



## Contents

<b>1</b>	<b>algorithm</b>	<b>3</b>
<b>2</b>	<b>Estructuras</b>	<b>3</b>
2.1	RMQ (static)	3
2.2	Segment Tree	3
2.2.1	Segment Tree Recursivo	3
2.2.2	ST Iterativo - (Consulta en rango, modificacion a posicion)	4
2.2.3	ST Iterativo - (Consulta a posicion, modificacion en rango)	4
2.2.4	Segment Tree con Punteros	4
2.2.5	Segment Tree 2D	5
2.2.6	Segment Tree Lazy - Suma	6
2.2.7	Segment Tree Lazy - Pintar	6
2.2.8	Segment Tree Persistente	7
2.3	Fenwick Tree	7
2.3.1	Fenwick Tree 2D	7
2.4	Union Find con rank	7
2.5	BigInteger C++	8
2.6	UnorderedSet	12
2.7	Ordered Set	12
2.8	Treap Modo Set	13
2.9	Treap Implicit(Rope)	15
2.10	Treap - Toby and Stones	15
2.11	Convex Hull Trick Estatico	17
2.12	Convex Hull Trick Dinamico	19
2.13	Misof Tree	19
2.14	SQRT Decomposition Basic	20
2.15	Nro. Elementos menores o iguales a $x$ en $O(\log(n))$	21

<b>3</b>	<b>Algos</b>	<b>21</b>
3.1	LIS en $O(n \log n)$ con Reconstruccion	21
3.2	Mo	21
3.3	Ternary Search - Reales	22
3.4	Ternary Search - Enteros	23
<b>4</b>	<b>Strings</b>	<b>23</b>
4.1	Manacher	23
4.2	Trie(estatico)	23
4.3	Suffix Array $O(n \log n)$ con LCP (Kasai) $O(n)$	24
4.4	Minima rotacion lexicografica	24
4.5	Matching	24
4.5.1	KMP	24
4.5.2	Z - Por aprender	24
4.5.3	Matching con hash	24
4.5.4	Matching con suffix array	25
4.5.5	Matching con BWT	25
4.5.6	Matching con Aho-Corasick	26
4.6	Suffix Automaton	26
4.7	K-esima permutacion de una cadena	28
<b>5</b>	<b>Geometria</b>	<b>28</b>
5.1	Cortar Poligono	28
5.2	Interseccion de rectangulos	28
5.3	Otra representacion de rectas y segmentos	29
5.4	Distancia punto-recta	29
5.5	Distancia punto-segmento	30
<b>6</b>	<b>Math</b>	<b>30</b>
6.1	Identidades	30
6.2	Ec. Caracteristica	30
6.3	Identidades de agustin y mario	30
6.4	Combinatorio	30
6.5	Exp. de Numeros Mod.	30
6.6	Matrices de 2x2 y Fibonacci en $\log(n)$	30
6.7	Gauss Jordan	31
6.8	Tridiagonal	31
6.9	Simplex	31
6.10	Matrices y determinante $O(n^3)$	33
6.11	Teorema Chino del Resto	34
6.12	Criba	34
6.13	Funciones de primos	34
6.14	Phollard's Rho (rolando)	35

6.15	GCD . . . . .	36	<b>8 Teoria de juegos</b>	<b>54</b>
6.16	Extended Euclid . . . . .	36	8.1 Teorema fundamental de los juegos optimos . . . . .	54
6.17	LCM . . . . .	36	8.2 Como calcular Grundy . . . . .	54
6.18	Inversos . . . . .	36	<b>9 Algunas formulas de probabilidad</b>	<b>54</b>
6.19	Simpson . . . . .	36	9.1 Regla general de la probabilidad . . . . .	54
6.20	Fraction . . . . .	36	9.2 Teorema de Bayes (Probabilidad condicional) . . . . .	54
6.21	Polinomio . . . . .	37	9.3 Regla de la suma . . . . .	54
6.22	Ec. Lineales . . . . .	37	9.4 Regla de la multiplicacion . . . . .	54
6.23	Karatsuba . . . . .	38	9.5 Esperanza matematica . . . . .	54
6.24	FFT . . . . .	39	9.6 Ley de la esperanza total . . . . .	54
6.25	Tablas y cotas (Primos, Divisores, Factoriales, etc) . . . . .	40	9.7 Esperanza de variables independientes . . . . .	55
<b>7 Grafos</b>		<b>40</b>	9.8 Varianza . . . . .	55
7.1	Bellman-Ford . . . . .	40	9.9 Distribucion Binomial . . . . .	55
7.2	Dijkstra grafos densos . . . . .	41	<b>10 Otros/utilitarios</b>	<b>55</b>
7.3	2 SAT definitivamente no con Tarjan . . . . .	41	10.1 Josephus . . . . .	55
7.4	Prim . . . . .	42	10.2 Josephus $k = 2$ . . . . .	55
7.5	Articulation Points (desgraciadamente Tarjan) . . . . .	42	10.3 Poker . . . . .	55
7.6	Componentes biconexas y puentes (Block Cut Tree) . . . . .	43	10.4 Iterar subconjuntos . . . . .	55
7.7	LCA saltitos potencias de 2 . . . . .	43	10.5 Como reconstruir una DP (normal) . . . . .	55
7.8	LCA sparse table query $O(1)$ . . . . .	43	10.6 Muajaja con $j$ . . . . .	56
7.9	HLD . . . . .	43	10.7 Comparar doubles for noobs . . . . .	56
7.10	Centroid decomposition . . . . .	46	10.8 Infix to postfix . . . . .	56
7.11	Euler cycle . . . . .	46	10.9 Numeros Romanos . . . . .	57
7.12	Diámetro y centro de un árbol . . . . .	46	10.10 Get $k$ -th permutation . . . . .	57
7.13	Algoritmo húngaro . . . . .	47	10.11 Sliding window . . . . .	58
7.14	Union Find dinámico . . . . .	47	10.12 Permutaciones de un dado . . . . .	58
7.15	Truquitos estúpidos por ejemplo second MST es con LCA . . . . .	48	10.13 Ternary search . . . . .	58
7.16	Erdős Gallai . . . . .	48	10.14 Liebre y el tortugo . . . . .	58
7.17	Grafo funcional hallar $k$ -ésimo partiendo de un nodo . . . . .	49	10.15 Como usar printf . . . . .	58
7.18	König . . . . .	49	10.16 Java . . . . .	58
7.19	Min-vertex cover bipartitos . . . . .	49	10.17 Python . . . . .	58
7.20	Max-flow (min cost versión) . . . . .	49	10.18 Template . . . . .	58
7.21	Max-flow corto con matriz . . . . .	50	10.19 File setup . . . . .	58
7.22	Max-flow sin matriz . . . . .	50		
7.23	Dinic . . . . .	51		
7.24	Máximo emparejamiento bipartito . . . . .	52		
7.25	Max-independent set en bipartitos . . . . .	53		
7.26	Min-path cover (ver tópicos raros de Halim) . . . . .	53		
7.27	Min-cost arborescence . . . . .	53		
7.28	Lema de diapositivas de Nico de grafos funcionales . . . . .	53		
7.29	Minimax y minimax con Kruskal y Dijkstra . . . . .	53		

## 1 algorithm

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	void ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	void llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	it al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	bool esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	it encuentra i $\in [f, l)$ tq. i=elem, pred(i), $i \in [f2, l2)$
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca $[f2, l2) \in [f, l)$
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	it min, max de [f, l)
lexicographical_compare	f1, l1, f2, l2	bool con $[f1, l1) \leq [f2, l2)$
next/prev_permutation	f, l	deja en [f, l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f, l), hace un heap de [f, l)
is_heap	f, l	bool es [f, l) un heap
accumulate	f, l, i, [op]	$T = \sum / \text{oper de } [f, l)$
inner_product	f1, l1, f2, i	$T = i + [f1, l1) \cdot [f2, \dots)$
partial_sum	f, l, r, [op]	$r+i = \sum / \text{oper de } [f, f+i) \forall i \in [f, l)$
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

## 2 Estructuras

### 2.1 RMQ (static)

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion:  $LVL \geq \text{ceil}(\log n)$ ; Usar [ ] para llenar arreglo y luego build().

```

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

```

### 2.2 Segment Tree

#### 2.2.1 Segment Tree Recursivo

```

1 //inclusive segment tree [L,R]
2 int T[4 * N];
3 void init(int node = 1, int l = 0, int r = n - 1){
4     if(l == r) T[node] = v[l];
5     else{
6         int mid = (l + r) >> 1;
7         init(2 * node, l, mid);
8         init(2 * node + 1, mid + 1, r);
9         T[node] = op(T[2 * node], T[2 * node + 1]);
10    }
11 }
12 void update(int pos, int val, int node = 1, int l = 0, int r = n - 1){
13     if(r < pos || l > pos) return;
14     if(l == r) T[node] = val;
15     else{
16         int mid = (l + r) >> 1;
17         update(pos, val, 2 * node, l, mid);
18         update(pos, val, 2 * node + 1, mid + 1, r);
19         T[node] = op(T[2 * node], T[2 * node + 1]);

```

```

20 }
21 }
22 int query(int x,int y,int node = 1,int l = 0,int r = n - 1){
23     if(r < x || l > y)return NEUTRO;
24     if(x <= l && r <= y)return T[node];
25     else{
26         int mid = (l + r) >> 1;
27         return op(query(x,y,2 * node,l,mid),query(x,y,2 * node + 1,mid + 1,r
28             ));
29     }
30 }

```

### 2.2.2 ST Iterativo - (Consulta en rango, modificacion a posicion)

```

1 //Segment tree iterative [l,r)
2 int T[2 * N];
3 void init(){
4     for(int i = n; i < 2 * n;i++)T[i] = val[i];
5     for(int i = n - 1; i >= 1; i--)T[i] = op(T[i << 1],T[i << 1 | 1]);
6 }
7 int op(int a,int b){
8     //an associative function
9     return a + b;
10 }
11 void update(int pos,int u){
12     pos += n;
13     for(pos >= 1; pos >= 1; pos >= 1)T[pos] = op(T[pos << 1],T[pos << 1
14         | 1]);
15 }
16 int query(int l,int r){
17     l += n, r += n;
18     int ans = NEUTRO;
19     while(l < r){
20         if(l & 1)ans = op(ans,T[l++]);
21         if(r & 1)ans = op(ans,T[--r]);
22         l >>= 1,r >>= 1;
23     }
24     return ans;
25 }

```

### 2.2.3 ST Iterativo - (Consulta a posicion, modificacion en rango)

```

1 /*Segment Tree modificar un rango, acceder a una posicion

```

```

2     solo sirve cuando la operacion que realizamos es conmutativa
3     por ejemplo la suma, pero no funciona con la asignacion
4 */
5 //adiciona value al rango [l, r)
6 void modify(int l, int r, int value) { // rango [l, r)
7     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
8         if (l&1) t[l++] += value;
9         if (r&1) t[--r] += value;
10    }
11 }
12 //acceder a la posicion
13 int query(int p) {
14     int res = 0;
15     for (p += n; p > 0; p >>= 1) res += t[p];
16     return res;
17 }
18 //Si necesitamos actualizar todo lo podemos hacer en O(n)
19 //Y luego acceder a las hojas en O(1)
20 void push() {
21     for (int i = 1; i < n; ++i) {
22         t[i<<1] += t[i];
23         t[i<<1|1] += t[i];
24         t[i] = 0;
25     }
26 }

```

### 2.2.4 Segment Tree con Punteros

```

1 /*La creacion y las queries son [0, n)
2 [cerrado, abierto)
3 Por alguna razon el destructor hace que se borre todo
4 inmediatamente despues de que termina el constructor*/
5 const int maxn = 1000000;
6 tipo v[maxn];
7
8 const tipo NEUTRO = PONGA_AQUI_EL_NEUTRO;
9 tipo oper(tipo a, tipo b) {
10     return min(a, b);
11 }
12 struct segment_tree {
13     segment_tree *L, *R;
14     int l, r;
15     tipo value;

