O(n!)	10..11
$O(2^n n^2)$	15..18
$O(2^n n)$	18..22
$O(n^4)$	100
$O(n^3)$	400
$O(n^2 \log^2 n)$	1000
$O(n^2 \log n)$	2000
$O(n^2)$	10000
$O(n \log^2 n)$	$3*10^5$
$O(n \log n)$	10^6
O(n)	10^8

Union Find: Página 54 CP3

BFS

```
//vector<int> dist(n,1e9);
void bfs(graph &g, int start, vector<int> &dist) {
    int inf = 1e9;
    queue<int> q;
    q.push(start);
    dist[start] = 0;
    while (not q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : g[u]) if (dist[v] == inf) {
            dist[v] = dist[u] + 1;
            q.push(v);
        }
    }
}
```

Binary search (int)

```
int l = (algo), r = (algo);
while (l != r) {
    int m = (l+r)/2;
//overflow?
    if (predicado(m))
        r = m;
    else
        l = m+1;
}
int ans = r; //nunca m !!!
```

(double)

```
double l = (algo), r =
(algo);
rep(i, 50) {
    double m = (l+r)/2;
    if (predicado(m))
        r = m;
    else
        l = m; //no sumar 1
}
double ans = r; //nunca m !!!
```

Declaraciones típicas

```
#include <bits/stdc++.h>
typedef long long intt;
typedef pair<int,int> par;
typedef vector<vector<int>>
    graph;
typedef vector<vector<par>>
    wgraph;

ios::sync_with_stdio(0);
cin.tie(0);

cout.setf(ios::fixed);
cout.precision(4); //123.0000
```

Dijkstra //graph es vector<par> con pares (vecino, peso)

```
//vector<int> dist(n,1e9);
void dijkstra(wgraph &g, int start, vector<int> &dist) {
    priority_queue<par, vector<par>, greater<par>> q;
    q.push({0, start});
    dist[start] = 0;
    while (not q.empty()) {
        int priority = q.top().first, u = q.top().second;
        q.pop();
        if (priority != dist[u])
            continue;
        for (par p : g[u]) {
            int v = p.first, w = p.second;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                q.push({dist[v], v});
            }
        }
    }
}
```

BFS/DFS	$O(V+E) = O(V^2)$
Dijkstra	O((V+E)*log V) ??
Hopcroft	O(E sqrt V)

pb	push_back
invrep(i,n)	for(int i=n-1; i>=0;--i)

```
ceildiv: return (a+b-1)/b;
gcd:  __gcd(a,b) //dos undersc
lcm:     return a*(b/gcd(a,b));
```

phi(p) = p-1 //p es primo
phi(p^k) = (p-1)*p^{k-1}
    //p es primo, k natural
phi(m*n) = phi(m)*phi(n)

Encontrar Componentes Conexas

```
//función auxiliar requerida
void dfs_ccs(graph& g, int u, vector<bool>& vis, vector<int>& cc)
{
    vis[u] = true;
    cc.pb(u);
    for (int v : g[u]) if (not vis[v])
        dfs_ccs(g,v,vis,cc);
}

//ccs debe partir vacío y terminará con las componentes conexas
void encontrar_ccs(graph& g, vector<vector<int>>& ccs) {
    int n = g.size();
    vector<bool> vis(n,false);
    rep(u,n) if (not vis[u]) {
        ccs.pb(vector<int>());
        dfs_ccs(g,u,vis,ccs.back());
    }
}
```

Bipartite check

```
//vector<int> color(n,0);
//llamar con cualquier nodo u y dejar c en su default
//retorna si *la componente conexa* es bipartita
bool es_bipartita(graph& g, int u, vector<int>& color, int c = 1) {
        color[u] = c;
        for (int v : g[u]) {
                if (color[v] == c)
                        return false;
                if (color[v] == 0 and not es_bipartita(g,v,color,3-c))
                        return false;
        }
        return true;
}
```

Hopcroft

```
class Hopcroft {
        graph g;
        vector<int> U,dist;
        int inf = 1e9;

        bool bfs() {
        queue<int> q;
        for (int u : U) if (match[u] == nil) {
                dist[u] = 0;
                q.push(u);
        } else dist[u] = inf;
        dist[nil] = inf;
        while (not q.empty()) {
                int u = q.front(); q.pop();
                if (u != nil) for (int v : g[u]) if (dist[match[v]] == inf) {
                        dist[match[v]] = dist[u]+1;
                        q.push(match[v]);
                }
        }
        return (dist[nil] != inf);
        }

        bool dfs(int u) {
        if(u != nil) {
                for (int v : g[u]) if (dist[match[v]] == dist[u]+1 and dfs(match[v])) {
                        match[v] = u;
                        match[u] = v;
                        return true;
                }
                dist[u] = inf;
                return false;
        }
        return true;
        }

public:
        vector<int> match;
        int nil, isPerfect, matchSize = 0;
        Hopcroft(graph& gg, vector<int>& UU) {
                g = gg; U = UU; nil = g.size();
                match.assign(g.size()+1,nil);
                dist.assign(g.size()+1,inf);
                while (bfs()) for (int u : U) if (match[u] == nil and dfs(u))
                        ++matchSize;
                isPerfect = (matchSize == U.size() and g.size() == U.size()*2);

        }
};
```

