# 3N1?M4 Team Notebook

Liga Cubana de Programación

May 12, 2019

# Contents



# 1 Matemática

## 1.1 Exponenciación e Inverso Modular

```
/* Exponeciacion Modular Iterativa e
 * Inverso Modular
 *
 * Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=2259
 **/

int64 pot(int64 base, int64 e){
    int64 ans = 1ll;

    while( e ){
        if( e & 1ll ) ans = (ans * base) % MOD;
        base = (base * base) % MOD;
        e>>=1ll;
    }

    return ans;
}

inline int64 mod_inv(int64 a, int64 b){
    return (a * pot(b, MOD-2ll)) % MOD;
}
```

## 1.2 Trabajo con Matrices

```
/* Matrix Work
 *
 * - mult() multiplica dos matrices
 * - pot() eleva la matriz al exponente e
 *
 * - El resultado es modulo MOD
 *
 * Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=2542
 * */

typedef vector< vector<int64> > matrix;

/* Multiplica dos matrices */
matrix mult(matrix& a, matrix& b){
    matrix ans(a.size(), vector<int64>(b[0].size(), 0) );

    for(int i=0; i < (int)a.size(); i++)
        for(int j=0; j < (int)b[0].size(); j++)
            for(int k=0; k < (int)b.size(); k++)
                ans[i][j] = (ans[i][j] + a[i][k] * b[k][j] % MOD ) % MOD;
    return ans;
}

/* Eleva una matriz al exponente e */
```

```
matrix pot(matrix& base, int64 e){
     matrix ans(base.size(), vector<int64>(base.size(), 0));

     for(int i=0; i < (int)base.size(); i++) ans[i][i] = 1;

     while( e ){
        if( e & 1ll ) ans = mult(ans, base);
        base = mult(base, base);
        e >>= 1ll;
     }
     return ans;
}
```

## 1.3 Resolución Recurrencias Lineales en $O(K^3 log(N))$

```
/*
 * Resuelve la ecuacion recursiva lineal:
 *
 * f(i) = c_k * f(i-K) + c_(k-1) * f(i-(k-1)) + ... + c_1 * f(i-1) + d
 *
 * - El metodo toma un entero n y un vector de tamanho K+1, cuyos
 *   indices [0...k-1] son los valores iniciales f(1), f(2), ..., f(K)
 *   y en la posicion k esta la constante d.
 * - Tambien toma el vector con los multiplicadores. En las posiciones
 *   [0..k-1] estan c_k, c_(k-1), ..., c_1
 * - Devuelve un vector con los valores [f(n), f(n+1), ..., f(n+K-1)]
 * - Los resultados son modulo MOD y usa las funciones mult() y pot()
 *   de trabajo con matrices
 *
 * Tested on: coj.uci.cu/24h/problem.xhtml?pid=1155
 * */

typedef vector<vector<int64>> matrix;

vector<int64> fast_recurrence(int N, vector<int64>& f, vector<int64>& c){
    int K = f.size();

    matrix F(K);
    matrix T(K, vector<int64>(K, 0) );

    T[K-2] = c; T[K-2].pb(0);

    for(int i=0; i < K; i++){
        F[i] = {f[i]};
        if( i+1 < K ) T[i][i+1] = 1;
    }
    T[K-1][K-1] = 1;

    T = pot(T, N-1);
    F = mult(T, F);

    vector<int64> ans;
    for(auto i: F) ans.pb(i[0]);

    return ans;
}
```

## 1.4 Criba y Funciones de Teoría de Números

```
/*
 * Criba de Eratostenes y Coleccion de Funciones Usadas en Teoria de
     Numeros
 *
 * - Se usan las variables globales mk[] y p para la criba y para
 *   guardar los primos respectivamente.
 *
 * - init()       Construye el vector con los primos menores que MX
 *   Tested on: http://dmoj.uclv.cu/problem/oci19day1a
 *
 * - euler_phi(N) Cuenta el numero de enteros positivos menores que N
 *                que son coprimos con N
 *   Tested on: http://dmoj.uclv.cu/problem/oci19day1a
 *
 * - cant_pf(N)   Cuenta la cantidad de factores primos de N
 *
 * - diff_pf(N)   Cuenta la cantidad de factores primos diferentes de N
 *   Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=3274
 *
 * - cant_div(N)  Cuenta la cantidad de divisores de N
 *
 * - sum_div(N)   Devuelve la suma de los divisores de N
 *   Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=1132
 *
 * */

/* Criba de Eratostenes */
bool mk[MX];
vector<int> p;

void init(){
    for(int64 i=2; i < MX; i++) if( !mk[i] ){
        p.pb(i);
        for(int64 j=i*i; j < MX; j+=i) mk[j] = true;
    }
}

/* Cantidad de Enteros Positivos Menores que N Coprimos con El */
int64 euler_phi(int64 N){
    int64 ans = N;
    for(int i=0; (int64)p[i]*p[i]<=N; i++){
        if( N % p[i] == 0 ) ans -= ans / p[i];
        while( N % p[i] == 0 ) N /= p[i];
    }
    if( N > 1ll ) ans -= ans / N;

    return ans;
}

/* Cantidad de Factores Primos */
int cant_pf(int64 N){
    int ans = 0;

    for(int i=0; (int64)p[i]*p[i]<=N; i++){
        while( N % p[i] == 0 ){
            ans++;
            N /= p[i];
        }
    }
    if( N > 1ll ) ans++;

    return ans;
}

/* Cantidad de Factores Primos Diferentes */
int diff_pf(int64 N){
    int ans = 0;

    for(int i=0; (int64)p[i]*p[i]<=N; i++){
        if( N % p[i] == 0 ) ans++;
        while( N % p[i] == 0 ) N /= p[i];
    }
    if( N > 1ll )
        ans++;
```

```
    return ans;
}

/* Cantidad de Divisores */
int cant_div(int64 N){
    int ans = 1, power;

    for(int i=0; (int64)p[i]*p[i]<=N; i++){
        power = 0;

        while( N % p[i] == 0 ){
            N /= p[i];
            power++;
        }
        ans *= power + 1;
    }
    if( N > 1ll ) ans *= 2;

    return ans;
}

/* Suma de los Divisores */
int64 sum_div(int64 N){
    int64 ans = 1ll, power;

    for(int i=0; (int64)p[i]*p[i]<=N; i++){
        power = 0;

        while( N % p[i] == 0 ){
            N /= p[i];
            power++;
        }
        ans *= ((int64)pow((double)p[i], power + 1.0) - 1) / (p[i] - 1);
    }
    if( N > 1ll )
        ans *= ((int64)pow((double)N, 2.0) - 1) / (N - 1);

    return ans;
}
```

## 1.5 Algoritmo de Euclides Extendido

```
/* Algoritmo de Euclides Extendido
 *
 * Devuelve en un objeto de tipo gcd, tres valores x y d tal que para los
 * a y b dados, se cumple que ax + by = d
 *
 * - Puede que el nombre de la estructura "gcd" no sea aceptado por algunos
 *   compiladores.
 * - Tested on: https://open.kattis.com/problems/generalchineseremainder
 * */

struct gcd{ int64 x, y, d; };

gcd ext_euclid(int64 a, int64 b){
    if( !b ) return {1, 0, a};
    gcd d = ext_euclid(b, a%b);
    return {d.y, d.x -(a/b) * d.y, d.d};
}
```

## 1.6 Teorema Chino del Resto

```
/* Teorema Chino del Resto
 *
 * - Programado en forma de estructura. Se le agregan una a una las
 *   relaciones de congruencia.
 * - Retorna false si no hay solucion. Si la hay (retorna true) esta
 *   en ans y esta tomada modulo lcm.
 *
 * - O(n * log(lcm(m_1, m_2, ..., m_n)) )
 * - Tested on: https://open.kattis.com/problems/generalchineseremainder
 * */

inline int64 norm(int64 a, int64 m){ return (a%m + m) % m; }

struct crt{
    int64 ans, lcm;

    crt():
        ans(0), lcm(1)
    {}

    bool add(int64 a, int m){
        auto d = ext_euclid(lcm, m);
        if( (a - ans) % d.d != 0 ) return false;

        ans = norm(ans + d.x * (a - ans)/d.d % (m/d.d) * lcm, lcm*m/d.d);
        lcm *= m / d.d;
        return true;
    }
};
```