```

```

16 segment_tree(): L(nullptr), R(nullptr), value(NEUTRO) {}
17 segment_tree(int _l, int _r) : l(_l), r(_r) {
18     if (l + 1 == r) {
19         value = v[l];
20     } else {
21         int mid = (l + r) >> 1;
22         L = new segment_tree(l, mid);
23         R = new segment_tree(mid, r);
24         value = oper(L->value, R->value);
25     }
26 }
27 //~segment_tree() {delete L; delete R;}//NO PONER!!! ERROR!!!
28 void update(int pos, int val) {
29     if (r <= pos || l > pos) return;
30     if (l + 1 == r) {
31         v[pos] = value = val;
32     } else {
33         L->update(pos, val);
34         R->update(pos, val);
35         value = oper(L->value, R->value);
36     }
37 }
38 tipo query(int a, int b) {
39     if (a <= l && r <= b) return value;
40     if (l >= b || r <= a) return NEUTRO;
41     return oper(L->query(a, b), R->query(a, b));
42 }
43 } tree;

```

### 2.2.5 Segment Tree 2D

```

1 typedef long long ll;
2 struct segmetree{
3     int n;
4     vector<ll>T;
5     segmetree(){n = 0;}
6     segmetree(int _){
7         n = _;
8         T.resize(2 * n + 1);
9     }
10    void rupdate(int pos,int value){
11        pos += n;
12        T[pos] = value;

```

```

13        for(pos >>= 1; pos >= 1; pos >>= 1)
14            T[pos] = T[pos << 1] + T[pos << 1 | 1];
15    }
16    void update(int pos,int value){
17        pos += n;
18        T[pos] += value;
19        for(pos >>= 1; pos >= 1; pos >>= 1)
20            T[pos] = T[pos << 1] + T[pos << 1 | 1];
21    }
22    int query(int l,int r){
23        l += n;r+= n;
24        int ans = 0;
25        while(l < r){
26            if(l & 1)ans += T[l++];
27            if(r & 1)ans += T[--r];
28            l >>= 1, r >>= 1;
29        }
30        return ans;
31    }
32 };
33 struct st{
34     int n;
35     vector<segmetree>T;
36     st(){
37         st(int _){
38             n = _;
39             for(int i = 0;i < 2 * n;i++){
40                 T.push_back(segmetree(n));
41             }
42         }
43     void update(int x,int y,int val){
44         x += n;
45         T[x].update(y,val);
46         segmetree ok;
47         for(x >>= 1; x >= 1; x >>= 1){
48             T[x].rupdate(y,T[x << 1].query(y,y + 1));
49             T[x].update(y,T[x << 1 | 1].query(y,y + 1));
50         }
51     }
52     ll query(int l,int b,int r,int t){
53         l += n;
54         r += n;
55         r++,t++;

```

```

56     ll ans = 0LL;
57     while(l < r){
58         if(l & 1)ans += T[l++].query(b,t);
59         if(r & 1)ans += T[--r].query(b,t);
60         l >>= 1, r >>= 1;
61     }
62     return ans;
63 }
64 };

```

### 2.2.6 Segment Tree Lazy - Suma

```

1 //Todo es [l, r)
2 void build(int id = 1,int l = 0,int r = n){
3     if(r - l < 2){ // l + 1 == r
4         s[id] = a[l];
5         return ;
6     }
7     int mid = (l+r)/2;
8     build(id * 2, l, mid);
9     build(id * 2 + 1, mid, r);
10    s[id] = s[id * 2] + s[id * 2 + 1];
11 }
12 //aqui poner los arrays lazy and id
13 void upd(int id,int l,int r,int x){// increase all members in this
14     interval by x
15     lazy[id] += x;
16     s[id] += (r - l) * x;
17 }
18 //A function to pass the update information to its children :
19 void shift(int id,int l,int r){//pass update information to the children
20     int mid = (l+r)/2;
21     upd(id * 2, l, mid, lazy[id]);
22     upd(id * 2 + 1, mid, r, lazy[id]);
23     lazy[id] = 0;// passing is done
24 }
25 //A function to perform increase queries :
26 void increase(int x,int y,int v,int id = 1,int l = 0,int r = n){
27     if(x >= r or l >= y) return ;
28     if(x <= l && r <= y){
29         upd(id, l, r, v);
30         return ;
31     }

```

```

31     shift(id, l, r);
32     int mid = (l+r)/2;
33     increase(x, y, v, id * 2, l, mid);
34     increase(x, y, v, id*2+1, mid, r);
35     s[id] = s[id * 2] + s[id * 2 + 1];
36 }
37 //(We should call increase(l r x))
38 int sum(int x,int y,int id = 1,int l = 0,int r = n){
39     if(x >= r or l >= y) return 0;
40     if(x <= l && r <= y) return s[id];
41     shift(id, l, r);
42     int mid = (l+r)/2;
43     return sum(x, y, id * 2, l, mid) +
44         sum(x, y, id * 2 + 1, mid, r);
45 }

```

### 2.2.7 Segment Tree Lazy - Pintar

```

1 void shift(int id){
2     if(lazy[id])
3         lazy[2 * id] = lazy[2 * id + 1] = lazy[id];
4     lazy[id] = 0;
5 }
6 //color > 1, por que se usa el 0 para decir que no hay lazy
7 void upd(int x,int y,int color, int id = 0,int l = 0,int r = n){//
8     painting the interval [x,y) with color "color"
9     if(x >= r or l >= y) return ;
10    if(x <= l && r <= y){
11        lazy[id] = color;
12        return ;
13    }
14    int mid = (l+r)/2;
15    shift(id);
16    upd(x, y, color, 2 * id, l, mid);
17    upd(x, y, color, 2*id+1, mid, r);
18 }
19 set <int> se;
20 void cnt(int id = 1,int l = 0,int r = n){
21     if(lazy[id]){
22         se.insert(lazy[id]);
23         return ; // there is no need to see the children, because all the
24         interval is from the same color

```

```

24 }
25 if(r - l < 2) return ;
26 int mid = (l+r)/2;
27 cnt(2 * id, l, mid);
28 cnt(2*id+1, mid, r);
29 }

```

### 2.2.8 Segment Tree Persistente

```

1 int segcnt = 0;
2 struct segment {
3     int l, r, lid, rid, sum;
4 } segs[2000000];
5 int build(int l, int r) {
6     if (l > r) return -1;
7     int id = segcnt++;
8     segs[id].l = l;
9     segs[id].r = r;
10    if (l == r) segs[id].lid = -1, segs[id].rid = -1;
11    else {
12        int m = (l + r) / 2;
13        segs[id].lid = build(l, m);
14        segs[id].rid = build(m + 1, r);
15        segs[id].sum = 0;
16        return id;
17    }
18    int update(int idx, int v, int id) {
19        if (id == -1) return -1;
20        if (idx < segs[id].l || idx > segs[id].r) return id;
21        int nid = segcnt++;
22        segs[nid].l = segs[id].l;
23        segs[nid].r = segs[id].r;
24        segs[nid].lid = update(idx, v, segs[id].lid);
25        segs[nid].rid = update(idx, v, segs[id].rid);
26        segs[nid].sum = segs[id].sum + v;
27        return nid;
28    }
29    int query(int id, int l, int r) {
30        if (r < segs[id].l || segs[id].r < l) return 0;
31        if (l <= segs[id].l && segs[id].r <= r) return segs[id].sum;
32        return query(segs[id].lid, l, r) + query(segs[id].rid, l, r);
33    }

```

### 2.3 Fenwick Tree

```

1 vector<tipo> tree;
2 int maxn;

```

```

3 void init(int n) {
4     tree = vector<tipo>(n, 0);
5     maxn = n;
6 }
7 void add(int i, tipo k) { //i valid [1, n]
8     for(; i < maxn; i += i&-i) tree[i] += k;
9 }
10
11 tipo get(int i){//returns sum [1, i]
12     tipo s = 0;
13     for(; i > 0; i-=i&-i) s+=tree[i];
14     return s;
15 }

```

### 2.3.1 Fenwick Tree 2D

```

1
2 ll T[1025][1025];
3 int n;
4
5 ll query(int x, int y)
6 {
7     ll res = 0;
8     for(int i = x; i >= 0; i = (i & (i+1)) - 1)
9         for(int j = y; j >= 0; j = (j & (j+1)) - 1)
10             res += T[i][j];
11     return res;
12 }
13
14 void update(int x, int y, int val)
15 {
16     for(int i = x; i < n; i = (i | (i+1)))
17         for(int j = y; j < n; j = (j | (j+1)))
18             T[i][j] += val;
19 }

```

### 2.4 Union Find con rank

```

1 /*===== <Union find rangos> =====
2 Complexity: O(N)
3 index 0 to n - 1 warning
4 Complexity O(N)
5 */
6 #define MAX_INSERTE_VALOR_AQUI

```

```

7  int padre[MAX];
8  int rango[MAX];
9  void MakeSet(int n){
10     for (int i = 0 ; i < n ; ++i) {
11         padre[i] = i; rango[i] = 0; }
12 }
13 int Find(int x) {
14     if(x == padre[x])
15         return x;
16     return padre[x] = Find(padre[x]);
17 }
18 void UnionbyRank(int x , int y){
19     int xRoot = Find(x);
20     int yRoot = Find(y);
21     //el padre de ambas componentes sera el de mayor altura
22     if(rango[xRoot] > rango[yRoot])//X tiene mas altura que Y
23         padre[yRoot] = xRoot;
24     }else{//Y >= X
25         padre[xRoot] = yRoot;
26         if(rango[xRoot] == rango[yRoot])//si poseen la misma altura
27             rango[yRoot]++;//incremento el rango de la nueva raiz
28     }
29 }

```

## 2.5 BigInteger C++

```

1  // g++ -std=c++11 "bigint.cpp" -o run
2  /**
3  ===== <Big Int c++ version> =====
4  Contain a useful big int, overload all operators, including cin, cout,
5  comparator, build via string (prefer this metod) or long long, for now
6  this not have a
7  to_string method
8  Problem for practice: UVA 494
9  */
10 // base and base_digits must be consistent
11 const int base = 1000000000;
12 const int base_digits = 9;
13 struct bigint {
14     vector<int> a;
15     int sign;
16

```

```

17     bigint() :
18         sign(1) {
19     }
20
21     bigint(long long v) {
22         *this = v;
23     }
24
25     bigint(const string &s) {
26         read(s);
27     }
28
29     void operator=(const bigint &v) {
30         sign = v.sign;
31         a = v.a;
32     }
33
34     void operator=(long long v) {
35         sign = 1;
36         if (v < 0)
37             sign = -1, v = -v;
38         for (; v > 0; v = v / base)
39             a.push_back(v % base);
40     }
41
42     bigint operator+(const bigint &v) const {
43         if (sign == v.sign) {
44             bigint res = v;
45
46             for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) || carry; ++i) {
47                 if (i == (int) res.a.size())
48                     res.a.push_back(0);
49                 res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
50                 carry = res.a[i] >= base;
51                 if (carry)
52                     res.a[i] -= base;
53             }
54             return res;
55         }
56         return *this - (-v);
57     }
58 }

```



```

59     bigint operator-(const bigint &v) const {
60         if (sign == v.sign) {
61             if (abs() >= v.abs()) {
62                 bigint res = *this;
63                 for (int i = 0, carry = 0; i < (int) v.a.size() || carry
64                     ; ++i) {
65                     res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] :
66                         0);
67                     carry = res.a[i] < 0;
68                     if (carry)
69                         res.a[i] += base;
70                 }
71                 res.trim();
72                 return res;
73             }
74             return -(v - *this);
75         }
76     }
77     void operator*=(int v) {
78         if (v < 0)
79             sign = -sign, v = -v;
80         for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
81             if (i == (int) a.size())
82                 a.push_back(0);
83             long long cur = a[i] * (long long) v + carry;
84             carry = (int) (cur / base);
85             a[i] = (int) (cur % base);
86             //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"
87                 "(base));
88         }
89         trim();
90     }
91     bigint operator*(int v) const {
92         bigint res = *this;
93         res *= v;
94         return res;
95     }
96
97     friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &
    b1) {

