\alpha) = \frac{1}{sin(\alpha)}$	$cot(\alpha) = \frac{1}{tan(\alpha)}$
Ley de los senos, con γ el angulo opuesto al lado c	$\frac{a}{sin(\alpha)} = \frac{b}{sin(\beta)} = \frac{c}{sin(\gamma)}$	
Ley de los cosenos, con γ el angulo opuesto al lado c	$c^2 = a^2 + b^2 - 2ab * cos(\gamma)$	

PROPIEDADES DEL MÓDULO (RESIDUO)	
Neutra	(a % b) % b = a % b
Asociativa en suma	(a + b) % c = ((a % c) + (b % c)) % c
Asociativa en multiplicación	(a*b) % c = ((a % c)*(b % c)) % c

CONSTANTES	
Pi	$\pi = acos(-1) \approx 3{,}14159$
e	$e \approx 2{,}71828$
Número áureo	$\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}61803$

FIGURAS			
Elipse	$A = PI * a * b$	Cono	$V = \frac{1}{3} * PI * r^2 * h$
Cilindro	$V = PI * r^2 * h$	Esfera	$V = \frac{4}{3} * PI * r^3$

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 $\leq n$ coprimos con n.
Números de Bell	1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, ...

Continúa en la siguiente columna

	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 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 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 $p(0) = 2$.

Continúa en la siguiente columna

Números tetraedrales	0, 1, 4, 10, 20, 35, 56, 84, 120, 165, 220, 286, 364, ...
	$f(n) = \frac{n*(n+1)*(n+2)}{6}$
Números triangulares	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \frac{n(n+1)}{2}$
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 $n = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$ se tiene que: $f(n) = \frac{p_1^{a_1+1} - 1}{p_1 - 1} * \frac{p_2^{a_2+1} - 1}{p_2 - 1} * ... * \frac{p_k^{a_k+1} - 1}{p_k - 1}$
Schroeder numbers	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 algoritmicas. Tomar esta tabla solo como referencia.

Complexity	**n**
$O(n!)$	11
$O(2^n * n^2)$	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.

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

typedef pair<int, int> ii;
typedef pair<int, ii> iii;

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ii> vii;
typedef vector<vii> vvii;

typedef unsigned long long int ulli;
typedef long long int lli;

```

```
#define mpiii(a, b, c) iii(a, ii(b, c))
#define inf 1000000000//10^9
#define INFmemset 5436//inf para memeset en enteros
#define INFmemsetLL 3586//inf para memset en lli
#define forr(i, n) for(int i = 0; i < n; ++i)
#define forab(i, a, b) for(int i = a; i < b; ++i)

double eps = 1e-5;//ajustar segun se necesite

int main(){//fast I/O con iostream
    cin.tie(NULL);
    ios_base::sync_with_stdio(false);
    cout << "hola mundo" << '\n';
    return 0;
}
```

9.2. Ayudas

```
/* Expandir pila de memoria C++ 11
#include <sys/resource.h>
rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur=1024L*1024L*256L;//256mb
setrlimit(RLIMIT_STACK, &rl);
*/

//iterar mascara de bits
for(int i=n; i; i=(i-1)&n) // Decreciente
for (int i=0; i=i-n&n; ) // creciente
x = __builtin_popcount(n);//bits encendidos en n
x = __builtin_ctz(n);//ceros a la derecha de n
x = __builtin_clz(n);//ceros a la izquierda de n
x = __builtin_ffs(n);//primera posicion en 1
x = __builtin_ctzll((lli) n);//para lli agregars ll al nombre
x = (n&(-n));//least significant bit en 1

//Expresiones Regulares
string expresion = "(take|love|know|like)s*";
string cadena = "likes knows";
regex reg(expresion);
bool match = regex_match(cadena,reg);//Existe?

smatch matches;
while(regex_search(cadena, matches, reg)){
    cadena = matches.suffix();//Recorrer matches
    cout << cadena << endl;
}
```

9.3. Formulas extra

Formulas extras	
Formula de números fibonacci	$f(n) = \frac{1}{\sqrt{5}} * [(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n]$
Ecuación de la recta que pasa por dos puntos	y = mx + b. $\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$
Ecuación del plano que pasa por 3 puntos	Al resolver la determinante, se tiene el plano que pasa por 3 puntos de la forma (x,y,z). $\begin{vmatrix} X-x_1 & Y-y_1 & Z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{vmatrix} = 0$
Distancia de un punto a una recta	Teniendo una recta con formula de la forma: $ax+by+c$ la distancia mínima 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 triángulos degenerados	Si el resultado es mayor a 0.5 es un triángulo de calidad buena. Es posible formar un triangulo si $a+b>c$ con $c>b>a$. $\frac{(a+b-c)*(a+c-b)*(b+c-a)}{a*b*c}$

Continúa en la siguiente columna

Formula de fibonacci con matrices	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^b = \begin{bmatrix} fib(b+1) & fib(b) \\ fib(b) & fib(b-1) \end{bmatrix}$
Progresión aritmética	Sea d la diferencia y a_1 el numero inicial, entonces $a_n = a_1 + (n-1)d$. y la sumatoria de los primeros n elementos es: $\sum_{i=1}^{n} a_i = n\frac{a_1+a_n}{2}$
Progresión geométrica	Sea r la razón y a_1 el numero inicial, entonces $a_n = a_1 * r^{n-1}$ y la sumatoria de los primeros n elementos es: $\sum_{i=1}^{n} a_i = a_1 * \frac{r^n-1}{r-1}$
Teorema de Erdős–Gallai	Una secuencia de enteros $d_1 \geq ... \geq d_n$ puede representar una secuencia de grados de un grafo si y solo si: para cada k en $1 \leq k \leq n$ $\sum_{i=1}^{k} d_i \leq k(k+1) + \sum_{i=k+1}^{n} min(d_i, k)$
Cantidad de divisores de un numero	con n = $p_1^{a_1} * p_2^{a_2} * ... * p_k^{a_k}$ la cantidad sera: $\prod_{i=1}^{k} a_i + 1$
Coeficientes binomiales	Encuentra n combinado k. para construir el triangulo de pascal solo poner en n la fila y en k la columna. $C(n,k) = \begin{cases} 0 & \text{k=0,n=k} \\ C(n-1,k-1) + C(n-1,k) & \text{c.c.} \end{cases}$
Números de catalán	Encontrar numero de arboles binarios de n nodos, numero de formas de emparejar paréntesis. $Cat(n) = \begin{cases} 1 & \text{si } n = 0 \\ \frac{2n*(2n-1)*Cat(n-1)}{(n+1)*n} & \text{c.c.} \end{cases}$

Continúa en la siguiente columna

Teorema de Pick	Sea A el área de un polígono con puntos enteros, B la cantidad de puntos enteros en el borde, I la cantidad de puntos enteros interiores, entonces: $A = I + \frac{B}{2} - 1$

9.4. 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 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 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

Primos cercanos a potencias de 10:

7 11, 89 97 101 103, 983 991 997 1009 1013 1019, 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

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 70368744177664 140737488355328 281474976710656 562949953421312 1125899906842624 2251799813685248 4503599627370496 9007199254740992 18014398509481984 36028797018963968 72057594037927936 144115188075855872 288230376151711744 576460752303423488 1152921504606846976 2305843009213693952 4611686018427387904 9223372036854775808