TopoSort

```
//función auxiliar requerida
void dfs_ts(graph& g, int u, vector<bool>& vis, vector<int>& ts) {
    vis[u] = true;
    for (int v : g[u]) if (not vis[v])
        dfs_ts(g, v, vis, ts);
    ts.pb(u);
}

//ts parte vacío y termina con el toposort
void toposort(graph& g, vector<int>& ts) {
    int n = g.size();
    vector<bool> vis(n, false);
    rep(u, n) if (not vis[u])
        dfs_ts(g, u, vis, ts);
    reverse(ts.begin(), ts.end());
}
```

Floyd Warshall (copia CP3)

```
//dist[i][j] contiene inicialmente el peso de la arista (i, j) o 1e9 si no existe esa arista
//el loop es k -> i -> j !!
rep(k, nGrafo) rep(i, nGrafo) rep(j, nGrafo)
    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

```
//variante: transitive closure
connected[i][j] es true ssi existe la arista (i, j)
cambiar última línea por: connected[i][j] |= (connected[i][k] & connected[k][j]);
```

Bellman Ford's (copia CP3)

```
vector<int> dist(nGrafo, 1e9); dist[start] = 0;
rep(i, nGrafo-1) rep(u, nGrafo) for (par p : g[u]) {
    int v = p.first, w = p.second;
    dist[v] = min(dist[v], dist[u] + w);
}
```

```
//después de ejecutar lo anterior
bool hayCicloNegativo = false;
rep(u, nGrafo) for (par p: g[u]) {
    int v = p.first, w = p.second;
    if (dist[v] > dist[u] + w)
        hayCicloNegativo = true;
}
```

Debugging (sacar comentarios para activar)

```
#define debugx(x) //cerr<<#x<<": "<<x<<endl

#define debugv(v) //cerr<<#v<<":";rep(i,(int)v.size())cerr<<" "<<v[i];cerr<<endl

#define debugm(m) //cerr<<#m<<endl;rep(i,(int)m.size()){cerr<<i<<":";rep(j,(int)m[i].size())\
cerr<<" "<<m[i][j];cerr<<endl;}

#define debugmp(m) //cerr<<#m<<endl;rep(i,(int)m.size()){cerr<<i<<":";rep(j,(int)m[i].size())\
cerr<<" {"<<m[i][j].first<<","<<m[i][j].second<<"}";cerr<<endl;}

//secciones en negrita son la única diferencia entre estas dos últimas
```

Algoritmo Húngaro (parte 1 de 3)

```
#define MAX_V 500

int V;
int cost[MAX_V][MAX_V];
int lx[MAX_V],ly[MAX_V];
int max_match,xy[MAX_V],yx[MAX_V],prev[MAX_V];
bool S[MAX_V],T[MAX_V];
int slack[MAX_V],slackx[MAX_V];

void init_labels(){
        memset(lx,0,sizeof(lx));
        memset(ly,0,sizeof(ly));

        for(int x = 0;x<V;++x)
                for(int y = 0;y<V;++y)
                        lx[x] = max(lx[x],cost[x][y]);
}

void update_labels(){
        int x,y,delta = INT_MAX;

        for(y = 0;y<V;++y)
                if(!T[y]) delta = min(delta,slack[y]);
        for(x = 0;x<V;++x)
                if(S[x]) lx[x] -= delta;
        for(y = 0;y<V;++y)
                if(T[y]) ly[y] += delta;
        for(y = 0;y<V;++y)
                if(!T[y]) slack[y] -= delta;
}

void add_to_tree(int x, int prevx){
        S[x] = true;
        prev[x] = prevx;

        for(int y = 0;y<V;++y){
                if(lx[x]+ly[y]-cost[x][y]<slack[y]){
                        slack[y] = lx[x]+ly[y]-cost[x][y];
                        slackx[y] = x;
                }
        }
}

int q[MAX_V],head,tail;
```

Algoritmo Húngaro (parte 2 de 3)