```

```

98         int norm = base / (b1.a.back() + 1);
99         bigint a = a1.abs() * norm;
100        bigint b = b1.abs() * norm;
101        bigint q, r;
102        q.a.resize(a.a.size());
103
104        for (int i = a.a.size() - 1; i >= 0; i--) {
105            r *= base;
106            r += a.a[i];
107            int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
108            int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() -
109                1];
110            int d = ((long long) base * s1 + s2) / b.a.back();
111            r -= b * d;
112            while (r < 0)
113                r += b, --d;
114            q.a[i] = d;
115        }
116
117        q.sign = a1.sign * b1.sign;
118        r.sign = a1.sign;
119        q.trim();
120        r.trim();
121        return make_pair(q, r / norm);
122    }
123
124    bigint operator/(const bigint &v) const {
125        return divmod(*this, v).first;
126    }
127
128    bigint operator%(const bigint &v) const {
129        return divmod(*this, v).second;
130    }
131
132    void operator/=(int v) {
133        if (v < 0)
134            sign = -sign, v = -v;
135        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
136            long long cur = a[i] + rem * (long long) base;
137            a[i] = (int) (cur / v);
138            rem = (int) (cur % v);
139        }
140        trim();

```

```

140     }
141
142     bigint operator/(int v) const {
143         bigint res = *this;
144         res /= v;
145         return res;
146     }
147
148     int operator%(int v) const {
149         if (v < 0)
150             v = -v;
151         int m = 0;
152         for (int i = a.size() - 1; i >= 0; --i)
153             m = (a[i] + m * (long long) base) % v;
154         return m * sign;
155     }
156
157     void operator+=(const bigint &v) {
158         *this = *this + v;
159     }
160     void operator-=(const bigint &v) {
161         *this = *this - v;
162     }
163     void operator*=(const bigint &v) {
164         *this = *this * v;
165     }
166     void operator/=(const bigint &v) {
167         *this = *this / v;
168     }
169
170     bool operator<(const bigint &v) const {
171         if (sign != v.sign)
172             return sign < v.sign;
173         if (a.size() != v.a.size())
174             return a.size() * sign < v.a.size() * v.sign;
175         for (int i = a.size() - 1; i >= 0; i--)
176             if (a[i] != v.a[i])
177                 return a[i] * sign < v.a[i] * v.sign;
178         return false;
179     }
180
181     bool operator>(const bigint &v) const {
182         return v < *this;

```

```

183     }
184     bool operator<=(const bigint &v) const {
185         return !(v < *this);
186     }
187     bool operator>=(const bigint &v) const {
188         return !(*this < v);
189     }
190     bool operator==(const bigint &v) const {
191         return !(*this < v) && !(v < *this);
192     }
193     bool operator!=(const bigint &v) const {
194         return *this < v || v < *this;
195     }
196
197     void trim() {
198         while (!a.empty() && !a.back())
199             a.pop_back();
200         if (a.empty())
201             sign = 1;
202     }
203
204     bool isZero() const {
205         return a.empty() || (a.size() == 1 && !a[0]);
206     }
207
208     bigint operator-() const {
209         bigint res = *this;
210         res.sign = -sign;
211         return res;
212     }
213
214     bigint abs() const {
215         bigint res = *this;
216         res.sign *= res.sign;
217         return res;
218     }
219
220     long long longValue() const {
221         long long res = 0;
222         for (int i = a.size() - 1; i >= 0; i--)
223             res = res * base + a[i];
224         return res * sign;
225     }

```

```

226
227 friend bigint gcd(const bigint &a, const bigint &b) {
228     return b.isZero() ? a : gcd(b, a % b);
229 }
230 friend bigint lcm(const bigint &a, const bigint &b) {
231     return a / gcd(a, b) * b;
232 }
233
234 void read(const string &s) {
235     sign = 1;
236     a.clear();
237     int pos = 0;
238     while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+'))
239     {
240         if (s[pos] == '-')
241             sign = -sign;
242         ++pos;
243     }
244     for (int i = s.size() - 1; i >= pos; i -= base_digits) {
245         int x = 0;
246         for (int j = max(pos, i - base_digits + 1); j <= i; j++)
247             x = x * 10 + s[j] - '0';
248         a.push_back(x);
249     }
250     trim();
251 }
252
253 friend istream& operator>>(istream &stream, bigint &v) {
254     string s;
255     stream >> s;
256     v.read(s);
257     return stream;
258 }
259
260 friend ostream& operator<<(ostream &stream, const bigint &v) {
261     if (v.sign == -1)
262         stream << '-';
263     stream << (v.a.empty() ? 0 : v.a.back());
264     for (int i = (int) v.a.size() - 2; i >= 0; --i)
265         stream << setw(base_digits) << setfill('0') << v.a[i];
266     return stream;
267 }

```

```

268 static vector<int> convert_base(const vector<int> &a, int old_digits
269     , int new_digits) {
270     vector<long long> p(max(old_digits, new_digits) + 1);
271     p[0] = 1;
272     for (int i = 1; i < (int) p.size(); i++)
273         p[i] = p[i - 1] * 10;
274     vector<int> res;
275     long long cur = 0;
276     int cur_digits = 0;
277     for (int i = 0; i < (int) a.size(); i++) {
278         cur += a[i] * p[cur_digits];
279         cur_digits += old_digits;
280         while (cur_digits >= new_digits) {
281             res.push_back(int(cur % p[new_digits]));
282             cur /= p[new_digits];
283             cur_digits -= new_digits;
284         }
285     }
286     res.push_back((int) cur);
287     while (!res.empty() && !res.back())
288         res.pop_back();
289     return res;
290 }
291
292 typedef vector<long long> vll;
293
294 static vll karatsubaMultiply(const vll &a, const vll &b) {
295     int n = a.size();
296     vll res(n + n);
297     if (n <= 32) {
298         for (int i = 0; i < n; i++)
299             for (int j = 0; j < n; j++)
300                 res[i + j] += a[i] * b[j];
301         return res;
302     }
303
304     int k = n >> 1;
305     vll a1(a.begin(), a.begin() + k);
306     vll a2(a.begin() + k, a.end());
307     vll b1(b.begin(), b.begin() + k);
308     vll b2(b.begin() + k, b.end());
309
310     vll a1b1 = karatsubaMultiply(a1, b1);

```

[illegible]

```

1 //Compiler: g++ --std=c++11
2 struct Hash{
3     size_t operator()(const ii &a)const{
4         size_t s=hash<int>()(a.fst);
5         return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
6     }
7     size_t operator()(const vector<int> &v)const{
8         size_t s=0;
9         for(auto &e : v)
10             s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
11         return s;
12     }
13 };
14 unordered_set<ii, Hash> s;
15 unordered_map<ii, int, Hash> m; //map<key, value, hasher>

```

```
1  /*
2   A brief explanation about use of a powerful library: orderd_set
3   Reference link: http://codeforces.com/blog/entry/11080
4   and a hash for the type pair
5   */
6  #include <ext/pb_ds/assoc_container.hpp>
7  #include <ext/pb_ds/tree_policy.hpp>
```

```

8 using namespace __gnu_pbds;
9 /*typedef tree<int,null_type,less<int>,rb_tree_tag,
   tree_order_statistics_node_update> ordered_set;
10 If we want to get map but not the set, as the second argument type must
   be used mapped type. Apparently, the tree supports the same
   operations as the set (at least I haven't any problems with them
   before), but also there are two new features - it is find_by_order
   () and order_of_key().
11 The first returns an iterator to the k-th largest element (counting from
   zero), the second - the number of items
12 in a set that are strictly smaller than our item. Example of use:*/
13 template <typename T>
14 using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
   tree_order_statistics_node_update>;
15 int main(){
16     ordered_set<int> s;
17     s.insert(1);
18     s.insert(3);
19     cout << *s.find_by_order(0) << endl; // print the 0-th smallest number
   in s(0-based)
20     cout << s.order_of_key(2) << endl; // the number of elements in the s
   less than 2
21     //find_by_order(i) devuelve iterador al i-esimo elemento
22     //order_of_key(k): devuelve la pos del lower bound de k
23     //Ej: 12, 100, 505, 1000, 10000.
24     //order_of_key(10) == 0, order_of_key(100) == 1,
25     //order_of_key(707) == 3, order_of_key(9999999) == 5
26     return 0;
27 }

```

## 2.8 Treap Modo Set

```

1 struct treap {
2     typedef struct _node {
3         int x, y, cnt;
4         _node *l, *r;
5         _node(int _x) : x(_x), y((rand() << 16) ^ rand()), cnt(1), l(nullptr)
   , r(nullptr) {}
6         ~_node() {delete l; delete r;}
7         void recalc() {
8             cnt = 1;
9             if (l) cnt += l->cnt;
10            if (r) cnt += r->cnt;

```

```

11    }
12    } *node;
13    treap(): root(nullptr) {}
14    ~treap() {delete root;}
15    node root;
16    /*Divide el arbol que tiene como raiz a "t", guarda en
17       L todos los nodos con key menor o igual a "x"
18       R todos los nodos con key mayor estricto a x
19       t es destruido/modificado
20    */
21    void split(node t, int x, node &L, node &R) {
22        if (t == nullptr) {L = R = nullptr; return;}
23        if (t->x <= x) {
24            split(t->r, x, t->r, R);
25            L = t;
26        } else {
27            split(t->l, x, L, t->l);
28            R = t;
29        }
30        t->recalc();
31    }
32    /*Une los dos nodos L y R en un solo arbol y los devuelve.
33       L y R son modificados
34    */
35    node merge(node L, node R) {
36        if (L == nullptr) return R;
37        if (R == nullptr) return L;
38        if (L->y > R->y) {
39            L->r = merge(L->r, R);
40            L->recalc();
41            return L;
42        } else {
43            R->l = merge(L, R->l);
44            R->recalc();
45            return R;
46        }
47    }
48    /*Inserta un solo nodo con key igual a "x"*/
49    void insert(int x) {
50        //verificar que no se inserten elementos repetidos
51        //pueden ocasionar que ya no cumplan la propiedad de BST
52        //pueden haber keys iguales a izquierda y derecha
53        node L, R;

```

```

54     split(root, x, L, R);
55     root = merge(merge(L, new _node(x)), R);
56 }
57 /*Borra todos los nodos con key igual a "x"
58    No pasa nada si no hay nodos con key igual a "x"*/
59 void erase(int x) {
60     node L, m, R;
61     split(root, x, L, R);
62     split(L, x - 1, L, m);
63     root = merge(L, R);
64 }
65 int count(int x) {
66     node L, m, R;
67     split(root, x, L, R);
68     split(L, x - 1, L, m);
69     int ans = cnt(m);
70     root = merge(merge(L, m), R);
71     return ans;
72 }
73 /*Borra un nodo con key igual a "x"
74    No sucede nada si no hay nodo con ese key*/
75 void erase_one(int x) {
76     node L, m, R;
77     split(root, x, L, R);
78     split(L, x - 1, L, m);
79     if (m)
80         m = merge(m->l, m->r);
81     root = merge(merge(L, m), R);
82 }
83 /*k-esimo indexado desde 0
84    devuelve INT_MAX si no hay k-esimo*/
85 int kthElement(int k) {
86     node cur = root;
87     while (cur != nullptr) {
88         int sizeLeft = cnt(cur->l);
89         if (sizeLeft == k)
90             return cur->x;
91         cur = sizeLeft > k ? cur->l : cur->r;
92         if (sizeLeft < k)
93             k -= (sizeLeft + 1);
94     }
95     return INT_MAX;
96 }