# 2 Manipulación de Bits

## 2.1 Metodos para Manipular Bits

```
/*
   Metodos Utiles con Bits
*/

/* Retorna el siguiente entero con igual cantidad de bits encendidos */
uint next_popcount(uint n){
    uint c = (n & -n);
    uint r = n + c;
    return (((r ^ n) >> 2) / c) | r;
}

/* Retorna el primer entero con n bits encendidos */
uint init_popcount(int n){
    return (1 << n) - 1;
}


/*
 * Metodos de C++ (si se agrega ll al final se puede usar con
 * unsigned long long)
 **/

/* Retorna la cantidad de leading zeros */
int __builtin_clz(unsigned int x)

/* Retorna la cantidad de trailing ceros */
int __builtin_ctz(unsigned int x)

/* Retorna la cantidad de 1-bits */
int __builtin_popcount(unsigned int x)

/* Retorna la cantidad de 1-bits modulo 2 */
```

```
int __builtin_parity(unsigned int x)

/* Retorna 1 + el 1-bit menos significativo de x. Si x == 0, retorna 0 */
int __builtin_ffs(int x)
```

## 2.2 Trie de Bits

```
/* Bit Trie
 *
 * Responde varias queries relacionadas con bits sobre un conjunto
 * de enteros con signo
 *
 * Tested on: https://icpcarchive.ecs.baylor.edu/index.php?Itemid=8&
     category=345&option=com_onlinejudge&page=show_problem&problem=2683
 *
 * */

struct bit_trie{
    vector<int> end;
    vector< vector<int> > T;

    bit_trie(){
       end = vector<int>(32 * MX, -1);
       T.pb(vector<int>(2, -1));
    }
    /* Inserta un entero al conjunto */
    void add(uint N){
        int nod = 0;
        for(int i=31; ~i; i--){
            bool c = N & (1<<i);

            if( T[nod][c] == -1 ){
                T[nod][c] = T.size();
                T.pb(vector<int>(2, -1));
            }
            nod = T[nod][c];
        }
        end[nod] = N;
    }

    /* Retorna el maximo xor de N con un numero dentro del trie */
    uint max_xor(uint N){
        int nod = 0;
        for(int i=31; ~i; i--){
            bool c = N & (1<<i);

            if( ~T[nod][c^1] )
                nod = T[nod][c^1];
            else if( ~T[nod][c] )
                nod = T[nod][c];
            else break;
        }
        if( ~end[nod] )
            return end[nod] ^ N;
        return -1;
    }
};
```

# 3 Combinatoria

## 3.1 Permutaciones: orden y n-ésima

```
/*  Solo para permutaciones sin repeticion
 *
 * - build_permutation: construye una permutacion sin repeticion
 *   dado su orden k y la cantidad de elementos N
 * - permutation_id: dada una permutacion devuelve el orden de esta
 *
 **/

int64 F[MN+10];

void init(){
    F[0] = 1ll;
    for(int64 i=1; i<=MN; i++)
        F[i] = F[i-1] * i;
}

vector<int> build_permutation(int64 k, int64 N){
    vector<int> ans;
    set<int> S;

    for(int i=1; i<=N; i++) S.insert(i);
    k--;

    for(int i=N, get; i > 0; i--){
        get = (int)(k/F[i-1]) + 1;
        k -= (get - 1) * F[i-1];

        for(auto it=S.begin(); ; get--, it++) if( get == 1 ){
            ans.push_back(*it);

            S.erase(it);
            break;
        }
    }

    return ans;
}

int64 permutation_id(vector<int>& V){
    int64 ans = 0;
    set<int> S;

    for(int i=1; i<=V.size(); i++) S.insert(i);

    for(int i=0, n=V.size(); i < V.size(); i++, n--){
        int pos = 1;
        for(auto it=S.begin(); ; pos++, it++) if( *it == V[i] ){
            S.erase(it);
            break;
        }
        ans += (pos - 1) * F[n - 1];
    }

    return ans + 1;
}

/*
   Llamar a init en el main
*/
```

## 3.2 Combinaciones $O(n^2)$

```
/* Triangulo de Pascal
 *
 * - Precalcula las combinaciones con n y k hasta N en O(N^2)
 * - O(1) para responder las queries
```

```
 *
 * Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=3335
 **/

int C[MX][MX];

void build_pascal(){
    for(int i=0; i<=1000; i++)
        C[i][0] = C[i][i] = 1;

    for(int n=1; n<=1000; n++) for(int k=1; k<=1000; k++)
        C[n][k] = ( C[n-1][k] + C[n-1][k-1]) % MOD;
}
```

# 4 Grafos

## 4.1 Dijkstra's Algorithm

```
/* Algoritmo de Dijkstra:
 *
 * - Calcula el camino mas corto desde un nodo a todos los otros
 * - Guarda los resultados en dist
 *
 * Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=1659
 * */


 void dijkstra(int ni, int* dist){
     for(Q.push({ni, 0}); !Q.empty(); Q.pop()){
         int nod = Q.top().nwn;

         if( dist[nod] < Q.top().cost )
             continue;

         for(auto i: G[nod]){
             if( dist[i.nwn] > dist[nod] + i.cost ){
                 dist[i.nwn] = dist[nod] + i.cost;
                 Q.push({i.nwn, dist[i.nwn]});
             }
         }
     }
 }
```

## 4.2 SPFA + Small-Label-First Optimization

```
/* Shortest Path Faster Algorithm + Small-Label-First Optimization
 *
 * Simple SPFA, excepto que al meter un nodo en la cola,
 * si la distancia del frente de la cola es mayor que la
 * de este nodo, en lugar de meterlo detras lo colocamos de primero
 *
 * Expected Complexity: O(E)
 *
 * Tested : [roadplane link]
 **/

bool in_queue[MX];
deque<int> Q;

void add_node(int nod){
    if( !in_queue[nod] ){
        if( dist[nod] < dist[Q.front()] )
            Q.push_front(nod);
        else
            Q.push_back(nod);
        in_queue[nod] = true;
    }
}

/* Inside the main */

for(int i=1; i<=cn; i++) dist[i] = INF;
dist[ni] = 0;

for(Q.push_back(ni); !Q.empty(); ){
    int nod = Q.front();
    Q.pop_front();

    in_queue[nod] = false;

    for(auto i: G[nod]){
        if( dist[i.nwn] > dist[nod] + i.cost ){
            dist[i.nwn] = dist[nod] + i.cost;
            add_node(i.nwn);
        }
    }
}
```

## 4.3 Bellman-Ford Algorithm

```
/* Algoritmo Bellman-Ford
 *
 * - Calcula los caminos minimos desde un nodo hacia otros, pudiendo
 *   haber aristas de costo negativo. Detecta los ciclos de costo
 *   negativo.
 * - O(|V| * |E|)
 *
 * Tested on: wormhole USACO Gold 2006 Dic
 *  */

for(int i=1; i<=cn; i++)
    dist[i] = INF;
dist[1] = 0;

for(int i=1; i < cn; i++)
    for(auto arc: G)
        if( dist[arc.nwn] > dist[arc.nod] + arc.cost )
            dist[arc.nwn] = dist[arc.nod] + arc.cost;

for(auto arc: G)
    if( dist[arc.nwn] > dist[arc.nod] + arc.cost )
        //HAY CICLOS NEGATIVOS
```