```
void augment(){
        int x,y,root;
        head = tail = 0;
        memset(S,false,sizeof(S));
        memset(T,false,sizeof(T));
        memset(prev,-1,sizeof(prev));

        for(x = 0;x<V;++x) if(xy[x]==-1){
                q[tail++] = root = x;
                prev[root] = -2;
                S[root] = true;
                break;
        }

        for(y = 0;y<V;++y){
                slack[y] = lx[root]+ly[y]-cost[root][y];
                slackx[y] = root;
        }

        while(true){
                while(head<tail){
                        x = q[head++];

                        for(y = 0;y<V;++y) if(cost[x][y]==lx[x]+ly[y] && !T[y]){
                                if(yx[y]==-1) break;

                                T[y] = true;
                                q[tail++] = yx[y];
                                add_to_tree(yx[y],x);
                        }

                        if(y<V) break;
                }

                if(y<V) break;

                update_labels();

                head = tail = 0;

                for(y = 0;y<V;++y) if(!T[y] && slack[y]==0){
                        if(yx[y]==-1){
                                x = slackx[y];
                                break;
                        }

                        T[y] = true;

                        if(!S[yx[y]]){
                                q[tail++] = yx[y];
                                add_to_tree(yx[y],slackx[y]);
                        }
                }

                if(y<V) break;
        }

        ++max_match;

        for(int cx = x,cy = y,ty;cx!=-2;cx = prev[cx],cy = ty){
                ty = xy[cx];
                yx[cy] = cx;
                xy[cx] = cy;
        }
}
```

Algoritmo Húngaro (parte 3 de 3)

```
int hungarian(){
        int ret = 0;
        max_match = 0;

        memset(xy,-1,sizeof(xy));
        memset(yx,-1,sizeof(yx));

        init_labels();
        for(int i = 0;i<V;++i) augment();

        for(int x = 0;x<V;++x)
                if(cost[x][xy[x]]>0) ret += cost[x][xy[x]];

        return ret;
}

int main(){
    while(true){
        int M,N; cin >> M >> N;if(M == 0 and N == 0)break;
        V = max(M,N);
        for(int i = 0;i<V;++i)
            for(int j = 0;j<V;++j)
                cost[i][j] = -10001;

        // Si es minimizar costo :
        /*int dumcost[M][N];
        int maxim = -1;
        for(int i =0;i<M;++i)
            for(int j=0;j<N;++j)
                {
                    cin >> dumcost[i][j];
                    if(dumcost[i][j] > maxim)
                    {
                        maxim = dumcost[i][j];
                    }
                }
        for(int i =0;i<M;++i)
            for(int j=0;j<N;++j)
                cost[i][j] = maxim - dumcost[i][j];
        int ans = hungarian();
        int realans = 0;
        for(int i =0;i<M;++i)
                 realans += dumcost[i][xy[i]];
        cout << realans << endl;
        */

        // Si es maximizar
        for(int i = 0; i < M ; ++i)
            for(int j = 0; j < N; ++j)
                cin >> cost[i][j];
        int ans = hungarian();
        cout << ans << endl;
    }
    return 0;
}
```

SegTree (no tan) simple

```cpp
template<class t> class SegTree {
    vector<intt> arr, st, leaf; int n;

    void build(int u, int i, int j) {
         if (i == j) {
                 leaf[i] = u;
                 st[u] = t::base(arr, i);
                 return;
         }
         int m = (i+j)/2, l = u*2+1, r = u*2+2;
         build(l, i, m);
         build(r, m+1, j);
         st[u] = t::op(st[l], st[r]);
    }

    intt query(int a, int b, int u, int i, int j) {
         if (j < a or b < i)
                 return t::neutro;
         if (a <= i and j <= b)
                 return st[u];
         int m = (i+j)/2, l = u*2+1, r = u*2+2;
         intt x = query(a, b, l, i, m);
         intt y = query(a, b, r, m+1, j);
         return t::op(x, y);
    }

    void update(int u) {
         int l = u*2+1, r = u*2+2;
         st[u] = t::op(st[l], st[r]);
         if (u) update((u-1)/2);
    }

public:
    SegTree(vector<intt>& v) {
         arr = v;
         n = v.size();
         leaf.resize(n);
         st.resize(n*4+5);
         build(0, 0, n-1);
    }

    intt query(int a, int b) {
         return query(a, b, 0, 0, n-1);
    }

    void update(int index, intt value) {
         int u = leaf[index];
         arr[index] = value;
         st[u] = t::base(arr, index);
         if (u) update((u-1)/2);
    }

    intt get(int i) {return arr[i];}
};
```

Range Sum Query

```cpp
struct RSQ {
    static intt const neutro = 0;
    static intt op(intt x, intt y) {
        return x + y;
    }
    static intt base
      (vector<intt>& arr, int i) {
        return arr[i];
    }
};
```

Range Minimum Query
(modificable para retornar índice)