```

```

97     /*Numero de elementos con keys menores a "x"*/
98     int less(int x) {
99         node cur = root;
100         int ans = 0;
101         while (cur != nullptr) {
102             if (cur->x >= x) {
103                 cur = cur->l;
104             } else {
105                 ans += cnt(cur->l) + 1;
106                 cur = cur->r;
107             }
108         }
109         return ans;
110     }
111     int cnt(node t) const {
112         return t ? t->cnt : 0;
113     }
114     int size() const {
115         return cnt(root);
116     }
117     /*//from e-maxx.ru son mas rapidos(no x mucho) pero menos entendibles
118     //solo borra un elemento
119     void erase(node &t, int x) {
120         if (t->x == x)
121             t = merge(t->l, t->r);
122         else
123             erase(x < t->x?t->l:t->r, x);
124         if (t)
125             t->recalc();
126     }
127     void erase_one(int x) {erase(root, x);}
128     void insert(node &t, node it) {
129         if (t == nullptr)
130             t = it;
131         else if (it->y > t->y)
132             split(t, it->x, it->l, it->r), t = it;
133         else
134             insert(it->x < t->x? t->l: t->r, it);
135         t->recalc();
136     }
137     void insert(int x) {insert(root, new _node(x));}*/
138 };

```

## 2.9 Treap Implicito(Rope)

```

1 struct rope {
2     typedef struct _node {
3         int value, y, cnt;
4         _node *l, *r;
5         _node(int _value) : value(_value), y((rand() << 16) ^ rand()), cnt
6             (1), l(nullptr), r(nullptr) {}
7         ~_node() {delete l; delete r;}
8         void recalc() {
9             cnt = 1;
10            if (l) cnt += l->cnt;
11            if (r) cnt += r->cnt;
12        }
13    } *node;
14    rope(): root(nullptr) {}
15    ~rope() {delete root;}
16    node root;
17    /*Divide el arbol que tiene como raiz a "t", guarda en
18    L los primeros "x" elementos del array
19    R el primer elemento de R es el elemento en posicion x(indexado desde
20    0) del array
21    t es destruido/modificado
22    L = [0, x)
23    R = [x, n) */
24    void split(node t, int x, node &L, node &R) {
25        if (t == nullptr) {L = R = nullptr; return;}
26        int curIndex = cnt(t->l) + 1;
27        if (curIndex <= x) {
28            split(t->r, x - curIndex, t->r, R);
29            L = t;
30        } else {
31            split(t->l, x, L, t->l);
32            R = t;
33        }
34        t->recalc();
35    }
36    /* Une los dos nodos L y R en un solo arbol y los devuelve.
37    L y R son modificados */
38    node merge(node L, node R) {
39        if (L == nullptr) return R;
40        if (R == nullptr) return L;
41        if (L->y > R->y) {

```

```

42            L->r = merge(L->r, R);
43            L->recalc();
44            return L;
45        } else {
46            R->l = merge(L, R->l);
47            R->recalc();
48            return R;
49        }
50    }
51    /*Inserta "value" en la posicion "pos"(indexado desde 0) recorre todos
52    los elementos a la derecha desde la posicion pos*/
53    void insert(int pos, int value) {
54        node L, R;
55        split(root, pos, L, R); //en R esta pos
56        root = merge(merge(L, new _node(value)), R);
57    }
58    /*Borra el elemento en posicion pos*/
59    void erase(int pos) {
60        node L, m, R;
61        split(root, pos, L, R);
62        split(R, 1, m, R);
63        root = merge(L, R);
64    }
65    int cnt(node t) const {
66        return t ? t->cnt : 0;
67    }
68    int size() const {
69        return cnt(root);
70    }
71    int main() {srand(time(nullptr)); return 0;}

```

## 2.10 Treap - Toby and Stones

```

1 const int PAINT = 0, FLIP = 1, REVERSE = 2;
2 struct rope {
3     typedef struct _node {
4         int value, y, cnt;
5         int negros;
6         bool rev;
7         bool lazy_flip;
8         int lazy_pintar;
9         _node *l, *r;

```

```

10  _node(int _value) : value(_value), y((rand() << 16) ^ rand()), cnt
    (1), l(nullptr), r(nullptr) {
11      negros = _value;
12      rev = false;
13      lazy_flip = false;
14      lazy_pintar = -1;
15  }
16  ~_node() {delete l; delete r;}
17  void recalc() {
18      cnt = 1;
19      negros = value;
20      if (l) cnt += l->cnt, negros += l->negros;
21      if (r) cnt += r->cnt, negros += r->negros;
22  }
23  void push_lazy(int type, int param = -1) {
24      if (type == PAINT) {
25          rev = false;
26          lazy_flip = false;
27          lazy_pintar = param;
28          negros = param * cnt;
29      }
30      if (type == FLIP) {
31          if (lazy_pintar != -1) {
32              assert(rev == false && lazy_flip == false);
33              lazy_pintar = (1 - lazy_pintar);
34              negros = lazy_pintar * cnt;
35          } else {
36              lazy_flip ^= true;
37              negros = cnt - negros;
38          }
39      }
40      if (type == REVERSE) {
41          if (lazy_pintar == -1) {
42              rev ^= true;
43          }
44      }
45  }
46  void verify() {
47      bool op1 = (lazy_pintar != -1);
48      bool op2 = (rev || lazy_flip);
49      //solo puede pasar uno de los 2 casos, o ninguno esta activado o
        solo uno de los dos
50      assert((!op1 and !op2) or (op1 xor op2));

```

```

51  }
52  string to_string() {
53      stringstream ss;
54      ss << "value=" << value << ",negros=" << negros << ",cnt=" << cnt
        << "(rev=" << rev << ", lazy_flip=" << lazy_flip << ")";
55      return ss.str();
56  }
57  } *node;
58  rope(): root(nullptr) {}
59  ~rope() {delete root;}
60  node root;
61  void push(node &t) {
62      if (t == nullptr) return;
63      bool op1 = ((t->lazy_pintar) != -1);
64      bool op2 = ((t->rev) || (t->lazy_flip));
65      //solo puede pasar uno de los 2 casos, o ninguno esta activado o
        solo uno de los dos
66      assert((!op1 and !op2) or (op1 xor op2));
67      if (op1) {
68          t->value = t->lazy_pintar;
69          if (t->l) t->l->push_lazy(PAINT, t->lazy_pintar);
70          if (t->r) t->r->push_lazy(PAINT, t->lazy_pintar);
71          t->lazy_pintar = -1;
72      }
73      if (op2) {//no importa el orden en que se aplique estos 2
        //reverse
74          if (t->rev) {
75              swap(t->l, t->r);
76              t->rev = false;
77              if (t->l) t->l->push_lazy(REVERSE);
78              if (t->r) t->r->push_lazy(REVERSE);
79          }
80          //invertir colores
81          if (t->lazy_flip) {
82              t->value = 1 - (t->value);
83              t->lazy_flip = false;
84              if (t->l) t->l->push_lazy(FLIP);
85              if (t->r) t->r->push_lazy(FLIP);
86          }
87      }
88  }
89  }
90  void split(node t, int x, node &L, node &R) {
91      push(t);

```



```

92     if (t == nullptr) {L = R = nullptr; return;}
93     int curIndex = cnt(t->l) + 1;
94     if (curIndex <= x) {
95         split(t->r, x - curIndex, t->r, R);
96         L = t;
97     } else {
98         split(t->l, x, L, t->l);
99         R = t;
100    }
101    t->recalc();
102 }
103 node merge(node L, node R) {
104     push(L); push(R);
105     if (L == nullptr) return R;
106     if (R == nullptr) return L;
107     if (L->y > R->y) {
108         L->r = merge(L->r, R);
109         L->recalc();
110         return L;
111     } else {
112         R->l = merge(L, R->l);
113         R->recalc();
114         return R;
115     }
116 }
117 void insert(int pos, int value) {
118     node L, R;
119     split(root, pos, L, R); //en R esta pos
120     root = merge(merge(L, new _node(value)), R);
121 }
122 int cnt(node t) {
123     return t ? t->cnt : 0;
124 }
125 int size() {
126     return cnt(root);
127 }
128 void reverse(int i, int j) {
129     node L, m, R;
130     split(root, i, L, R);
131     split(R, j - i + 1, m, R);
132     m->push_lazy(REVERSE);
133     root = merge(merge(L, m), R);
134 }

```

```

135 void flip(int i, int j) {
136     node L, m, R;
137     split(root, i, L, R);
138     split(R, j - i + 1, m, R);
139     m->push_lazy(FLIP);
140     root = merge(merge(L, m), R);
141 }
142 void pintar(int i, int j, int color) {
143     node L, m, R;
144     split(root, i, L, R);
145     split(R, j - i + 1, m, R);
146     m->push_lazy(PAINT, color);
147     root = merge(merge(L, m), R);
148 }
149 void query(int i, int j) {
150     node L, m, R;
151     split(root, i, L, R);
152     split(R, j - i + 1, m, R);
153     int negros = m->negros;
154     int blancos = m->cnt - negros;
155     printf("%d_%d\n", negros, blancos);
156     root = merge(merge(L, m), R);
157 }
158 void print(node &t, string spacio, char lado = '_') {
159     if (t == nullptr) return;
160     cout << spacio << lado << " _->_" << (t->to_string()) << endl;
161     print(t->l, spacio + "\t", 'L');
162     print(t->r, spacio + "\t", 'R');
163 }
164 void print() {
165     print(root, "");
166 }
167 };

```

## 2.11 Convex Hull Trick Estatico

```

1 // g++ "convexhulltrick.cpp" -o run
2 /**
3 ===== <Convex hull trick normal version> =====
4 Contain a sample about convex hull trick optimization this recieve N
   pairs:
5 a "value of length" and a cost, we need to minimize the value of
   grouping

```

```

6 this pairs taken the most large pair as the cost of the group
7
8 Problem for practice: acquire
9 */
10 #include <iostream>
11 #include <vector>
12 #include <algorithm>
13 using namespace std;
14 int pointer; //Keeps track of the best line from previous query
15 vector<long long> M; //Holds the slopes of the lines in the envelope
16 vector<long long> B; //Holds the y-intercepts of the lines in the
    envelope
17 //Returns true if either line l1 or line l3 is always better than line
    l2
18 bool bad(int l1,int l2,int l3)
19 {
20     /*
21     intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
22     intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
23     set the former greater than the latter, and cross-multiply to
24     eliminate division
25     */
26     return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
27 }
28 //Adds a new line (with lowest slope) to the structure
29 void add(long long m,long long b)
30 {
31     //First, let's add it to the end
32     M.push_back(m);
33     B.push_back(b);
34     //If the penultimate is now made irrelevant between the
        antepenultimate
35     //and the ultimate, remove it. Repeat as many times as necessary
36     while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
37     {
38         M.erase(M.end()-2);
39         B.erase(B.end()-2);
40     }
41 }
42 //Returns the minimum y-coordinate of any intersection between a given
    vertical
43 //line and the lower envelope
44 long long query(long long x)

```

```

45 {
46     //If we removed what was the best line for the previous query, then
        the
47     //newly inserted line is now the best for that query
48     if (pointer>=M.size())
49         pointer=M.size()-1;
50     //Any better line must be to the right, since query values are
        //non-decreasing
51     while (pointer<M.size()-1&&
52         M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
53         pointer++;
54     return M[pointer]*x+B[pointer];
55 }
56
57 int main()
58 {
59     int M,N,i;
60     pair<int,int> a[50000];
61     pair<int,int> rect[50000];
62     scanf("%d",&M);
63     for (i=0; i<M; i++)
64         scanf("%d_%d",&a[i].first,&a[i].second);
65     //Sort first by height and then by width (arbitrary labels)
66     sort(a,a+M);
67     for (i=0,N=0; i<M; i++)
68     {
69         /*
70         When we add a higher rectangle, any rectangles that are also
71         equally thin or thinner become irrelevant, as they are
72         completely contained within the higher one; remove as many
73         as necessary
74         */
75         while (N>0&&rect[N-1].second<=a[i].second)
76             N--;
77         rect[N++]=a[i]; //add the new rectangle
78     }
79     long long cost;
80     add(rect[0].second,0);
81     //initially, the best line could be any of the lines in the envelope,
82     //that is, any line with index 0 or greater, so set pointer=0
83     pointer=0;
84     for (i=0; i<N; i++)
85     {
86         cost=query(rect[i].first);