## 4.4 Kahn's Algorithm

```
/*
    Kahn's Algorithm

    -  Seudocodigo:
       enqueue vertices with zero incoming degree into a (priority) queue Q
           ;
       while (Q is not empty) {
          vertex u = Q.dequeue();
          put vertex u into a topological sort list;
          remove this vertex u and all outgoing edges from this vertex;

          if such removal causes vertex v to have zero incoming degree
```

```
            Q.enqueue(v);
        }
    - delta guarda el grado de entrada de los nodos
      llenarlo en la entrada
    - Si al final ans no contiene todos los nodos del grafo, el grafo
     tiene un ciclo
    - Si se una una cola de prioridad el orden de los nodos es el menor
      lexicograficamente.

    Tested: http://usaco.org/index.php?page=viewproblem2&cpid=838
*/

 vector<int> kahn(vector<int>& delta){
     vector<int> ans;

     for(int i=1; i<=cn; i++){
        if( !delta[i] )
           Q.push(i);

        off[i] = false;
     }

     for(; !Q.empty(); ){
        int nod = Q.top();
        Q.pop();
        ans.push_back(nod);

        off[nod] = true;
        for(auto i: G[nod]){
             delta[i.nwn]--;

            if( !delta[i.nwn] && !off[i.nwn] )
                Q.push(i.nwn);
        }
    }

    return ans;
 }
```

## 4.5 Diámetro de un Árbol

```
/*
    Diametro de un arbol

    - max_dist guarda el resultado
*/

 int diameter(int nod){
     mk[nod] = 1;
     int _max = 0;

     for(int i=0; i < G[nod].size(); i++){
        int nwn = G[nod][i].nod;
        if(mk[nwn])
           continue;
        int tmp = G[nod][i].cost + diameter(nwn);

        max_dist = max(max_dist, _max + tmp);
        _max = max(_max, tmp);
     }

     return _max;
 }
```

## 4.6 Diámetro de un Árbol Construido

```
/*
 * Halla el diametro de un arbol y construye un
 * vector (path) con los nodos
 *
 * Tested: http://codeforces.com/contest/1068/problem/E
 * */

/*
 * Halla el nodo mas alejado de un nodo y devuelve
 * la respuesta como un par {nodo, distancia}
 *
 * */
 pair<int, int> farther(int nod, int deep=0){
     mk[nod] = true;

     pair<int, int> fnod = {nod, deep};

     for(auto nwn: G[nod]){
         if( !mk[nwn] ){
             auto fnwn = farther(nwn, deep+1);

             if( fnod.second < fnwn.second )
                 fnod = fnwn;
         }
     }
     return fnod;
 }

 /*
  * Dados los dos nodos del diametro, almacena los nodos
  * del diametro en el vector path
  * Usa el booleano flag que es falso antes de la llamada
  * */

 void build_path(int nod, int nf, vector<int>& path){
     mk[nod] = true;

     path.push_back(nod);

     if( nod==nf ){
         flag = true;
         return;
     }

     for(auto nwn: G[nod]){
         if( !flag && !mk[nwn] ){
             build_path(nwn, nf, path);
         }
     }

     if( !flag )path.pop_back();
 }
```

## 4.7 Tarjan's SCC Algorithm

```
/* Tarjan's SCC Algorithm
 * - inc guarda si el nodo esta actualmente en la componente
 *   fuertemente conexa. El dt de un nodo solo actualiza a otro si
 *   esta destro de la pila.
**/

 void tarjan_scc(int nod){
```

```
        dt[nod] = low[nod] = ++t;
        K.push(nod);
        inc[nod] = true;

        for(auto nwn: G[nod]){
            if( !dt[nwn] ){
                tarjan_scc(nwn);
                low[nod] = min(low[nod], low[nwn]);
            }
            else if( inc[nwn] )
                low[nod] = min(low[nod], dt[nwn]);
        }

        if( dt[nod] == low[nod] ){
            while( K.top() != nod ){
                SET[K.top()] = nod;
                inc[K.top()] = false;
                K.pop();
            }
            SET[K.top()] = nod;
            inc[K.top()] = false;
            K.pop();
        }
    }
```

# 5 Árboles

## 5.1 Lowest Common Ancestor

```
/* Lowest Common Ancestor
 *  - Version simple para calcular distancias
 *
 *  Prec: O(cn*log(cn))
 *  Query: O(log(cn))
 *
 *  Tested: http://coj.uci.cu/24h/problem.xhtml?pid=1239
 * */

struct lca_tree{
    int cn;
    int L[MX], P[MX][20];
    bool mk[MX];
    int64 dist[MX];
    vector<par> G[MX];

    void add_edge(int a, int b, int c){
        G[a].pb({b, c});
        G[b].pb({a, c});
    }

    void dfs(int nod){
        mk[nod] = true;

        for(auto i: G[nod]) if( !mk[i.nwn] ){
            P[i.nwn][0] = nod;
            L[i.nwn] = L[nod] + 1;
            dist[i.nwn] = dist[nod] + i.cost;

            dfs(i.nwn);
        }

    }

    void build(){
        dist[1] = 0ll, L[1] = 1;
        dfs(1);

        for(int j=1, l=log2(cn); j<=l; j++)
            for(int i=1; i<=cn; i++)
                P[i][j] = P[ P[i][j-1] ][j-1];
    }

    int query(int a, int b){
        if( L[b] > L[a] )
            swap(a, b);
        for(int i=log2(L[a]); i>=0; i--)
            if( P[a][i] && L[ P[a][i] ] >= L[b] )
                a = P[a][i];
        if( a==b )
            return a;

        for(int i=log2(L[a]); i>=0; i--)
            if( P[a][i] && P[a][i] != P[b][i] )
                a = P[a][i], b = P[b][i];
        return P[a][0];
    }

    int64 shortest_path(int a, int b){
        return dist[a] + dist[b] - 2ll * dist[query(a, b)];
    }
};
```

## 5.2 Centroid Decomposition

```
/* Centroid Decomposition
 *
 * - up[i] guarda el padre del nodo i en el centroide
 *
 * Tested on: [Copy link of  Xenia and Tree ]
 *  */


struct graph{
    int cn;
    int up[MX], sz[MX];
    bool mk[MX];
    vector<int> G[MX];

    void add_edge(int a, int b){
        G[a].pb(b);
        G[b].pb(a);
    }

    int dfs0(int nod, int p){
       sz[nod] = 1;

        for(auto nwn: G[nod]) if( !mk[nwn] && nwn != p )
           sz[nod] += dfs0(nwn, nod);
       return sz[nod];
    }

    int dfs1(int nod, int p, const int& tam){
        for(auto nwn: G[nod]){
            if( !mk[nwn] && nwn != p && sz[nwn] > tam )
                 return dfs1(nwn, nod, tam);
        }
        return nod;
    }

    int centdec(int nod){
       int tam = dfs0(nod, -1);
```

```
        int cen = dfs1(nod, -1, tam>>1);

        mk[cen] = true;

        for(auto nwn: G[cen]) if( !mk[nwn] ){
            int c = centdec(nwn);
            up[c] = cen;
        }
        return cen;
    }

    void build(){
        // Here we build a LCA or something else...

        centdec(1);
    }
};
```

# 6 Matching y Flujo

## 6.1 Ford-Fulkerson $O(E^2 \cdot log(c))$

```
/* Ford-Fulkerson Method with Scaling Algorithm
 *
 *  - Inicializar cn, ca, ni, nf;
 *  - O(E^2 * log(max_edge))
 *
 * Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=1695
 * */

struct fordfulk{
    int cn, ca, ni, nf, max_edge, f;
    int P[MX];
    int G[MX][MX];
    bool mk[MX];

    void add_edge(int a, int b, int c){
        G[a][b] = c;
        max_edge = max(max_edge, c);
    }

    bool augment(int nod, int th){
        mk[nod] = true;

        if( nod==nf )
            return true;

        int ans = false;
        for(int nwn=1; nwn<=cn; nwn++){
            if( !mk[nwn] && G[nod][nwn] > th ){
                P[nwn] = nod;
                ans |= augment(nwn, th);
            }
        }

        return ans;
    }

    void resolve(int nod, int min_edge){
        if( nod==ni ){ f = min_edge; return; }
        else{
            resolve(P[nod], min(min_edge, G[ P[nod] ][nod]));
            G[ P[nod] ][nod] -= f, G[nod][ P[nod] ] += f;
        }
    }

    int throw_flow(){
        int mf = 0, th = max_edge;

        while( true ){
            for(int i=0; i<=cn; i++)
                mk[i] = false;

            if( augment(ni, th) ){
                resolve(nf, INF);
                mf += f;
            }
            else if( !th )
                break;
            else
                th>>=1;
        }
        return mf;
    }
};
```