```cpp
struct RMinQ {
    static intt const neutro = 1e18;
    static intt op(intt x, intt y) {
        return min(x, y);
    }
    static intt base
      (vector<intt>& arr, int i) {
        return arr[i];
};
```

Range Maximum Query

```cpp
struct RMaxQ { neutro = -1e18;
    ...return max(x, y);... };
```

neutro: Query en un rango de largo 0.

op(x, y): Query para dos elementos x, y.

base(arr, i): Query para singleton arr[i].

Inicialización: SegTree<RSQ> st(arr);
* Cambiar RSQ por el tipo de query deseada.

IMPORTANTE
Los rangos se asumen válidos e indexados desde 0.
Además, se debe usar el getter de la clase para obtener valores individuales en el arreglo.

SegTree Lazy Template (podría haber sido SegTree Template Lazy, pero eso era plagio de la STL)

```
template<class t> class SegTreeLazy {
    vector<intt> arr, st, lazy; int n;

    void build(int u, int i, int j) {
        if (i == j) {
            st[u] = arr[i];
            return;
        }
        int m = (i+j)/2, l = u*2+1, r = u*2+2;
        build(l, i, m);
        build(r, m+1, j);
        st[u] = t::op(st[l], st[r]);
    }

    void propagate(int u, int i, int j, intt x) {
        st[u] += t::lazy_op(i, j, x);
        if (i != j) {
            lazy[u*2+1] += x;
            lazy[u*2+2] += x;
        }
        lazy[u] = 0;
    }

    intt query(int a, int b, int u, int i, int j) {
        if (j < a or b < i)
            return t::neutro;
        int m = (i+j)/2, l = u*2+1, r = u*2+2;
        if (lazy[u])
            propagate(u, i, j, lazy[u]);
        if (a <= i and j <= b)
            return st[u];
        intt x = query(a, b, l, i, m);
        intt y = query(a, b, r, m+1, j);
        return t::op(x, y);
    }

    void update(int a, int b, intt value,
    int u, int i, int j) {
        int m = (i+j)/2, l = u*2+1, r = u*2+2;
        if (lazy[u])
            propagate(u, i, j, lazy[u]);
        if (a <= i and j <= b)
            propagate(u, i, j, value);
        else if (j < a or b < i) return; else {
            update(a, b, value, l, i, m);
            update(a, b, value, r, m+1, j);
            st[u] = t::op(st[l], st[r]);
        }
    }

public:
    SegTreeLazy(vector<intt>& v) {
        arr = v;
        n = v.size();
        st.resize(n*4+5);
        lazy.assign(n*4+5, 0);
        build(0, 0, n-1);
    }

    intt query(int a, int b) {
        return query(a, b, 0, 0, n-1);
    }

    void update(int a, int b, intt value) {
        update(a, b, value, 0, 0, n-1);
    }
};
```

Range Sum Query

```
struct RSQ {
    static intt const neutro = 0;
    static intt op(intt x, intt y) {
        return x + y;
    }
    static intt
      lazy_op(int i, int j, intt x) {
        return (j - i + 1)*x;
    }
};
```

Range Minimum Query

```
struct RMinQ {
    static intt const neutro = 1e18;
    static intt op(intt x, intt y) {
        return min(x, y);
    }
    static intt
      lazy_op(int i, int j, intt x) {
        return x;
};
```

Range Maximum Query

```
struct RMaxQ {...return max(x, y);...};
```

neutro: Query en un rango de largo 0.

op(x, y): Query para dos elementos x, y.

lazy_op(i, j, x): ¿En cuánto debe aumentar la respuesta para [i, j] si la información adicional en lazy[u] corresponde a x?

En general la información adicional en lazy[u] corresponde al incremento pendiente en caso de ser constante en el rango.

Inicialización: SegTreeLazy<RSQ> st(arr);
* Cambiar RSQ por el tipo de query deseada.

IMPORTANTE Los rangos se asumen válidos e indexados desde 0

HeavyLight incompleto

```
class HeavyLight {
    vector<int> treeSize, spChild, hpStart;
    wgraph g; LcaGetter lg; //?? ??