```

```

87     if (i<N)
88         add(rect[i+1].second,cost);
89     }
90     printf("%lld\n",cost);
91     return 0;
92 }

```

## 2.12 Convex Hull Trick Dinamico

```

1 // g++ -std=c++11 "convexhulltrick_dynamic.cpp" -o run
2 /**
3 ===== <Convex hull trick dynamic version version>
4 =====
5 warning with the use of this, this is a black box, try to use only in an
6 emergency.
7 Problem for practice: aquire
8 */
9 #include <bits/stdc++.h>
10 using namespace std;
11 typedef long long ll;
12 const ll is_query = -(1LL<<62);
13 struct Line {
14     ll m, b;
15     mutable multiset<Line>::iterator it;
16     const Line *succ(multiset<Line>::iterator it) const;
17     bool operator<(const Line& rhs) const {
18         if (rhs.b != is_query) return m < rhs.m;
19         const Line *s=succ(it);
20         if(!s) return 0;
21         ll x = rhs.m;
22         return b - s->b < (s->m - m) * x;
23     }
24 };
25 struct HullDynamic : public multiset<Line>{ // will maintain upper hull
26     for maximum
27     bool bad(iterator y) {
28         iterator z = next(y);
29         if (y == begin()) {
30             if (z == end()) return 0;
31             return y->m == z->m && y->b <= z->b;
32         }
33         iterator x = prev(y);
34         if (z == end()) return y->m == x->m && y->b <= x->b;
35     }
36 }

```

```

32     return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
33 }
34 iterator next(iterator y){return ++y;}
35 iterator prev(iterator y){return --y;}
36 void insert_line(ll m, ll b) {
37     iterator y = insert((Line) { m, b });
38     y->it=y;
39     if (bad(y)) { erase(y); return; }
40     while (next(y) != end() && bad(next(y))) erase(next(y));
41     while (y != begin() && bad(prev(y))) erase(prev(y));
42 }
43 ll eval(ll x) {
44     Line l = *lower_bound((Line) { x, is_query });
45     return l.m * x + l.b;
46 }
47 }h;
48 const Line *Line::succ(multiset<Line>::iterator it) const{
49     return (++it==h.end())? NULL : &*it;}

```

## 2.13 Misof Tree

```

1 /*
2 http://codeforces.com/blog/entry/10493#comment-159335
3 Sirve para encontrar el i-esimo numero de un conjunto de numeros que
4 vamos insertando en el arbol.
5 Sirve solo si nuestros numeros son del 0 al n-1 (pero podemos mapearlos
6 antes de usarlos)
7 La idea es esta:
8 Funcionamiento:
9 - En el fondo sigue siendo un Segment-Tree (hacemos que 'n' sea 2^x)
10 - Cada nodo guarda cuantos numeros hay en el intervalo (entonces en
11 tree[1] dice cuantos numeros tenemos en total)
12 - Se sigue representando los hijos del nodo 'i' con '2 * i' (izq) y '2
13 * i + 1' (der);
14 Query:
15 - si kth es mas grande que todos los que tenemos(tree[1]) o es
16 negativo entonces -1
17 - siempre nos mantenemos en el nodo de la izquierda y si es necesario
18 avanzamos al de la derecha
19 'i <= 1'
20 - si kth es mas grande que el nodo de la izquierda(el actual) quiere
21 decir que podemos quitarle todos esos

```

```

15     numeros a nuestra busqueda 'kth - tree[i]' y buscar el nuevo kth en
        el arbol de la derecha
16     if (kth > tree [i]) kth -= tree [i++];
17     - Ojo en el 'i++' ahi es donde avanzamos al nodo de la derecha
18     - luego hace su formula rara que aun no entendi xD:
19     'i - leaf + (kth > tree [i])';
20 */
21 const int MaxN = 1e6;
22
23 int a [MaxN], s [MaxN];
24 int leaf, tree [100 + MaxN << 2];
25
26 void bld (int n) { leaf = 1 << (32 - __builtin_clz (n)); }
27 void add (int x) { for (int i = leaf + x; i; i >>= 1) ++tree [i]; }//
        Podemos insertar mas de una copia la vez tree [i] += xcopies;
28 void del (int x) { for (int i = leaf + x; i; i >>= 1) --tree [i]; }//
        Podemos eliminar mas de una copia la vez tree [i] -= xcopies;
29 // en "leaf + x" esta cuantas copias tenemos de "x"
30 //Cuidado con intentar hacer del con mas copias de las disponibles, el
        kth() no funcionaria
31 long kth (int kth, int i = -1) {
32     if (kth > tree [1] || kth <= 0) return i;
33     for (i = 1; i < leaf; i <= 1) if (kth > tree [i]) kth -= tree [i++];
34     return i - leaf + (kth > tree [i]);
35 }

```

## 2.14 SQRT Decomposition Basic

```

1 const int maxn = 500010;
2 int n;
3
4 tipo v[maxn]; //vector principal
5
6 tipo lazy[maxn];
7 pair<tipo, tipo> t[maxn]; //para poder reordenar los elementos
8
9 int SQRT;
10 int N; //nro. de buckets
11
12 //Recalcula y aplica el lazy al bucket con indice idx
13 //guarda la informacion necesaria del bucket en otros vectores
14 //podria ser la suma del bucket, o el min/max del bucket
15 void recalc(int idx) {

```

```

16     int a = idx * SQRT, b = min(n, (idx + 1) * SQRT);
17     for (int i = a; i < b; i++) {
18         v[i] += lazy[idx];
19         t[i] = make_pair(v[i], i);
20     }
21     lazy[idx] = 0;
22     sort(t + a, t + b);
23 }
24
25 //adiciona delta a todos los elementos
26 //en el intervalo cerrado [a, b]
27 void add(int a, int b, tipo delta) {
28     int idx_a = a / SQRT, idx_b = b / SQRT;
29     if (idx_a == idx_b) {
30         for (int i = a; i <= b; i++)
31             v[i] += delta;
32         recalc(idx_a);
33     } else {
34         //head
35         for (int i = a, lim = min(n, (idx_a + 1) * SQRT); i < lim; i++)
36             v[i] += delta;
37         recalc(idx_a); //OJO puede ser necesario
38         //body
39         for (int i = idx_a + 1; i < idx_b; i++)
40             lazy[i] += delta;
41         //tail
42         for (int i = idx_b * SQRT; i <= b; i++)
43             v[i] += delta;
44         recalc(idx_b); //OJO puede ser necesario
45     }
46 }
47
48 //tambien podria ser en un rango como en el add
49 tipo query(tipo val) {
50     tipo ans = 0;
51     //recorro todos los buckets
52     for (int idx = 0; idx < N; idx++) {
53         int a = idx * SQRT, b = min(n, (idx + 1) * SQRT);
54         //... hacer algo ...
55     }
56     return ans;
57 }
58 int main() {

```

```

59 //leer n, q y los elementos de v
60
61 SQRT = (int)sqrt(n) + 1;
62 N = (n + SQRT - 1) / SQRT; //nro. de buckets
63 //construir cada bucket
64 for (int idx = 0; idx < N; idx++)
65     recalc(idx);
66
67 //resto del programa
68 return 0;
69 }

```

## 2.15 Nro. Elementos menores o iguales a $x$ en $O(\log(n))$

```

1 //insercion y consulta de cuantos <= en log n
2 struct leqset {
3     int maxl; vector<int> c;
4     int pref(int n, int l) { return (n>>(maxl-l))|(1<<l); }
5     void ini(int ml) { maxl=ml; c=vector<int>(1<<(maxl+1)); }
6     //inserta c copias de e, si c es negativo saca c copias
7     void insert(int e, int q=1) { for(n=l,maxl+1) c[pref(e,l)]+=q; }
8     int leq(int e) {
9         int r=0,a=1;
10        for(i,maxl) {
11            a<<=1; int b=(e>>maxl-i-1)&1;
12            if (b) r+=c[a]; a|=b;
13        } return r + c[a]; //sin el c[a] da los estrictamente menores
14    }
15    int size() { return c[1]; }
16    int count(int e) { return c[e|(1<<maxl)]; }
17 };

```

## 3 Algos

### 3.1 LIS en $O(n \log n)$ con Reconstruccion

```

1 //Para non-increasing, cambiar comparaciones y revisar busq binaria
2 //Given an array, paint it in the least number of colors so that each
   color turns to a non-increasing subsequence.
3 //Solution:Min number of colors=Length of the longest increasing
   subsequence
4
5 // Las lineas marcadas con // Camino no son necesarias si no se desea
   reconstruir el camino.

```

```

6 #define MAXN 1000000
7 int v[MAXN]; // INPUT del algoritmo.
8 int mv[MAXN];
9 int mi[MAXN], p[MAXN]; // Camino
10 int l[MAXN]; // Aca apareceria la maxima subsecuencia creciente(los
   indices)
11 int lis(int n) {
12     for(n,i,n) mv[i] = INF;
13     for(n,i,n) mi[i] = -1; // Camino
14     for(n,i,n) p[i] = -1; // Camino
15     mv[0] = -INF;
16     int res = 0;
17     for(i,n) {
18         // Con upper_bound es maxima subsecuencia no decreciente.
19         // Con lower_bound es maxima subsecuencia creciente.
20         int me = upper_bound(mv,mv+n,v[i]) - mv;
21         p[i] = mi[me-1]; // Camino
22         mv[me] = v[i];
23         mi[me] = i; // Camino
24         if (me > res) res = me;
25     }
26     for(int a = mi[res], i = res - 1; a != -1; a = p[a], i--) // Camino
27         l[i] = a; // Indices: poniendo l[i] = v[a] quedan los valores.
28     return res;
29 }

```

## 3.2 Mo

```

1 // g++ -std=c++11 "mo.cpp" -o run
2 /**
3 ===== <Mo> =====
4 Contain a sample about Mo algorithm
5 Brief explanation when use Mo:
6 Explain where and when we can use above algorithm
7
8 As mentioned, this algorithm is offline, that means we cannot use it
   when we are forced to stick to given order of queries.
9 That also means we cannot use this when there are update operations.
   Not just that, there is one important possible limitation:
10 We should be able to write the functions add and remove. There will be
   many cases where add is trivial but remove is not.
11 One such example is where we want maximum in a range. As we add elements
   , we can keep track of maximum. But when we remove elements

```

```

12 it is not trivial. Anyways in that case we can use a set to add elements
    , remove elements and report minimum.
13 In that case the add and delete operations are O(log N) (Resulting in O(
    N * Sqrt(N) * log N) algorithm).
14
15 Suggestion first use the add operation, then the erase operation
16 Problem for practice: DQUERY spoj
17 Input: N, then N elements of array M querys with a range L,R
18 */
19 const int MAXV = 1e6 + 10;
20 const int N = 30010;
21 const int M = 200010;
22 int cnt[MAXV];
23 int v[N];
24
25 struct query{
26     int l,r,pos;
27     query(){}
28 };
29 int n;
30 query qu[M];
31 int ans[M];
32
33 int ret = 0;
34 void add(int pos){
35     pos = v[pos];
36     cnt[pos]++;
37     if(cnt[pos] == 1){
38         ret++;
39     }
40 }
41 void erase(int pos){
42     pos = v[pos];
43     cnt[pos]--;
44     if(!cnt[pos])ret--;
45 }
46 int main(){
47     n = in();
48     for(int i = 0; i < n;i++){
49         v[i] = in();
50     }
51     int block = ceil(sqrt(n));
52     int q = in();