# 7 Estructuras de Datos

## 7.1 Union-Find

```
/*
   Union-Find
*/

int SET[MN], R[MN];

void init(){
    for(int i=1; i<=cn; i++)
        SET[i] = i, R[i] = 1;
}

int find_set(int nod){
    if( nod != SET[nod] )
        return find_set(SET[nod]);
    return nod;
}

void join_set(int nod, int nwn){
    if( R[nod] > R[nwn] )
        SET[nwn] = nod, R[nod]++;
    else
        SET[nod] = nwn, R[nwn]++;
}
```

## 7.2 Cola que Soporta la Operación Máximo en $O(1)$

```
/* Doble Stack Queue
 *
 * - Soporta la operacion de maximo (minimo) en los
 * elementos que contiene
 *
 *  - Para el minimo :
 *      - numeric_limits<T>::min() ---->  numeric_limits<T>::max()
 *      - ans < s1.top().second ------->  ans < s1.top().second
 *      - max(s2.top().first... ------->  min(s2.top().first...
 *      - max(val, ... ---------------->  min(val, ...
 *  - O tambien puedes introducir el opuesto del valor y cuando
```

```
 *    lo saques de la cola, multiplicarlo por -1.
 *
 * - (Not Tested)
 * */

template <class T>
struct ds_queue{
    stack< pair<T, T> > s1, s2;

    int size(){ return s1.size() + s2.size(); }
    int empty(){ return s1.empty() && s2.empty(); }

    T top(){
       T ans = numeric_limits<T>::min();

       if( !s1.empty() && ans < s1.top().second )
           ans = s1.top().second;
       if( !s2.empty() && ans < s2.top().second )
           ans = s2.top().second;

       return ans;
    }

    void push(T val){
        if( s2.empty() )
            s2.push({val, val});
        else
            s2.push({val, max(val, s2.top().second)});
    }

    void pop(){
        if( s1.empty() ){
             while( !s2.empty() ){
                 if( s1.empty() )
                     s1.push({s2.top().first, s2.top().first});
                 else
                     s1.push({s2.top().first, max(s2.top().first, s1.top()
                         .second)});
                 s2.pop();
             }
        }
        assert( !s1.empty() );
        s1.pop();
    }
};
```

## 7.3 Set con Order Statics

```
/*
   Set con Order Statics  en la STL

   Para usarlo como multiset se guardan los elementos como un par {element,
        id}
   y asi diferenciarlos. Luego para hacer referencia a un elemento se busca
        el
   par {element, 0}

   Tested: https://codeforces.com/contest/459/problem/D
*/
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

//Declaracion
typedef tree<
   pair<int, int>,
   null_type,
   less< pair<int, int> >,
   rb_tree_tag,
   tree_order_statistics_node_update>
ordered_set;

int t = 0;
ordered_set Set;

//Insertar
Set.insert( {T[i], ++t} );
//Eliminar uno
Set.erase( Set.lower_bound({T[i], 0}) );
//Cantidad de menores que p
Set.order_of_key( {p, 0} )
//Cantidad de mayores o iguales que p
Set.size() - Set.order_of_key( {p, 0} );
//Cantidad de mayores que p
Set.size() - Set.order_of_key( {p + 1, 0} );
//Devuelve un iterador al elemeto en la posicion p (comenzando por cero)
Set.find_by_order(p)
```

## 7.4 Tabla Acumulativa de Dos Dimensiones

```
/* Tabla Acumulativa
 *
 * Se define el rectangulo dentro de una matriz,
 * de la forma:
 *
 * o--c--o
 * |     |
 * a     b
 * |     |
 * o--d--o
 * */

struct accumulative{
    int N;
    int A[MX][MX], acum[MX][MX];

    void build(){
        for(int i=1; i<=N; i++)
            for(int j=1; j<=N; j++)
                acum[i][j] = acum[i-1][j] + acum[i][j-1] - acum[i-1][j-1],
                acum[i][j] += A[i][j] > 0;
    }

    int query(int a, int b, int c, int d){
        return acum[d][b] - acum[c-1][b] - acum[d][a-1] + acum[c-1][a-1];
    }
};
```

## 7.5 Range Min-Max Query

```
/* Range Min-Max Query (template)
 *
 * - El constructor acepta el areglo base, el numero de elementos,
 * y una funcion asociativa(min, max, gcd, ...)
 *
 * - En la mayoria de los casos hay que programar nuestras funciones
 *
 * */

template<class any>
struct rmq{
```

```
    int N;
    function<any(any, any)> f;
    vector< vector<any> > T;

    rmq(any *A, int sz, any (*fun)(any, any)){
        N = sz;
        f = *fun;
        T.resize(sz + 5, vector<any>(20, 0) );

        for(int i=1; i<=N; i++)
            T[i][0] = A[i];

        for(int j=1, l=log2(N); j<=l; j++){
            N -= 1<<(j-1);
            for(int i=1; i<=N; i++)
                T[i][j] = f(T[i][j-1], T[i + (1<<(j-1))][j-1]);
        }
        N = sz;
    }

    any query(int a, int b){
        int l = log2(b-a+1);
        return f(T[a][l], T[b-(1<<l)+1][l]);
    }
};

inline int _min(int a, int b){ return min(a, b); }
inline int _max(int a, int b){ return max(a, b); }
```

## 7.6 Wavelet Tree

```
/* Wavelet Tree
 *
 * - v[i] representa la cantidad de elementos que van en el subarbol
 *       izquierdo. Los i - v[i] restantes van en el subarbol derecho.
 * - kth(): Devuelve el k-esimo elemento en el rango
 * - lte(): Devuelve la cantidad de elementos menores o iguales a k
 *          en el rango
 * - great(): Devuelve la cantidad de elementos mayores que k en el
 *          rango
 * - count(): Devuelve la cantidad de elementos iguales k en el rango
 *
 * - Prep: O(n*log(Max))
 * - Queries: log(Max)
 *
 * Tested on: promote (USACO 2017 January Contest, Platinum)
**/

struct wavelet_tree{
    int lo, hi;
    vector<int> v;
    wavelet_tree *L, *R;

    wavelet_tree(){}
    wavelet_tree(int *x, int *xend, int a, int b){
        lo = a, hi = b;

        if( lo==hi || x>=xend ) return;

        int m = (lo+hi)/2;
        auto f = [m](int p){ return p <= m; };
        v.reserve(xend - x + 1); v.pb(0);

        for(auto it=x; it!=xend; it++)
            v.pb(v.back() + f(*it));

        auto pivot = stable_partition(x, xend, f);
        L = new wavelet_tree(x, pivot, a, m);
        R = new wavelet_tree(pivot, xend, m+1, b);
    }

    int lte(int a, int b, int k){
        if( a > b || k < lo ) return 0;
        if( hi<=k ) return b - a + 1;

        return L->lte(v[a-1]+1, v[b], k) + R->lte(a - v[a-1], b - v[b], k);
    }

    int great(int a, int b, int k){
        return b - a + 1 - lte(a, b, k);
    }

    int kth(int a, int b, int k){
        if( a > b ) return 0;
        if( lo==hi ) return lo;

        int in_left = v[b] - v[a-1];

        if( k<=in_left )
            return L->kth(v[a-1] + 1, v[b], k);
        else
            return R->kth(a - v[a-1], b - v[b], k - in_left);
    }

    /* These following methods are not tested yet */
    int count(int a, int b, int k){
        if( a > b || k < lo || k > hi ) return 0;
        if( lo==hi ) return b - a + 1;

        int m = (lo+hi)/2;

        if( k<=m ) return L->count(v[a-1] + 1, v[b], k);
        else return R->count(a - v[a-1], b - v[b], k);
    }

    ~wavelet_tree(){
        delete L;
        delete R;
    }
};
```