    void build(wgraph& gg, int u) {
        treeSize[u] = 1;
        for (par p : gg[u]) {
            int v = p.first, w = p.second;
            if (treeSize[v])
                continue;
            g[u].pb(p);
            build(gg, v);
            treeSize[u] += treeSize[v];
            if (spChild[u] == -1 or treeSize[v] > treeSize[spChild[u]])
                spChild[u] = v;
        }
    }

    void hpbuild(int u) {
        for (int p : g[u]) {
            int v = p.first;
            if (v == spChild[u])
                hpStart[v] = hpStart[u];
            hpbuild(v);
        }
    }

    void query(int u) {
        ...
    }

public:
    HeavyLight(wgraph& gg) {
        int n = gg.size();
        g.assign(n, vector<par>());
        treeSize.assign(n, 0);
        spChild.assign(n, -1);
        build(gg, 0);
        hpStart.resize(n);
        rep(i, n)
            hpStart[i] = i;
        hpbuild(0);
        lg = LcaGetter(...);
    }

    void query(int u, int v) {
        int lca = lg.query(u, v);
        return query(u) + query(v) - query(lca)*2; //generalizar
    }

    void update(int u, int v, int w) {
        ... transformar a update(u, i, w) ...
    }
};
```

Código de rolling hash (UVa: Glass Beads) (¡Sufijos cíclicos!) (apunte antiguo)

```
const intt X = 33; //127 para alfabetos gigantes
const int N = 10000; //largo máximo del string
vector<intt> x(N + 1, 1);
void precalculo_rh() {
    for (int i = 1; i <= N; ++i)
        x[i] = x[i - 1]*X;
}
int charn(int c) {
    return c - 'a' + 1; //o lo que sea, depende de alfabeto (bug +1 arreglado)
}

//hashs da el hash de un substring dados los hash de los sufijos
intt hashs(vector<intt>& h, int n, int start, int len) {
    if (start + len > n) //substring cíclico, eliminable si no aplica
        return h[start] + hashs(h, n, 0, (start + len) % n)*x[n - start];

    if (start + len == n) //hash de un sufijo
        return h[start];

    return h[start] - h[start + len]*x[len]; //hash de un substring normal
}

bool es_menor(string& str, vector<intt>& h, int n, int i, int j) {
    int l = 1, r = n; //prefijos desde largo 1 hasta n
    while (l < r) {
        int m = (l + r)/2;
        if (hashs(h, n, i, m) != hashs(h, n, j, m)) //encontrar primer prefijo distinto
            r = m;
        else
            l = m + 1;
    }
    int idiff = (i + l - 1 + n) % n, jdiff = (j + l - 1 + n) % n; //como es bs sobre largo el
índice es -1
    return (str[idiff] < str[jdiff]); //comparación en índice distinto
}

int main() {
    precalculo_rh();
    int T;
        cin >> T;
    for (int t = 0; t < T; ++t) {
        string str;
            cin >> str;
        int n = str.size();
        vector<int64> h(n);
        h[n - 1] = charn(str[n - 1]);
        for (int i = n - 2; i >= 0; --i)
            h[i] = h[i + 1]*X + charn(str[i]);

        int menor = 0;
        for (int i = 1; i < n; ++i)
            if (es_menor(str, h, n, i, menor))
                menor = i;
        cout << menor + 1 << endl;
    }
    return 0;
}
```

DFS para ciclos en Miss Worm (apunte antiguo)

```
void dfs_ciclo(graph& g, int parent, int u, vector<int>& ciclo, vector<int>& color, int& G) {
    color[u] = 1;

    for (int v : g[u]) if (v != parent) {
        switch (color[v]) {
            case 0:
                dfs_ciclo(g, u, v, ciclo, color, G);
                break;
            case 1:
                if (ciclo[v] == 0)
                    ciclo[v] = ++G;
        }

        if (ciclo[v] and color[v] != 3) {
            if (ciclo[u] == 0)
                ciclo[u] = ciclo[v];
            else
                color[u] = 3;
        }
    }
    if (color[u] != 3)
        color[u] = 2;
}
```

?? ?? (dentro del main)

```
vector<int> ciclo(n, 0), color(n, 0);
int G = 0;
dfs_ciclo(g, -1, 0, ciclo, color, G);

rep(i,n)
    cout << ciclo[i] << " ";
```

Longest Increasing Subsequence (apunte antiguo)

```
vector<int> L(A.size(), -1), P;
rep(i,(int)A.size()) {
    int l = 1, r = L.size();
    while (l < r) {
        int m = (l + r)/2;
        if (A[i] < A[L[m]] or L[m] == -1) // < LIS      <= LNDS
                    r = m;
        else
                    l = m + 1;
    }
    L[r] = i;
    P.pb(L[r - 1]);
}
```

```
int iL;
for (iL = 0; iL < L.size() and L[iL] != -1; ++iL) {} //este for no está incompleto