```

```

53     for(int i = 0; i < q;i++){
54         qu[i].l = in() - 1,qu[i].r = in() - 1,qu[i].pos = i;
55     }
56     sort(qu,qu + q,&)(const query &a,const query &b){
57         if(a.l / block != b.l / block)
58             return a.l / block < b.l / block;
59         return a.r < b.r;
60     });
61     int l = 0, r = 0;
62     for(int i = 0; i < q;i++){
63         int nl = qu[i].l,nr = qu[i].r;
64         while(l > nl){
65             add(--l);
66         }
67         while(r <= nr){
68             add(r++);
69         }
70         while(l < nl){
71             erase(l++);
72         }
73         while(r > nr + 1){
74             erase(--r);
75         }
76
77         ans[qu[i].pos] = ret;
78     }
79     for(int i = 0; i < q;i++)printf("%d\n",ans[i]);
80 }

```

### 3.3 Ternary Search - Reales

$f[x]$  increases and then decreases, and we want the maximum value of  $f[x]$ .

```

1 //yeputons ~ 300 iteraciones es mas que suficiente
2 double l = a, r = b;
3 for(int i=0; i<200; i++) {
4     double l1 = (l*2+r)/3;
5     double l2 = (l+2*r)/3;
6     if(f(l1) > f(l2))
7         r = l2;
8     else
9         l = l1;
10 }
11 x = l;

```

Si  $f(x)$  es facil derivar se deriva  $f(x) - > f'(x)$  luego se iguala  $f'(x) = 0$  y se despeja  $x$ , estamos consiguiendo el punto donde la pendiente es zero (si es de varias variables se deriva parcialmente).

Si no se puede despejar  $x$  de  $f'(x)$  y estamos seguros que tiene forma de parabola, se puede *aplicar binary search*, buscamos el punto donde la pendiente cambie de signo(osea sea igual a 0)

### 3.4 Ternary Search - Enteros

$f[x]$  increases and then decreases, and we want the maximum value of  $f[x]$ .

```
1 //f[x] increases and then decreases, and we want the maximum value of f[
  x].
2 int lo = -1, hi = n;
3 while (hi - lo > 1){
4     int mid = (hi + lo)>>1;
5     if (f(mid) > f(mid + 1))
6         hi = mid;
7     else
8         lo = mid;
9 }
10 //lo + 1 is the answer
```

## 4 Strings

### 4.1 Manacher

```
1 vector<int> manacher(const string &s) {
2     int n = s.size();
3     string s(2 * n + 3, '#');
4     s[0] = '%', s[s.size() - 1] = '$';//no deben estar en la cadena
5     for (int i = 0; i < n; i++)
6         s[(i + 1) * 2] = s[i];
7
8     n = s.size();
9     vector<int> P(n, 0);
10    int C = 0, R = 0;
11    for (int i = 1; i < n - 1; i++) {
12        int j = C - (i - C);
13        if (R > i)
14            P[i] = min(R - i, P[j]);
15        while (s[i + 1 + P[i]] == s[i - 1 - P[i]])
```

```
16        P[i]++;
17        if (i + P[i] > R) {
18            C = i;
19            R = i + P[i];
20        }
21    }
22    return P;
23 }
24 bool is_pal(const vector<int> &mnch_vec, int i, int j) {//[i, j] - i<=j
25     int len = j - i + 1;
26     i = (i + 1) * 2;//idx to manacher vec idx
27     j = (j + 1) * 2;
28     int mid = (i + j) / 2;
29     return mnch_vec[mid] >= len;
30 }
31 int main() {
32     string s;
33     cin >> s;
34     vector<int> mnch_vec= manacher(s);
35     if (is_pal(mnch_vec, 2, 7)) {
36         //la subcadena desde la posicion 2 a la 7 es palindrome
37     }
38     return 0;
39 }
```

### 4.2 Trie(estatico)

```
1 int trie[10000000][26];
2 int sig;
3 int root = 0;
4 void reset() {
5     sig = -1;
6     addNode();//Root
7 }
8 void addNode() {
9     sig++;
10    memset(trie[sig], -1, sizeof(trie[sig]));
11 }
12 void insert(const string &s) {
13     int cur = root;
14     for (int i = 0; i < sz(s); i++) {
15         int c = s[i] - 'a';
16         if (trie[cur][c] == -1) {
```

```

17     addNode();
18     trie[cur][c] = sig;
19 }
20 cur = trie[cur][c];
21 }
22 }

```

### 4.3 Suffix Array $O(n \log n)$ con LCP (Kasai) $O(n)$

#### 4.4 Minima rotacion lexicografica

```

1  /*
2  Rotacion Lexicografica minima MinRotLex(cadena,tamano)
3  para cambiar inicio de la cadena char s[300]; int h; s+h;
4  retorna inicio de la rotacion minima :D
5  */
6  int MinRotLex(const char *s, const int slen) {
7      int i = 0, j = 1, k = 0, x, y, tmp;
8      while(i < slen && j < slen && k < slen) {
9          x = i + k;
10         y = j + k;
11         if(x >= slen) x -= slen;
12         if(y >= slen) y -= slen;
13         if(s[x] == s[y]) {
14             k++;
15         } else if(s[x] > s[y]) {
16             i = j+1 > i+k+1 ? j+1 : i+k+1;
17             k = 0;
18             tmp = i, i = j, j = tmp;
19         } else {
20             j = i+1 > j+k+1 ? i+1 : j+k+1;
21             k = 0;
22         }
23     }
24     return i;
25 }
26 int main(){
27     int n;
28     scanf("%d",&n);getchar();
29     while(n--){
30         char str[1000009];
31         gets(str);
32         printf("%d\n",MinRotLex(str,strlen(str))+1);
33     }

```

```

34 }

```

## 4.5 Matching

### 4.5.1 KMP

```

1  string T;//cadena donde buscar(what)
2  string P;//cadena a buscar(what)
3  int b[MAXLEN];//back table b[i] maximo borde de [0..i)
4  void kmppre(){//by gabina with love
5      int i=0, j=-1; b[0]=-1;
6      while(i<sz(P)){
7          while(j>=0 && P[i] != P[j]) j=b[j];
8          i++, j++, b[i] = j;
9      }
10 }
11 void kmp(){
12     int i=0, j=0;
13     while(i<sz(T)){
14         while(j>=0 && T[i]!=P[j]) j=b[j];
15         i++, j++;
16         if(j==sz(P)) printf("P is found at index %d in T\n", i-j), j=b[j];
17     }
18 }
19
20 int main(){
21     cout << "T=";
22     cin >> T;
23     cout << "P=";
24     cin.ignore();
25     cin >> P;
26     kmppre();
27     kmp();
28     return 0;
29 }

```

### 4.5.2 Z - Por aprender

### 4.5.3 Matching con hash

```

1  typedef unsigned long long ull;
2  const int p = 500009;
3  int x;
4  long long pw[500010];

```



```

5 long long h[500010];
6 long long poly(const string &t,int le,int ri){
7     long long h = 0LL;
8     string s = "";
9     for(int i = le; i < ri;i++)s += t[i];
10    for(int i = s.size() - 1; i >= 0; i--){
11        h = ( ( h % p ) * ( x % p ) + s[i] ) % p;
12    }
13    return h;
14 }
15 bool equal(const string &s,const string &t,int ini){
16     for(int i = 0; i < t.size();i++){
17         if(s[i + ini] != t[i])return false;
18     }
19     return true;
20 }
21 int main() {
22     x = (rand() % (p - 1) ) + 1;
23     pw[0] = 1;
24     for(int i = 1; i <= 5e5 + 5;i++){
25         pw[i] = (pw[i - 1] * x % p) % p;
26     }
27     string pattern, text;
28     cin >> pattern >> text;
29     long long y = 1LL;
30     h[text.size() - pattern.size()] = poly(text, text.size() - pattern.
        size(), text.size());
31     for(int i = 0; i < pattern.size(); i++){
32         y = ( y % p * x % p ) % p;
33     }
34     for(int i = text.size() - pattern.size() - 1; i >= 0; i--){
35         long long cal = (text[i] - (y * text[i + pattern.size()]) % p + p )
            % p;
36         if(cal < 0) cal += p;
37         h[i] = ( ( x * h[i + 1] ) % p + cal) % p;
38     }
39     vector<int>res;
40     int pHash = poly(pattern,0,pattern.size());
41     for(int i = 0; i <= text.size() - pattern.size(); i++){
42         if(pHash != h[i])continue;
43         if(equal(text,pattern,i))res.push_back(i);
44     }
45     for(auto u:res)cout << u << " ";

```

```

46     cout << "\n";
47     return 0;
48 }

```

#### 4.5.4 Matching con suffix array

```

1 vector<int> FindOccurrences(const string& pattern, const string& text) {
2     vector<int> result;
3     int minIndex = 0,maxIndex = text.size();
4     while(minIndex < maxIndex){
5         int mid = (minIndex + maxIndex) >> 1;
6         if(cmp(pattern,sa[mid]) > 0)minIndex = mid + 1;
7         else maxIndex = mid;
8     }
9     int start = minIndex;
10    maxIndex = text.size();
11    while(minIndex < maxIndex){
12        int mid = (minIndex + maxIndex) >> 1;
13        if(cmp(pattern,sa[mid]) < 0)maxIndex = mid;
14        else minIndex = mid + 1;
15    }
16    int end = maxIndex;
17    for(int i = start; i < end;i++){
18        result.push_back(sa[i]);
19    }
20    return result;
21 }

```

#### 4.5.5 Matching con BWT

```

1 map<char,int>fo;//first ocurrence
2 map<char,vector<int> >count;//count the i-th ocurrence of symbol
3 string first;//first column of bwt
4 string alpha = "ACGT$";//change this
5 void preprocess(const string& bwt) { //recieves a BWT
6     string ans = "";
7     first = bwt;
8     sort(first.begin(),first.end());
9     for(int i = 0;first[i];i++){
10         if(!fo.count(first[i]))fo[first[i]] = i;
11     }
12     for(char u:alpha)count[u].push_back(0);
13     for(int i = 1; i <= bwt.size();i++){
14         for(char u:alpha)

```

```

15     count[u].push_back(count[u].back() + (bwt[i - 1] == u));
16 }
17 }
18 //return the number of occurrences of the pattern
19 int bwtmatch(int bot,string &pattern){
20     int top = 0;
21     while(top <= bot){
22         if(pattern.size()){
23             char letter = pattern.back();
24             pattern.pop_back();
25             if(count[letter][bot + 1]){
26                 top = fo[letter] + count[letter][top];
27                 bot = fo[letter] + count[letter][bot + 1] - 1;
28             }
29             else return 0;
30         }
31         else return bot - top + 1;
32     }
33     return 0;
34 }

```

#### 4.5.6 Matching con Aho-Corasick

```

1 struct trie{
2     map<char, trie> next;
3     trie* tran[256]; //transiciones del automata
4     int idhoja, szhoja; //id de la hoja o 0 si no lo es
5     //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que
6     //es hoja
7     trie *padre, *link, *nxthoja;
8     char pch; //caracter que conecta con padre
9     trie(): tran(), idhoja(), padre(), link() {}
10    void insert(const string &s, int id=1, int p=0){ //id>0!!!
11        if(p<sz(s)){
12            trie &ch=next[s[p]];
13            tran[(int)s[p]]=&ch;
14            ch.padre=this, ch.pch=s[p];
15            ch.insert(s, id, p+1);
16        }
17        else idhoja=id, szhoja=sz(s);
18    }
19    trie* get_link() {