## 7.7 Mo's Algorithm

```
/* Mo's Algorithm
 *
 * - Ordena las queries, garantizando un tiempo de O(N * sqrt(N))
 *
 * Solution for: (circlecroos) Why Did the Cow Cross the Road III
 *               USACO 2017 February Contest, Gold
 **/

struct mo{
    struct query{
        int x, xend, id, ans;
    };

    struct bundle{
        int cant;
        bool C[MX];
        bundle(): cant(0){}

        void add(int x){
```

```
            if( C[x] ) cant--;
            else cant++;

            C[x] ^= 1;
        }
    };

    int N;
    int A[MX];
    vector<query> Q;

    void add_query(int a, int b, int id){
        Q.pb({a, b, id, -1});
    }

    vector<int> solve(){
        sort(all(Q), [this](query& a, query& b){
            int sq = sqrt(Q.size());
            if( a.x/sq != b.x/sq ) return a.x/sq < b.x/sq;
            return a.xend/sq < b.xend/sq;
        });

        int l = 1, r = 0;
        bundle B;

        for(auto& q: Q){
            while( r < q.xend )
                B.add(A[++r]);
            while( l < q.x )
                B.add(A[l++]);
            while( r > q.xend )
                B.add(A[r--]);
            while( l > q.x )
                B.add(A[--l]);

            q.ans = B.cant;
        }
        vector<int> ans(Q.size() + 1);
        for(auto q: Q)
            ans[q.id] = q.ans;

        return ans;
    }
};
```

## 7.8 Fenwick Tree

```
/* Fenwick Tree (Binary Indexed Tree)
 *
 * Tested on: http://dmoj.uclv.cu/problem/oci19day2b
 * */

#define MN  (int)1e6

struct fenwick_tree{
    int T[MN+5];

    int acc(int b){
        int ans = 0;
        for(; b; b-=(b&(-b))) ans += T[b];
        return ans;
    }

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

    void update(int p, int val){
        for(; p<=MN; p+=(p&(-p)))
            T[p] += val;
    }
};
```

# 8 Strings

## 8.1 Knuth-Morris-Pratt Algorithm

```
/*
    Knuth-Morris-Pratt Algorithm
    - fail guarda el largo del mayor prefijo que es
      a la vez sufijo del patron
*/
scanf("%s\n%s\n", &T, &P);
lt = strlen(T);
lp = strlen(P);

fail[0] = 0;
for(int i=1; i < lp; i++){
    st = fail[i-1];
    while( st > 0 && P[st]!=P[i] ) st = fail[st-1];
    if( P[st]== P[i])
        fail[i] = st + 1;
}

for(int i=0; i < lt; i++){
    while( st > 0 && P[st]!=T[i]) st = fail[st-1];
    if(P[st]==T[i]) st++;
    if( st==lp ){
        printf("There is an occurrence at %d\n", i-lp+1);
        st = fail[st-1];
    }
}
```

## 8.2 Función Z

```
/* Z-Algorithm
 *
 * - Recibe una cadena y devuelve un vector en el que cada posicion i
 *   contiene el lcp entre el sufijo de indice cero y el de indice i.
 * - O(n)
 *
 * Not Tested Yet!!!
 * */

vector<int> z_function(string& S){
    int N = S.size();
    vector<int> z(N);

    z[0] = N;
    for(int i=1, l=0, r=0; i < N; i++){
        if( i <= r )
            z[i] = min(r-i+1, z[i-l]);
        while( i + z[i] < N && S[ z[i] ] == S[ i + z[i] ] )
            ++z[i];
        if( i + z[i] - 1 > r )
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

## 8.3 Suffix Array en $O(n * log^2(n))$

```
/* Suffix Array
 *
 * - Las posiciones de cada sufijo i en el orden del suffix array
 *   se guardan en la fila P[i][k-1]
 * - Prec: O(N * log^2(N))
 * - lcp: O(log(N))
 *
 * Tested on: hackerrank.com/challenges/string-similarity/problem
**/

struct suffix{
    int a, b, p;

    bool operator < (const suffix &p)const{
        return a==p.a ? b < p.b : a < p.a;
    }
    bool operator == (const suffix &p)const{
        return a==p.a && b == p.b;
    }
};

struct suffix_array{
    int ls, k, del;
    vector<vector<int> > P;
    suffix L[MX];

    suffix_array():
        P(MX, vector<int>(20, 0))
    {}

    void build(char *S){
        ls = strlen(S);

        for(int i=0; i < ls; i++)
            P[i][0] = S[i] - 'a';

        for(k=del=1; (del>>1) < ls; k++, del<<=1) {
            for(int i=0; i < ls; i++){
                L[i].a = P[i][k-1];
                L[i].b = (i + del < ls) ? P[i+del][k-1] : -1;
                L[i].p = i;
            }
            sort(L, L+ls);

            for(int i=0; i < ls; i++)
                P[ L[i].p ][k] = (i && L[i]==L[i-1]) ? P[ L[i-1].p ][k]: i;
        }
    }

    int lcp(int x, int y){
        if( x==y ) return ls-x;

        int ans = 0;

        for(int i=k-1; i>=0 && x < ls && y < ls; i--){
            if( P[x][i] == P[y][i] )
                x += (1<<i), y += (1<<i), ans+=(1<<i);
        }
        return ans;
    }
};
```

## 8.4 Suffix Tree Implícito en $O(n * log^2(n))$

```
/*
 * Implicit DFS on Suffix Tree with Suffix Array
 *
 * - Hay que implementar la funcion next y dfs dentro del suffix array.
 *   next devuelve la ultima posicion menor que xend en el arreglo de
 *   sufijos que posee mas de p caracteres comunes con la posicion x.
 * - Cada tripla (x, xend, p) para la que se llama a la funcion dfs
 *   representa un nodo del suffix tree implicito.
 *
 * */

int next(int x, int xend, int p){
    if( sa[x] + p >= N ) return x;

    int ans = x;
    for(int i=log2(xend-x+1); ~i; i--)
        if( ans + (1<<i) <= xend && sa[ans + (1<<i)] + p < N &&
            lcp(sa[ans + (1<<i)] + p, sa[x] + p) )
            ans += (1<<i);
    return ans;
}

void dfs(int x, int xend, int p){
    printf("%d %d %d\n", x, xend, p);

    if( x==xend && sa[x] + p >= N ) // Es un nodo final
        return;

    for(int ii=x, jj; ii<=xend; ii=jj+1){
       jj = next(ii, xend, p);

       dfs(ii, jj, lcp(sa[ii], sa[jj]));
    }
}
```

## 8.5 Hashing para Cadenas

```
/**
 * Hash para Strings
 *
 * - Llamar a init() dentro del main
 * - Preproc: O(n)
 * - compare(int a, int b, int len) devuelve true
 *   si S[a+len-1] == S[b+len-1] en O(1)
 * - hashing(int a, int b) devuelve el hashing relativo de
 *   la subcadena S[a..b] en O(1). Dos hashings de este tipo
 *   pero pertenecientes a dos cadenas diferentes, pueden
 *   compararse si la basede ambos es la misma.
 *
 * Tested: http://coj.uci.cu/24h/problem.xhtml?pid=1624
 ***/

const unt64 p = 31ull;
unt64 P[MX];

void init(){
    P[0] = 1ull;

    for(int i=1; i < MX; i++)
        P[i] = P[i-1] * p;
}
```

```
struct hasher{
    int l;
    unt64 H[MX];

    hasher(){}
    hasher(string &S){
        l = S.size();

        for(int i=0; i < l; i++){
            H[i] = (unt64)(S[i] - 'a' + 1) * P[i];