--> LARGO LIS: iL
for (int i = L[iL]; i != -1; i = P[i])
    --> A[i] es el siguiente elemento de dcha a iz en la LIS
```

Criba y Divisores de un número (apunte antiguo) (modificado)

```
const int N = (tamaño);
int criba[N]; rep(i,n) criba[i] = i;
for (intt i = 2; i*i <= N; ++i) if (criba[i] == i)
    for (intt j = i*2; j < N; j+=i)
        criba[j] = i;
int d[N]; d[1] = 1;
for (int i = 2; i < N; ++i) {
    int divisor = criba[i];
    if (i == divisor) d[i] = 2; else {
        intt total = 1; int multi = 0, j = i;
        while (j != 1) {
            multi = 0;
            while (j%divisor == 0) {
                ++multi;
                j/=divisor;
            }
            total*=(multi+1);
            divisor = criba[j];
        }
        d[i] = total;
    }
}
```

Convex Hull (funciona si los puntos son colineales) (se asume que los puntos no se repiten) (los casos con menos de 3 puntos deben ser definidos aparte)

```
#define x first
#define y second
typedef compt double; //tipo de las componentes del punto, podría ser int
typedef pair<compt, compt> Point;
compt cross(Point o, Point a, Point b) {
    return (a.x - o.x)*(b.y - o.y) - (a.y - o.y)*(b.x - o.x);
}

//recibe ch vacío
void convex_hull(vector<Point>& p, vector<Point>& ch) {
    sort(p.begin(), p.end());
    vector<Point> L, U;
    int n = p.size();
    rep(i, n) {
        while (L.size() >= 2 and cross(L[L.size()-2], L.back(), p[i]) <= 0)
            L.pop_back();
        L.pb(p[i]);
    }
    rep(i, n) {
        while (U.size() >= 2 and cross(U[U.size()-2], U.back(), p[n-1-i] <= 0)
            U.pop_back();
        U.pb(p[n-1-i]);
    }
    rep(i,(int)L.size()-1) ch.pb(L[i]);
    rep(i,(int)U.size()-1) ch.pb(U[i]);
}
```

Diámetro de un árbol

```
asignar distancias 1e9 a la componente conexa
aplicar BFS desde un nodo arbitrario U
buscar con un for el nodo X más lejano a U

asignar distancias 1e9 a la componente conexa
aplicar BFS desde X
buscar con un for el nodo Y más lejano a X

el diámetro del árbol corresponde al camino entre X e Y, de largo dist[Y];
```

Tarjan SCC

... (página 134 CP3)

Least Common Ancestor

Usar Segtree simple con RMQ y …

```
E[2n – 1]: E[i] es el iésimo nodo del euler tour
L[2n – 1]: L[i] es el level del nodo E[i]
H[n]:      H[i] es el índice de la primera aparición del nodo i en E
```

para H[u] < H[v]LCA(u, v) = E[RMQ_L(H[u], H[v])]

Union Find

Usar el del CP3 (página 54), está very easy de usar

Max Flow

... (página 165 CP3)

BigInteger

... (página 198 CP3)

Inversos Modulares

Minimum Spanning Tree

... (página 138 CP3)

Primos hasta	
100	25
1000	168
10000	1229
10^5	9592
10^6	78498
10^7	664579
10^8	5761455
10^9	50487534

Potencia de 2 más cercana
(con x<2 ambas retornan 1)

```
int floor_powof2(int x) {
    int ans = 1;
    for (; x >= 2; x/=2)
        ans*=2;
    return ans;
}
```

```
int ceil_powof2(int x) {
    return floor_powof2(x*2-1);
}
```

Binary Indexed Tree / Fenwick Tree
(los índices van de 1 a n !!)

```
template<class T> class FenwickTree {
    vector<T> arr;

public:
    FenwickTree(int n) {arr.assign(n+1, 0);}

    T query(int i) {
        T sum = 0;
        for (; i; i -= i&-i) sum += arr[i];
        return sum;
    }

    T query(int a, int b) {
        return query(b) - (a<=1 ? 0 : query(a-1));
    }

    //puede recibir i > n y no hace nada
    void update(int i, T value) {
        for (; i < (int)arr.size(); i += i&-i)
            arr[i] += value;
    }
};
```

MO