```

```

20    if(!link){
21        if(!padre) link=this; //es la raiz
22        else if(!padre->padre) link=padre; //hijo de la raiz
23        else link=padre->get_link()->get_tran(pch);
24    }
25    return link; }
26    trie* get_tran(int c) {
27        if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
28        return tran[c]; }
29    trie *get_nxthoja(){
30        if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
31        return nxthoja; }
32    void print(int p){
33        if(idhoja) cout << "found" << idhoja << "at position" << p-
34            szhoja << endl;
35        if(get_nxthoja()) get_nxthoja()->print(p); }
36    void matching(const string &s, int p=0){
37        print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1); }
38    }tri;
39
40    int main(){
41        tri=trie(); //clear
42        tri.insert("ho", 1);
43        tri.insert("hoho", 2);

```

#### 4.6 Suffix Automaton

```

1 //***** Suffix Automata *****/
2 const int N = INSERTE_VALOR; //maxima longitud de la cadena
3 struct State { //OJO!!! tamaño del alfabeto, si MLE -> map
4     State *pre,*go[26]; //se puede usar un map<char, State*> go
5     int step;
6     void clear() {
7         pre=0;
8         step=0;
9         memset(go,0,sizeof(go)); //go.clear();
10    }
11 } *root,*last;
12 State statePool[N * 2],*cur;
13 void init() {
14     cur=statePool;
15     root=last=cur++;

```

```

16     root->clear();
17 }
18 void Insert(int w) {
19     State *p=last;
20     State *np=cur++;
21     np->clear();
22     np->step=p->step+1;
23     while(p&&!p->go[w])
24         p->go[w]=np,p=p->pre;
25     if(p==0)
26         np->pre=root;
27     else {
28         State *q=p->go[w];
29         if(p->step+1==q->step)
30             np->pre=q;
31         else {
32             State *nq=cur++;
33             nq->clear();
34             memcpy(nq->go,q->go,sizeof(q->go)); //nq->go = q->go; para
35             mapa
36             nq->step=p->step+1;
37             nq->pre=q->pre;
38             q->pre=nq;
39             np->pre=nq;
40             while(p&&p->go[w]==q)
41                 p->go[w]=nq, p=p->pre;
42         }
43     }
44     last=np;
45 }
46
47 /***** Suffix Automata *****/
48
49 /***** Algunas aplicaciones *****/
50 //Obtiene el LCS substring de 2 cadenas en O(|A| + |B|)
51 string lcs(char A[N], char B[N]) {
52     int n,m;
53     n = strlen(A); m = strlen(B);
54     //Construccion: O(|A|)
55     //solo hacerlo una vez si A no cambia
56     init();
57     for(int i=0; i<n; i++)
58         Insert(A[i]-'a'); //Fin construccion
59     //LCS: O(|B|)

```

```

58     int ans = 0, len = 0, bestpos = 0;
59     State *p = root;
60     for(int i = 0; i < m; i++) {
61         int x = B[i]-'a';
62         if(p->go[x]) {
63             len++;
64             p = p->go[x];
65         } else {
66             while (p && !p->go[x]) p = p->pre;
67             if(!p) p = root, len = 0;
68             else len = p->step+1, p = p->go[x];
69         }
70         if (len > ans)
71             ans = len, bestpos = i;
72     }
73     //return ans; //solo el tamaño del lcs
74     return string(B + bestpos - ans + 1, B + bestpos + 1);
75 }
76
77 /*Numero de subcadenas distintas + 1(subcadena vacia) en O(|A|)
78 OJO: Por alguna razon Suffix Array es mas rapido
79 Se reduce a contar el numero de paths que inician en q0 y terminan
80 en cualquier nodo. dp[u] = # de paths que inician en u
81 - Se debe construir el automata en el main(init y Insert's)
82 - Setear dp en -1
83 */
84 number dp[N * 2];
85 number num_dist_substr(State *u = root) {
86     if (dp[u - statePool] != -1) return dp[u - statePool];
87     number ans = 1; //el path vacio que representa este nodo
88     for (int v = 0; v < 26; v++) //usar for (auto) para mapa
89         if (u->go[v])
90             ans += num_dist_substr(u->go[v]);
91     return (dp[u - statePool] = ans);
92 }
93
94 /*Suma la longitud de todos los substrings en O(|A|)
95 - Construir el automata(init y insert's)
96 - Necesita el metodo num_dist_substr (el de arriba)
97 - setear dp's en -1
98 */
99 number dp1[N * 2];
100 number sum_length_dist_substr(State *u = root) {

```

```

101     if (dp1[u - statePool] != -1) return dp1[u - statePool];
102     number ans = 0; //el path vacio que representa este nodo
103     for (int v = 0; v < 26; v++) //usar for (auto) para mapa
104         if (u->go[v])
105             ans += (num_dist_substr(u->go[v]) + sum_length_dist_substr(u
106                 ->go[v]));
107     return (dp1[u - statePool] = ans);
108 }
109 /*
110 Pregunta si p es subcadena de la cadena con la cual esta construida
111 el automata.
112 Complejidad: - Construir O(|Texto|) - solo una vez (init e insert's)
113               - Por Consulta O(|patron a buscar|)
114 */
115 bool is_substring(char p[N]) {
116     State *u = root;
117     for (int i = 0; p[i]; i++) {
118         if (!u->go.count(p[i])) //esta con map!!!
119             return false;
120         u = u->go[p[i]]; //esta con map!!!
121     }
122     return true;
123 }

```

## 4.7 K-esima permutacion de una cadena

```

1 //Entrada: Una cadena cad(std::string), un long th
2 //Salida : La th-esima permutacion lexicografica de cad
3 string ipermutacion(string cad, long long int th){
4     sort(cad.begin(), cad.end());
5     string sol = "";
6     int pos;
7     for(int c = cad.size() - 1; c >= 0; c--){
8         pos = th / fact[c];
9         th %= fact[c];
10        sol += cad[pos];
11        cad.erase(cad.begin() + pos);
12    }
13    return sol;
14 }

```

## 5 Geometria

### 5.1 Cortar Poligono

```

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

```

### 5.2 Interseccion de rectangulos

```

1 #define MAXC 2501
2 struct Rect{
3     int x1,y1, x2,y2;
4     int color;
5     int area;
6     Rect(int _x1, int _y1, int _x2, int _y2){
7         x1 = _x1;
8         y1 = _y1;
9         x2 = _x2;
10        y2 = _y2;
11        getArea();
12    }
13    int getArea(){
14        if(x1>=x2 || y1>=y2) return area = 0;
15        return area = (x2-x1)*(y2-y1);
16    }
17 };
18 Rect interseccion(Rect t, Rect r){
19     int x1,y1,x2,y2;
20     x1 = max(t.x1,r.x1);
21     y1 = max(t.y1,r.y1);
22     x2 = min(t.x2,r.x2);
23     y2 = min(t.y2,r.y2);
24     Rect res(x1,y1,x2,y2);
25     return res;

```

26 | }

### 5.3 Otra representacion de rectas y segmentos

$$y = mx + b$$

La fraccion debe estar normalizada todo el tiempo. O podria causar errores.

```

1 bool between(fraction a, fraction b, fraction c) {
2     return a <= c && c <= b;
3 }
4 struct segment {
5     fraction pendiente, ordenada;
6     fraction A, B;
7     segment(fraction p, fraction ord, fraction a, fraction b) {
8         pendiente = p;
9         ordenada = ord;
10        A = a;
11        B = b;
12    }
13    bool read() { //retorna true si no se vertical
14        fraction a, b, c, d; //(a,b) (c,d)
15        a.read();
16        b.read();
17        c.read();
18        d.read();
19        if (a == c) { //vertical
20            A = min(b, d);
21            B = max(b, d);
22            ordenada = a;
23            return false;
24        }
25        A = min(a, c);
26        B = max(a, c);
27        pendiente = (d - b) / (c - a);
28        ordenada = b - (pendiente * a);
29        return true;
30    }
31    bool same_line(const segment &other) const {
32        return (pendiente == other.pendiente) && (ordenada == other.ordenada);
33    }
34    bool intersect(const segment &other) const {
35        if (!same_line(other)) return false;
36        return between(A, B, other.A) || between(A, B, other.B);

```

```

37    }
38    void merge(const segment &other) const {
39        A = min(A, other.A);
40        B = min(B, other.B);
41    }
42    bool operator < (const segment &other) const {
43        if (A == other.A)
44            return B < other.B;
45        return A < other.A;
46    }
47    string to_string() {
48        stringstream ss;
49        ss << "(" << A.to_string() << ", " << B.to_string() << ")";
50        return ss.str();
51    }
52 };
53 int main() {
54     int n; cin >> n;
55     vector<segment> verticales;
56     vector<segment> normal;
57     for (i, n) {
58         segment x;
59         if (x.read()) {
60             normal.pb(x);
61         } else {
62             verticales.pb(x);
63         }
64     }
65     return 0;
66 }

```

### 5.4 Distancia punto-recta

```

1 double distance_point_to_line(const point &a, const point &b, const
    point &pnt){
2     double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) /
        distsq(a, b);
3     point intersection;
4     intersection.x = a.x + u*(b.x - a.x);
5     intersection.y = a.y + u*(b.y - a.y);
6     return dist(pnt, intersection);
7 }

```

## 5.5 Distancia punto-segmento

```

1 struct point{
2     double x,y;
3 };
4 inline double dist(const point &a, const point &b){
5     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
6 }
7 inline double distsq(const point &a, const point &b){
8     return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
9 }
10 double distance_point_to_segment(const point &a, const point &b, const
    point &pnt){
11     double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) /
        distsq(a, b);
12     point intersection;
13     intersection.x = a.x + u*(b.x - a.x);
14     intersection.y = a.y + u*(b.y - a.y);
15
16     if (u < 0.0 || u > 1.0)
17         return min(dist(a, pnt), dist(b, pnt));
18
19     return dist(pnt, intersection);
20 }

```

## 6 Math

### 6.1 Identidades

$$\begin{aligned}
 \sum_{i=0}^n \binom{n}{i} &= 2^n \\
 \sum_{i=0}^n i \binom{n}{i} &= n * 2^{n-1} \\
 \sum_{i=m}^n i &= \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2} \\
 \sum_{i=0}^n i &= \sum_{i=1}^n i = \frac{n(n+1)}{2} \\
 \sum_{i=0}^n i^2 &= \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \\
 \sum_{i=0}^n i(i-1) &= \frac{8}{6} \left(\frac{n}{2}\right) \left(\frac{n}{2} + 1\right) (n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par} \\
 \sum_{i=0}^n i^3 &= \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^n i\right]^2 \\
 \sum_{i=0}^n i^4 &= \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30} \\
 \sum_{i=0}^n i^p &= \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^p \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1} \\
 r &= e - v + k + 1
 \end{aligned}$$

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

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

## 6.2 Ec. Caracteristica

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

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

Sean  $r_1, r_2, \dots, r_q$  las raíces distintas, de mult.  $m_1, m_2, \dots, m_q$

$$T(n) = \sum_{i=1}^q \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$$

Las constantes  $c_{ij}$  se determinan por los casos base.