            H[i] += i ? H[i-1] : 0ull;
        }
    }
    /* not tested yet */
    bool compare(int a, int b, int len){
        if(b < a) swap(a, b);

        unt64 h1 = H[a + len - 1]; if(a) h1 -= H[a-1];
        unt64 h2 = H[b + len - 1]; if(b) h2 -= H[b-1];

        if( h1 * P[b - a] == h2 )
            return true;
        else
            return false;
    }

    unt64 hashing(int a, int b){
        if( a > b ) return 0ull;
        return (H[b] + ( a ? (1ull<<64) - H[a-1] : 0ull)) * P[MX-1-a];
    }
 };
```

## 8.6 Palindromizer

```
/**
 *  Palindromizer
 *
 *  - Preproc: O(n)
 *  - Devuelve si S[a, b] es un palindrome en O(1)
 *
 *  Tested: http://coj.uci.cu/24h/problem.xhtml?pid=1624
 * */

struct palindromizer{
    hasher A, R;

    palindromer(string S){
        A = hasher(S);

        for(int i=0, j=S.size()-1; i < j; i++, j--)
            swap(S[i], S[j]);

        R = hasher(S);
    }

    bool is_palin(int a, int b){
         return A.hashing(a, b) == R.hashing(A.l - b - 1, A.l - a - 1);
    }
};
```

## 8.7 Aho-Corasick Algorithm

```
/*  Aho-Corasick
 *
 * - Implementacion con arreglo de vectores
 *
 * - T es el Trie, ac[i] guarda los ids de las cadenas que
 *   terminan en el nodo i, y F es un arreglo de punteros
 *   enteros al nodo del sufijo propio mas largo que es tambien
 *   sufijo de un patron.
 *
 * - match() devuelve un vector de pares donde cada par guarda el
 *   1-indice del patron segun el orden en que fueron agregados
 *   al trie y la ultima posicion de una ocurrencia.
 *
 * - cn es la cantidad de nodos y sz es la cantidad de cadenas
 *   almacenadas
 *
 * - Se puede guardar tambien el largo de las cadenas en un arreglo
 *   aparte para deducir el comienzo de las ocurrencias.
 *
 * - Tested on: Croatia 2006 severina
 *
 * */

#define SIGMA 26

struct aho_corasick{
     int cn, sz;
     int F[MX];
     vector<int> T[MX], ac[MX];

     aho_corasick(){
        cn = 1, sz = 0;
        T[0] = vector<int>(SIGMA, -1);
     }

     void add(char S[]){
         int nod = 0;
         int N = strlen(S);
         sz++;

         for(int i=0; i < N; i++){
             int c = S[i] - 'a';

             if( T[nod][c] == -1 ){
                 T[++cn] = vector<int>(SIGMA, -1);
                 T[nod][c] = cn;
             }
             nod = T[nod][c];
         }
         ac[nod].pb(sz);
     }

     void build(){
         queue<int> Q;
         for(int c=0; c < SIGMA; c++) if( T[0][c] != -1 )
             Q.push(T[0][c]);

         for(; !Q.empty(); Q.pop() ){
             int nod = Q.front();

             for(int c=0; c < SIGMA; c++) if( T[nod][c] != -1 ){
                 int st = F[nod];

                 while( st > 0 && T[st][c] == -1 ) st = F[st];

                 if( T[st][c] != -1 )
                     st = T[st][c];
                 F[ T[nod][c] ] = st;

                 for(auto i: ac[st]) ac[ T[nod][c] ].pb(i);
```

```
                Q.push(T[nod][c]);
            }
        }
    }

    vector< pair<int, int> > match(char A[]){
        int N = strlen(A);
        vector< pair<int, int> > ans;

        for(int i=0, st=0; i < N; i++){
            int c = A[i] - 'a';

            while( st > 0 && T[st][c] == -1 ) st = F[st];

            if( T[st][c] != -1 ){
                st = T[st][c];

                for(auto feat: ac[st])  ans.pb({feat, i});
            }
        }

        return ans;
    }
};
```

## 8.8 ReTRIEval Tree

```
/* Trie
 *
 * - Implementacion simple con vectores
 *
 * */

#define SIGMA 26

struct trie{
    bool end[MX];
    vector< vector<int> > T;

    trie(){
        T.pb(vector<int>(SIGMA, -1));
    }

    void add(string S){
        int nod = 0;

        for(int i=0; i < (int)S.size(); i++){
            int c = S[i] - 'a';

            if( T[nod][c] == -1 ){
                T[nod][c] = T.size();
                T.pb(vector<int>(SIGMA, -1));
            }
            nod = T[nod][c];
        }
        end[nod] = true;
    }

    bool find(string S){
        int nod = 0;

        for(int i=0; i < (int)S.size(); i++){
            int c = S[i] - 'a';

            if( T[nod][c] == -1 )
                return 0;
            else nod = T[nod][c];
        }
        return end[nod];
    }
};
```

## 8.9 Dictionary Of Basic Factors en $O(n * log^2(n))$

```
/* Dictionary Of Basic Substrings
 *
 * - Dada una subcadena, proporciona un par ordenado (factor) que sirve
 *   para compararla con otra subcadena de igual largo, lexicograficamente.
 *
 * - Preproc: O(n * log^2(n))
 * - Query: O(1)
 * - Tested on: https://coj.uci.cu/24h/problems.xhtml?pid=3540
 *  */

struct factor{
    int a, b, p;

    bool operator < (const factor& p)const{
        return a == p.a ? b < p.b : a < p.a;
    }
    bool operator == (const factor& p)const{
        return a == p.a && b == p.b;
    }
};

struct fact_dict{
    int N;
    vector<factor> L;
    vector<vector<int>> P;

    fact_dict(string& S):
        N(S.size()),
        P(N, vector<int>(log2(N) + 1, -1))
    {
        for(int i=0; i < N; i++)
            L.pb({S[i] - 'a', 0, i});

        sort(all(L));

        for(int i=0; i < N; i++)
            P[ L[i].p ][0] = i && L[i]==L[i-1] ? P[ L[i-1].p ][0] : i;

        for(int j=1, l = log2(N); j<=l; j++){
            N -= (1<<(j-1));
            L.clear();

            for(int i=0; i < N; i++)
                L.pb({P[i][j-1], P[i + (1<<(j-1))][j-1], i});

            sort(all(L));

            for(int i=0; i < N; i++)
                P[ L[i].p ][j] = i && L[i]==L[i-1] ? P[ L[i-1].p ][j] : i;
        }
    }

    factor fact(int a, int b){
        int k = log2(b - a + 1);
        return {P[a][k], P[b - (1<<k) + 1][k], -1};
    }
};
```

## 8.10 Manachar's Algorithm

```
/* Manachar's Algorithm
 *
 * - Recibe un cadena S de largo n.
 * - Devuelve un vector de largo 2*n-1, donde las posiciones 2*i
 *   contienen el largo del mayor palindrome impar con centro en el
 *   caracter i. Las posiciones 2*i-1 guardan el largo del mayor
 *   palindrome par con centro entre los caracteres i, i+1.
 * - En otras palabras, devuelve el vector relativo a "a#b#c#d#e" donde
 *   "abcde" es la cadena original.
 * - Para recorrer todos los caracteres de un palindrome centrado en j
 *   (relativo al vector resultante y correspondiente a la cadena
 *   "a#b#c#d#e") se usa un ciclo que vaya de dos en dos, desde
 *   (j - p[j] + 1) hasta (j + p[j] - 1) inclusive. Cada posicion real
 *   en la cadena tiene indice j/2.
 *
 * - Complejidad: O(n)
 * - Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=1389
 * */

vector<int> manachar(string& S){
    int N = S.size();
    vector<int> ans(2 * N - 1);

    for(int i=0, j=0, k=0; i < 2*N-1; i+=k){
        while( i-j >= 0 && i+j+1 < 2*N && S[(i-j)/2] == S[(i+j+1)/2] )
            j++;
        ans[i] = j;

        for(k=1; k<=ans[i] && ans[i-k] != ans[i] - k; k++)
            ans[i+k] = min(ans[i-k], ans[i] - k);
        j = max(j-k, 0);
    }
    return ans;
}
```

## 8.11 Palindromic Tree