```
struct RMQ{
    static int const neutro = 1e9 + 5;
    static int op(int ans1, int ans2){return min(ans1,ans2);}
};

template<class t> class MO{
    int ne = t::neutro;
    int linearQuery(int l,int r,vector<int> &v){
        int n = ne;
        if(l > r) return ne;
        for(int i = l; i <= r; ++i)
            n = t::op(n,v[i]);
        return n;
    }
    int longQuery(int l,int rq){
        int ll = l/r + 1;
        int rr = rq/r - 1;
        int mint = linearQuery(ll,rr,Pre);
        int f = linearQuery(l,l+r,arr);
        int la = linearQuery(rq-r,rq,arr);
        return t::op(mint,t::op(f,la));
    }
public:
    vector<int> arr; int s,r;
    vector<int> Pre;
    MO(vector<int>& v){
        r = sqrt(v.size()); r += (r*r != v.size()); int s = r*r;
        while(v.size() < s) v.push_back(ne);
        arr = v;
        for(int i = 0; i < s; i+= r)
            Pre.push_back(linearQuery(i,i+r-1,arr));
    }
    int query(int a,int b) {return ( b - a > r ? longQuery(a,b) : linearQuery(a,b,arr));}
};
```

Lowest Common Ancestor (recibe un árbol-grafo)

```
class LCA {
    vector<int> parent, level;
    vector<vector<int>> P;
    graph g; int n, m;

    void build(int u, int p=-1, int lv=0) {
        parent[u] = p;
        level[u] = lv;
        for (int v : g[u]) if (v != p)
            build(v, u, lv+1);
    }

public:
    LCA(graph& gg, int root) {
        g = gg;
        n = g.size();
        parent.resize(n);
        level.resize(n);
        build(root);

        P.assign(n, vector<int>(m=log2(n*2), -1));
        rep(i, n) P[i][0] = parent[i];
        rep(j, m-1) rep(i, n) if (P[i][j] != -1)
            P[i][j+1] = P[ P[i][j] ][j];
    }

    int lca(int u, int v) {
        if (level[u] < level[v]) swap(u, v);
        int i=0, x; for (; x=level[u]-level[v]; ++i)
            if (x & 1<<i) u = P[u][i];
        if (u == v) return u;
        for (++i; i>=0; --i)
            if (P[u][i] != -1 and P[u][i] != P[v][i])
                u = P[u][i], v = P[v][i];
        return parent[u];
    }
};
```

Euler Tour (recibe un árbol-grafo)

```
void euler_tour(graph& g, int u, vector<int>& et, int p=-1) {
    et.pb(u);
    for (int v : g[u]) if (v != p)
        euler_tour(g, v, et, u), et.pb(v);
}
```

```
void sin_regreso(graph& g, int root, vector<int>& et) {
    euler_tour(g, root, et);
    int p=-1;
    #define geb g[et.back()]
    while (geb.size() >= 1 and geb[0] != p)
        p = geb, et.pop_back();
}
```

Transformada Rápida de Fourier (FFT)

```
#include<bits/stdc++.h>
#define rep(i,n) for(int i = 0; i < n; ++i)
using namespace std;

typedef long long ll;
typedef complex<double> cd;
typedef vector<cd> vcd;

#define maxD (1<<19) // potencia de 2 !!

const double PI = 2*acos(0);

cd W[maxD][2], a[maxD],aa[maxD],b[maxD],bb[maxD],cc[maxD], c[maxD];

void fft(cd *v, cd *a, int step, int size, int dir) {
        if (size == 1) a[0] = v[0];
        else{
         size /=2;
         fft(v, a, step*2, size, dir);
         fft(v + step, a + size, step*2, size, dir);
         rep(i,size) {
                 cd even = a[i]; cd odd = a[i + size];
                 a[i] = even + W[i*step][dir] * odd;
                 a[i + size] = even + W[i*step + maxD / 2][dir] * odd;
         }
    }
}

void MulPol(vector<int> &A, vector<int> &B, vector<int> &C){
    rep(i,maxD) {
         double ang = 2*PI*i / maxD;
        a[i] = cd(A[i],0); b[i] = cd(B[i],0);
        W[i][0] = cd(cos(ang), sin(ang));
        W[i][1] = cd(W[i][0].real(), -W[i][0].imag());
        }
        fft(a,aa,1,maxD,0); fft(b,bb,1,maxD,0);
        rep(i,maxD) cc[i] = aa[i] * bb[i];
        fft(cc, c, 1, maxD, 1);
        rep(i,maxD) c[i] /= maxD;
        rep(i,maxD) C[i] = (int)round(c[i].real());
}

int main() {
    string s; // cin >> s;
        int n; cin >> n;
        vector<int> A(maxD,0), B(maxD,0),C(maxD);
        rep(i,n+1) cin >> A[i];
        rep(i,n+1) cin >> B[i];
        MulPol(A,B,C);
         rep(i,2*n+1) cout << C[i] << " ";
        return 0;
}
```

Euler Totient Function programada