```
/* Palindromic Tree
 *
 * Tested on: http://acm.timus.ru/problem.aspx?space=1&num=1960
 *  */

#define SIGMA 26

struct node{
    int a, b, link;
    int ins[SIGMA];

    node(){
         a = b = link = 0;;
         memset(ins, 0, sizeof ins);
    }
    node(int _a, int _b, int _link){
         a = _a, b = _b, link = _link;
         memset(ins, 0, sizeof ins);
    }
};

struct eertree{
    int nod, sz;
    vector<node> T;
    string S;

    eertree(){
        T = vector<node>(MX);
        T[1] = node(1, -1, 1);
        T[2] = node(1, 0, 1);
        nod = 1, sz = 2;
    }

    int size(){ return sz - 2; }

    void add(char c){
        int t = nod, len;
        S += c;

        while( true ){
            len = T[t].b - T[t].a + 1;

            if( S.size() - len > 1 && S[S.size()-len-2] == c )
                break;

            t = T[t].link;
        }

        if( T[t].ins[c -'a'] ){
            nod = T[t].ins[c -'a'];
            return;
        }

        T[t].ins[c -'a'] = ++sz;
        T[sz] = node(S.size()-len-2, S.size()-1, 0);

        t = T[t].link;
        nod = sz;

        if( T[nod].b - T[nod].a == 0 ){
            T[nod].link = 2;
            return;
        }

        while( true ){
            len = T[t].b - T[t].a + 1;

            if( S.size() - len > 1 && S[S.size()-len-2] == c )
                 break;

            t = T[t].link;
        }

        T[nod].link = T[t].ins[c-'a'];
    }
};
```

# 9 Geometría

## 9.1 Geometry Library

```
/* Geometry Library
 *
 * Not Tested Yet!!!
 *  */

#define EPS 1e-12
typedef double type;

/* Funciones Basicas y Constantes */
```

```
const type PI = acos(-1);

/* Conversions */
inline type deg_to_rad(type deg){ return deg * PI / 180.0; }
inline type rad_to_deg(type rad){ return rad * 180.0 / PI; }

/* Sign */
int sgn(type a){ return a < -EPS ? -1 : a > EPS; }

/* Class Point-Vector */
struct pt{
    type x, y;

    /* Basics */
    pt(type x, type y): x(x), y(y){}
    pt operator + (const pt p){ return {x + p.x, y + p.y}; }
    pt operator - (const pt p){ return {x - p.x, y - p.y}; }
    pt operator * (type d){ return {x * d, y * d}; }
    pt operator / (type d){ return {x / d, y / d}; }
    bool operator == (const pt p)
        { return fabs(x - p.x) < EPS && fabs(y - p.y) < EPS; }

    /* Dot and Cross Product */
    type operator * (const pt p){ return x * p.x + y * p.y; }
    type operator ^ (const pt p){ return x * p.y - y * p.x; }

    /* Norma del vector */
    type norm(){ return sqrt(x * x + y * y); }

    /* Norma al Cuadrado del Vector */
    type norm2(){ return x * x + y * y; }

    /* Traslada un punto en la direcccion del vector vec */
    pt translate(pt vec){ return *this + vec; }

    /* Escala el punto, a un cierto ratio, alrededor de un centro c */
    pt scale(pt c, type ratio){ return c + (*this - c) * ratio; }

    /* Rota el punto un angulo a */
    pt rot(type a){ return {x * cos(a) - y * sin(a), x * sin(a) + y * cos(a
        ) }; }

    /* Vector perpendicular */
    pt perp(){ return {-y, x};  }
};

/* Stream Overload for points */
ostream &operator << (ostream &os, const pt &p) {
    return os << "(" << p.x << ", " << p.y << ")";
}

/* Distance between two points */
type dist(pt& a, pt& b){ return sqrt((a - b) * (a - b)); }

/* Square Distance between two points */
type dist2(pt& a, pt& b){ return (a - b) * (a - b); }

/* Angle between two vectors */
type angle(pt& a, pt& b){
    type cos = a * b / a.norm() / b.norm();
    return acos(max(-1.0, min(1.0, cos)));
}

/* Devuelve + si c esta izq(ab), - der(ab) y 0 si abc son colineares */
type orient(pt a, pt b, pt c){ return (b - a) ^ (c - a); }

/* Class Line with form v ^ (x, y) = c */
struct line{
    /* Vector director (-b, a) */
    pt v;
    type c;

    line(pt v, type c): v(v), c(c){}
    line(type a, type b, type c): v({-b, a}), c(c){}
    line(pt a, pt b): v(b-a), c(v ^ a){}

    /* Interseccion de dos lineas */
    pt operator & (line l){
        type not_parall = v ^ l.v;

        assert(not_parall);
        return (l.v * c - v * l.c) / not_parall;
    }

    /* Devuelve + si izq, 0 si esta sobre la linea y - der */
    type side(pt p){ return (v ^ p) - c; }

    /* Distancia de un punto a una recta */
    type dist(pt p){ return fabs(side(p)) / v.norm(); }

    /* Distancia Cuadratica de un punto a una recta */
    type dist2(pt p){ return side(p) * side(p) / (type)v.norm2(); }

    /* Compara dos puntos por cual aparece primero en la linea */
    bool cmp_proj(pt a, pt b){ return v * a < v * b; }

    /* Traslada una linea en la direccion del vector t */
    line tranlate(pt t){ return {v, c + (v * t)}; }

    /* Proyeccion de p sobre la recta */
    pt proj(pt p){ return p - v.perp() * side(p) / v.norm2(); }

    /* Reflexion de p sobre la recta */
    pt refl(pt p){ return p - v.perp() * 2 * side(p) / v.norm2(); }
};

/* Stream Overload */
ostream &operator << (ostream &os, const line &l) {
    return os << l.v.y << "x + " << -l.v.x << "y = " << l.c;
}

/* Bisectriz del angulo entre dos vectores */
line bisector(line& l1, line& l2, bool interior){
    assert((l1.v ^ l2.v) != 0);

    type sign = interior ? 1: -1;
    return {l2.v / l2.v.norm() + l1.v / l1.v.norm() * sign,
            l2.c / l2.v.norm() + l1.c / l1.v.norm() * sign };
}

/*
 * Segments
 *  */

/* Cheque si un punto p esta sobre el disco de diametro [ab] */
bool in_disk(pt a, pt b, pt p){
    return (a - p) * (b - p) <= 0;
}

bool on_segment(pt a, pt b, pt p){
    return orient(a, b, p) == 0 && in_disk(a, b, p);
}

/*
 * Polygons
 * */

/* Tests if a polygon is convex */
```

```
bool is_convex(vector<pt>& P){
   bool has_pos = false, has_neg = false;

   for(int i=0, n = P.size(); i < n; i++){
       int o = orient(P[i], P[(i+1)%n], P[(i+2)%n]);
       if( o > 0 ) has_pos = true;
       if( o < 0 ) has_neg = true;
   }

   return !(has_pos && has_neg);
}

/* Area de un Triangulo */
type area_triangle(pt a, pt b, pt c){
    return fabs((b-a) ^ (c-a)) / 2.0;
}

/* Area del Poligono */
type area_poligon(vector<pt>& P){
    type ans = 0.0;

    for(int i=0, n = P.size(); i < n; i++)
            ans += P[i] ^ P[(i+1)%n];
    return fabs(ans) / 2.0;
}

/* Checks if [pq] crosses ray from a */
bool crosses_ray(pt a, pt p, pt q){
    return ((q.y >= a.y) - (p.y >= a.y)) * orient(a, p, q) > 0;
}

/* Tests if a point is inside of a polygon */
bool in_polygon(vector<pt>& P, pt a, bool strict = true){
    int ans = 0;

    for(int i=0, n = P.size(); i < n; i++){
        if( on_segment(P[i], P[(i+1)%n], a) )
            return !strict;
        ans += crosses_ray(a, P[i], P[(i+1)%n]);
    }
    return ans & 1;
}

/*
 * Points Groups
 **/

/* Ordena los puntos por su angulo, desde (-PI, PI] */
void polar_sort(vector<pt>& P, pt o = {0, 0}){
    sort(all(P), [o](pt a, pt b){
        a = a - o, b = b - o;
        assert((a.x != 0 || a.y != 0) && (b.x != 0 || b.y != 0));

        bool as = a.y > 0 || (a.y == 0 && a.x < 0);
        bool bs = b.y > 0 || (b.y == 0 && b.x < 0);

        return as == bs ? 0 < (a ^ b) : as < bs;
    });
}

/* Cicles
 *
 * */

/* Class Circle-Circunference */
struct circle{
    pt o;
    type r;

    /* Constructores Basicos */
    circle(pt o, type r): o(o), r(r){}

    /* Constructor dados tre puntos no colineales */
    circle(pt a, pt b, pt c): o(0, 0){
        b = b - a, c = c - a;

        assert( fabs(b ^ c) > EPS );

        o = a + (b * c.norm2() - c * b.norm2()) / (b ^ c) / 2;
        r = dist(o, a);
    }

    /* Longitud de la Circunferencia */
    type length(){ return 2.0 * PI * r; }

    /* Intersecciones con una recta */
    vector<pt> operator & (line l){
        vector<pt> ans;

        type h2 = r*r - l.dist2(o);

        if( h2 > -EPS ){
            pt p = l.proj(o);
            pt h = l.v * sqrt(h2) / l.v.norm();

            ans.push_back(p + h);
            if( h2 > EPS ) ans.push_back(p - h);
        }
        return ans;
    }
};
```

## 9.2 Par de Puntos Más Cercanos (Sweep Line)

```
/*
    Closest pair of Points
    Sweep-Line Approach
*/

struct point{double x, y;};
struct cmp_x{
    bool operator ()(const point &a, const point &b){
        return a.x < b.x;
    }
};
struct cmp_y{
    bool operator ()(const point &a, const point &b){
        return a.y < b.y;
    }
};

double dist(point a, point b){
    return sqrt( (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y) );
}

int N, last;
double min_dist = 1e10;
point P[MX];
set <point, cmp_y> S;

main(void){

    scanf("%d", &N);
    for(int i=1; i<=N; i++)
```

```
        scanf("%lf%lf", &P[i].x, &P[i].y);

    sort(P+1, P+N+1, cmp_x()), last++;

    for(int i=1; i<=N; i++){
        for( ; last < i && P[i].x - P[last].x >= min_dist; last++)
            S.erase( S.find(P[last]) );

        auto lo = S.lower_bound((point){P[i].x, P[i].y - min_dist});
        auto hi = S.upper_bound((point){P[i].x, P[i].y + min_dist});

        for( ; lo!=hi; lo++)
            min_dist = min(min_dist, dist(*lo, P[i]));

        S.insert(P[i]);
    }

    printf("%.4lf\n", min_dist);

 return 0;
}
```

## 9.3 Union Área (Sweep Line)

```
/** Union Area (Sweep Line)
 *
 *  - DELTA es el intervalo que se le suma a las coordenadas negativas
 **/

const int DELTA = 100000000;

 struct event{
     int x, y1, y2, ty ;

     bool operator < (const event &p)const{
         return x!=p.x ? x < p.x : ty < p.ty;
     }
 };

 int N, last;
 int64 sol;
 vector<event> E;

 struct node{
     int64 x, xend, val, seg;
     node *R, *L;
     node(int64 a, int64 b){
         val = 0, R = NULL, L = NULL, x = a, xend = b, seg = 0;
     }

     void update(int a, int b, int c){
        if(x > b || xend < a)
           return;

        if(L==NULL) L = new node(x, mid);
        if(R==NULL) R = new node(mid+1, xend);

        if(a<=x && xend<=b)
           seg += c;
        else
           L->update(a, b, c), R->update(a, b, c);

        if( !seg )
           val = (x==xend)? 0ll : L->val + R->val;
        else
           val = xend - x + 1ll;
     }
 };

 int main(void){
     node ST = node(0, 2*DELTA);

     scanf("%d", &N);
     for(int i=1, a, b, c, d; i<=N; i++){
        scanf("%d%d%d%d", &a, &b, &c, &d);

        E.pb({a, b-1, d, 1});
        E.pb({c, b-1, d, 0});
     }

     sort(all(E));

     for(auto e: E){
         if( !e.ty ){
             sol += ST.val * (int64)(e.x - last);
             ST.update(e.y2 + DELTA, e.y1 + DELTA, -1);
         }
         else{
            sol += ST.val * (int64)(e.x - last);
            T.update(e.y2 + DELTA, e.y1 + DELTA, 1);
         }
         last =  e.x;
     }

     printf("%lld\n", sol);
```

# 10 Utilidades

## 10.1 Funcion Debug

```
/*
 * Debug function
 *
 * */

#define d(args...) { cerr << "("<< #args << ") --> (" << err(args) << ")\n"
    ; }
string err() {return "";}
template<typename T, typename... Args>
string err(T a, Args... args) { return to_string(a) + " " + err(args...); }
```

## 10.2 Procesar Expresiones Matemáticas

```
/*
 * Procesar Expresiones Matematicas
 *
 * Tested: http://codeforces.com/gym/100676/problem/A
*/

ScriptEngineManager mgr = new ScriptEngineManager();
ScriptEngine engine = mgr.getEngineByName("JavaScript");

Scanner sc = new Scanner(System.in);

while(sc.hasNextLine ())
    System.out.println(engine.eval(sc.nextLine()));
```

## 10.3 Parsear una Línea Completa a Enteros

```
/*
 * Lee una linea y la divide en enteros, guardandolos en el arreglo A
 *
 * */

int A[5];
char S[100000];
char *in;

cin.getline(S, 99, '\n')

cant = 0;
for( in=strtok(S, " "); in; in=strtok(NULL, " ") )
    A[cant++] = atoi(in);
```

## 10.4 Tiempo Real de Ejecición de un Fragmento de Código

```
/* Medir el tiempo real que demora un fragmento de codigo en ejecutarse
 *
 * - La respuesta se da en segundos
 *
 * */

auto _l = std::chrono::high_resolution_clock::now();

/* Aqui se pone el codigo a ser medido */

auto _r = std::chrono::high_resolution_clock::now();

std::chrono::duration<double> diff = _r - _l;
cerr << diff.count() << endl;
```

## 10.5 Línea de Compilación Más Restrictiva

```
g++ -std=c++17 -Wshadow -Wall -o %e %f -fsanitize=address -fsanitize=
    undefined -D_GLIBCXX_DEBUG -g
```

## 10.6 Stack Size

```
/* Aumentar la Memoria de la Pila del Sistema,
 *
 * - func debe ser declarada static (e.g., static void main2)
 *
 * - Tested on: http://coj.uci.cu/24h/problem.xhtml?pid=4150
 */
static void run_with_stack_size(void (*func)(), size_t stsize) {
    char *stack, *send;
    stack = (char *)malloc(stsize);
    send = stack+stsize-16;
    send = (char *)((uintptr_t)send/16*16);

    asm volatile(
        "mov %%rsp, (%0)\n"
        "mov %0, %%rsp\n"
        :
        : "r" (send));
    func();

    asm volatile(
        "mov (%0), %%rsp\n"
        :
        : "r" (send));
    free(stack);
}

void main2(){
    cout << dfs() << endl;
}

int main(){
  run_with_stack_size(main2, 100 * (1<<20));
}
```