```
using namespace std;
const int N = 1e6+1;
int criba[N];

int pw(int base, int exp){
        if(exp == 0) return 1;
        if(exp == 1) return base;
        int aux = pw(base,exp/2);
        return aux*aux*pw(base,exp%2);
}

int eulerT(int n){
        if(n==1) return 1;
        if(criba[n] == n) return n-1;
        int fd = 0; int c = n; while(criba[n] == criba[c]) {c/= criba[c]; fd++;}
        int k = pw(criba[n],fd-1);
        return (criba[n] - 1)*k*eulerT(n/(k*criba[n]));
}

int main(){
                rep(i,N) criba[i] = i;
                for(int i = 2; i*i <= N; ++i) if(criba[i] == i)
                        for(int j = i*2; j < N; j+= i) criba[j] = i;
                int T; cin >> T; rep(t,T){
                        int n; cin >> n; cout << eulerT(n) << endl;
                }
return 0;}
```

Algoritmo de Dinic para flujos rápidos

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

struct flow_graph{
        int MAX_V,E,s,t,head,tail;
        int *cap,*to,*next,*last,*dist,*q,*now;

        flow_graph(){}

        flow_graph(int V, int MAX_E){
        MAX_V = V; E = 0;
        cap = new int[2*MAX_E], to = new int[2*MAX_E], next = new int[2*MAX_E];
        last = new int[MAX_V], q = new int[MAX_V], dist = new int[MAX_V], now = new int[MAX_V];
        fill(last,last+MAX_V,-1);
        }

        void clear(){
        fill(last,last+MAX_V,-1);
        E = 0;
        }

        void add_edge(int u, int v, int uv, int vu = 0){
        to[E] = v, cap[E] = uv, next[E] = last[u]; last[u] = E++;
        to[E] = u, cap[E] = vu, next[E] = last[v]; last[v] = E++;
        }

    bool bfs(){
         fill(dist,dist+MAX_V,-1);
         head = tail = 0;
         q[tail] = t; ++tail;
         dist[t] = 0;

         while(head<tail){
                 int v = q[head]; ++head;

                 for(int e = last[v];e!=-1;e = next[e]){
                         if(cap[e^1]>0 && dist[to[e]]==-1){
                                 q[tail] = to[e]; ++tail;
                                 dist[to[e]] = dist[v]+1;
                         }
                 }
         }

         return dist[s]!=-1;
    }

    int dfs(int v, int f){
         if(v==t) return f;

         for(int &e = now[v];e!=-1;e = next[e]){
                 if(cap[e]>0 && dist[to[e]]==dist[v]-1){
                         int ret = dfs(to[e],min(f,cap[e]));

                         if(ret>0){
                                 cap[e] -= ret;
                                 cap[e^1] += ret;
                                 return ret;
                         }
                 }
         }

         return 0;
    }

    long long max_flow(int source, int sink){
         s = source; t = sink;
         long long f = 0;
         int x;

         while(bfs()){
                 for(int i = 0;i<MAX_V;++i) now[i] = last[i];

                 while(true){
                         x = dfs(s,INT_MAX);
                         if(x==0) break;
                         f += x;
                 }
         }

         return f;
    }
}G;

int main(){
    int V,E,u,v,c;
    //Acelerar I/O
    ios::sync_with_stdio(0);
    cin.tie(0);
    //Lectura vertices aristas
    cin >> V >> E;
    // Inicializacion mega-hermosa
    G = flow_graph(V,E);
    //Lectura del grafo
    for(int i = 0;i<E;++i){
         cin >> u >> v >> c;
         // Los -1 son por numerar desde 1.
         G.add_edge(u-1,v-1,c,c);
    }
    // Respuesta hermosa
    cout << G.max_flow(0,V-1) << endl;

    return 0;
}
```

Gauss Shoelace + Partial sum’s technique

```
const int empty = -1;

void preCalc(vector<ii> &V,vector<double> &PS){
    int n = V.size(); PS[0] = 0.0;
    repx(i,1,n) PS[i] = (i-1 ? PS[i-1] : 0.0) +  V[i].x*(V[i+1].y - V[i-1].y);
}

double Shoelace(vector<ii> &V,vector<double> &PS,int j, int k){
    int n = V.size();
    if(PS[k] == empty) preCalc(V,PS);
    if(k == j) return 0.0;
    if(k < j) return Shoelace(V,PS,0,n-1) - Shoelace(V,PS,k,j);
    return abs(PS[k-1] - (j ? PS[j] : 0.0) + V[k].x*(V[j].y - V[k-1].y)+V[j].x*(V[j+1].y
-V[k].y))/2.0;
}
```