## 6.3 Identidades de agustin y mario

## 6.4 Combinatorio

```

1 for(i, MAXN+1){ //comb[i][k]=i tomados de a k
2     comb[i][0]=comb[i][i]=1;
3     forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
4 }
5 ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p teniendo comb[p][p]
    precalculado.
6     ll aux = 1;
7     while (n + k) aux = (aux * comb[n%p][k%p]) %p, n/=p, k/=p;
8     return aux;
9 }

```

## 6.5 Exp. de Numeros Mod.

```

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

```

## 6.6 Matrices de 2x2 y Fibonacci en log(n)

```

1 tipo mod;
2 struct M22 { //|a b|
3     tipo a, b, c, d; //|c d|
4     M22 operator * (const M22 &p) const {
5         return (M22){(a*p.a+b*p.c) % mod, (a*p.b+b*p.d) % mod, (c*p.a+d*p.c)
            % mod, (c*p.b+d*p.d) % mod};
6     }
7 };
8 M22 operator ^ (const M22 &p, tipo n) {
9     if(!n) return (M22){1, 0, 0, 1}; //matriz identidad
10    M22 q = p ^ (n/2); q = q * q;

```

```

11 return (n % 2)? p * q : q;
12 }
13 //devuelve el n-esimo fibonacci (index 0)
14 //f0 = fibo(0), f1 = fibo(1);
15 tipo fibo(tipo n, tipo f0, tipo f1) {
16     M22 mat=(M22){0, 1, 1, 1}^n;
17     return (mat.a*f0 + mat.b*f1) % mod;
18 }

```

## 6.7 Gauss Jordan

```

1 const int N = 300;
2 typedef vector<double> col;
3 typedef vector<double> row;
4 typedef vector<row>Matrix;
5 col solution;
6 int main(){
7     Matrix M;
8     M.resize(300);
9     solution.resize(300);
10    for(int i = 0; i < 30;i++)M[i].resize(30);
11    int n;
12    cin >> n;
13    for(int i = 0; i < n;i++)
14        for(int j = 0; j <= n;j++)
15            cin >> M[i][j];
16
17    for(int j = 0; j < n - 1;j++){
18        int l =j;
19        for(int i = j + 1; i < n;i++){
20            if(fabs(M[i][j]) > fabs(M[l][j]))l = i;
21        }
22        for(int k = j; k <= n;k++){
23            swap(M[j][k],M[l][k]);
24        }
25        for(int i = j + 1; i < n;i++)
26            for(int k = n; k >= j;k--)
27                M[i][k] -= M[j][k] * M[i][j] / M[j][j];
28    }
29    double t = 0;
30    for(int j = n - 1; j >= 0; j--){
31        t = 0.0;
32        for(int k = j + 1; k < n;k++)t += M[j][k] * solution[k];

```

```

33     solution[j] = (M[j][n] - t) / M[j][j];
34 }
35 cout.precision(4);
36 for(int i = 0; i < n;i++)cout<<fixed << solution[i] << " ";
37 return 0;
38 }

```

## 6.8 Tridiagonal

```

1 //TRIDIAGONAL SOLVER
2 // solve ai*x_i-1 + bi*x_i + ci*x_i+1 = d_i
3 //a_0 = 0 c_n-1 = 0
4 /*
5     b :subdiagonal
6     a :diagonal
7     c :super diagonal
8 */
9 //the answer is in D
10 #define MAXN 5000
11 long double A[MAXN], B[MAXN], C[MAXN], D[MAXN], X[MAXN];
12 void solve(int n) {
13     C[0] /= B[0]; D[0] /= B[0];
14     for(int i = 1; i < n - 1;i++) C[i] /= B[i] - A[i]*C[i-1];
15     for(int i = 1; i < n;i++)D[i] = (D[i] - A[i] * D[i-1]) / (B[i] - A[i]
16         ] * C[i-1]);
17     X[n-1] = D[n-1];
18     for (int i = n-2; i>=0; i--) X[i] = D[i] - C[i] * X[i+1];
19 }

```

## 6.9 Simplex

```

1 // Two-phase simplex algorithm for solving linear programs of the form
2 //
3 //     maximize    c^T x
4 //     subject to  Ax <= b
5 //                x >= 0
6 //
7 // INPUT: A -- an m x n matrix
8 //         b -- an m-dimensional vector
9 //         c -- an n-dimensional vector
10 //         x -- a vector where the optimal solution will be stored
11 //
12 // OUTPUT: value of the optimal solution (infinity if unbounded
13 //         above, nan if infeasible)

```

```

14 //
15 // To use this code, create an LPSolver object with A, b, and c as
16 // arguments. Then, call Solve(x).
17
18 #include <iostream>
19 #include <iomanip>
20 #include <vector>
21 #include <cmath>
22 #include <limits>
23
24 using namespace std;
25
26 typedef long double DOUBLE;
27 typedef vector<DOUBLE> VD;
28 typedef vector<VD> VVD;
29 typedef vector<int> VI;
30
31 const DOUBLE EPS = 1e-9;
32
33 struct LPSolver {
34     int m, n;
35     VI B, N;
36     VVD D;
37
38     LPSolver(const VVD &A, const VD &b, const VD &c) :
39         m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
40         for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
41         for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
42         for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
43         N[n] = -1; D[m + 1][n] = 1;
44     }
45
46     void Pivot(int r, int s) {
47         double inv = 1.0 / D[r][s];
48         for (int i = 0; i < m + 2; i++) if (i != r)
49             for (int j = 0; j < n + 2; j++) if (j != s)
50                 D[i][j] -= D[r][j] * D[i][s] * inv;
51         for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
52         for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
53         D[r][s] = inv;
54         swap(B[r], N[s]);

```

```

55     }
56
57     bool Simplex(int phase) {
58         int x = phase == 1 ? m + 1 : m;
59         while (true) {
60             int s = -1;
61             for (int j = 0; j <= n; j++) {
62                 if (phase == 2 && N[j] == -1) continue;
63                 if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
64                     N[s]) s = j;
65             }
66             if (D[x][s] > -EPS) return true;
67             int r = -1;
68             for (int i = 0; i < m; i++) {
69                 if (D[i][s] < EPS) continue;
70                 if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
71                     (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B[r])
72                     r = i;
73             }
74             if (r == -1) return false;
75             Pivot(r, s);
76         }
77
78         DOUBLE Solve(VD &x) {
79             int r = 0;
80             for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
81             if (D[r][n + 1] < -EPS) {
82                 Pivot(r, n);
83                 if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<
84                     DOUBLE>::infinity(); //NO SOLUTION
85             }
86             for (int i = 0; i < m; i++) if (B[i] == -1) {
87                 int s = -1;
88                 for (int j = 0; j <= n; j++)
89                     if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j]
90                         < N[s]) s = j;
91                 Pivot(i, s);
92             }
93             if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity(); //
94             INFINITY
95             x = VD(n);
96             for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];

```



```

93     return D[m][n + 1]; //solution find
94 }
95 };
96
97 int main() {
98
99     const int m = 4;
100    const int n = 3;
101    DOUBLE _A[m][n] = {
102        { 6, -1, 0 },
103        { -1, -5, 0 },
104        { 1, 5, 1 },
105        { -1, -5, -1 }
106    };
107    DOUBLE _b[m] = { 10, -4, 5, -5 };
108    DOUBLE _c[n] = { 1, -1, 0 };
109
110    VVD A(m);
111    VD b(_b, _b + m);
112    VD c(_c, _c + n);
113    for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
114
115    LPSolver solver(A, b, c);
116    VD x;
117    DOUBLE value = solver.Solve(x);
118
119    cerr << "VALUE:_" << value << endl; // VALUE: 1.29032
120    cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
121    for (size_t i = 0; i < x.size(); i++) cerr << "_" << x[i];
122    cerr << endl;
123    return 0;
124 }

```

## 6.10 Matrices y determinante $O(n^3)$

```

1 struct Mat {
2     vector<vector<double> > vec;
3     Mat(int n): vec(n, vector<double>(n) ) {}
4     Mat(int n, int m): vec(n, vector<double>(m) ) {}
5     vector<double> &operator[](int f){return vec[f];}
6     const vector<double> &operator[](int f) const {return vec[f];}
7     int size() const {return sz(vec);}
8     Mat operator+(Mat &b) { ///this de n x m entonces b de n x m

```

```

9         Mat m(sz(b),sz(b[0]));
10        forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j];
11        return m;    }
12    Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
13        int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
14        Mat mat(n,t);
15        forn(i,n) forn(j,t) forn(k,m) mat[i][j] += vec[i][k] * b[k][j];
16        return mat;    }
17    double determinant(){//sacado de e maxx ru
18        double det = 1;
19        int n = sz(vec);
20        Mat m(*this);
21        forn(i, n){//para cada columna
22            int k = i;
23            forr(j, i+1, n)//busco la fila con mayor val abs
24                if(abs(m[j][i])>abs(m[k][i])) k = j;
25            if(abs(m[k][i])<1e-9) return 0;
26            m[i].swap(m[k]); //la swapeo
27            if(i!=k) det = -det;
28            det *= m[i][i];
29            forr(j, i+1, n) m[i][j] /= m[i][i];
30            //hago 0 todas las otras filas
31            forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
32                forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
33        }
34        return det;
35    }
36 };
37
38 int n;
39 int main() {
40     //DETERMINANTE:
41     //https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&
42     //page=show_problem&problem=625
43     freopen("input.in", "r", stdin);
44     ios::sync_with_stdio(0);
45     while(cin >> n && n){
46         Mat m(n);
47         forn(i, n) forn(j, n) cin >> m[i][j];
48         cout << (ll)round(m.determinant()) << endl;
49     }
50     cout << "*" << endl;

```

```

50     return 0;
51 }

```

### 6.11 Teorema Chino del Resto

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

### 6.12 Criba

```

1  #define MAXP 100000 //no necesariamente primo
2  int criba[MAXP+1];
3  void crearcriba(){
4      int w[] = {4,2,4,2,4,6,2,6};
5      for(int p=25;p<=MAXP;p+=10) criba[p]=5;
6      for(int p=9;p<=MAXP;p+=6) criba[p]=3;
7      for(int p=4;p<=MAXP;p+=2) criba[p]=2;
8      for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
9          for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
10 }
11 vector<int> primos;
12 void buscarprimos(){
13     crearcriba();
14     forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
15 }
16 //~ Useful for bit trick: #define SET(i) ( criba[i]>>5)|=1<<((i)&31) ),
17 //~ #define INDEX(i) ( (criba[i]>>5)>>((i)&31))&1 ), unsigned int criba[
18 //~ MAXP/32+1];
19
20 int main() {
21     freopen("primos", "w", stdout);
22     buscarprimos();

```

### 6.13 Funciones de primos

Sea  $n = \prod p_i^{k_i}$ , fact(n) genera un map donde a cada  $p_i$  le asocia su  $k_i$

```

1  //factoriza bien numeros hasta MAXP^2
2  map<ll,ll> fact(ll n){ //0 (cant primos)
3      map<ll,ll> ret;
4      forall(p, primos){
5          while(!(n%*p)){
6              ret[*p]++; //divisor found

```

```

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

```

```

49     if ((ll)i*i > n) break;
50     if (n % i == 0){
51         while (n%i == 0) n/=i;
52         r -= r/i; }
53     }
54     if (n != 1) r-= r/n;
55     return r;
56 }
57
58 int main() {
59     buscarprimos();
60     forr (x,1, 500000){
61         cout << "x_=" << x << endl;
62         cout << "Numero_de_factores_primos:" << numPrimeFactors(x) << endl;
63         cout << "Numero_de_distintos_factores_primos:" <<
64             numDiffPrimeFactors(x) << endl;
65         cout << "Suma_de_factores_primos:" << sumPrimeFactors(x) << endl;
66         cout << "Numero_de_divisores:" << numDiv(x) << endl;
67         cout << "Suma_de_divisores:" << sumDiv(x) << endl;
68         cout << "Phi_de_Euler:" << eulerPhi(x) << endl;
69     }
70     return 0;
71 }

```

## 6.14 Phollard's Rho (rolando)

```

1 ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}
2
3 ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c, and minimize overflow
4     ll x = 0, y = a%c;
5     while (b > 0){
6         if (b % 2 == 1) x = (x+y) % c;
7         y = (y*2) % c;
8         b /= 2;
9     }
10    return x % c;
11 }
12
13 ll expmod (ll b, ll e, ll m){//O(log b)
14     if(!e) return 1;
15     ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
16     return e%2? mulmod(b,q,m) : q;

```

```

17 }
18
19 bool es_primo_prob (ll n, int a)
20 {
21     if (n == a) return true;
22     ll s = 0,d = n-1;
23     while (d % 2 == 0) s++,d/=2;
24
25     ll x = expmod(a,d,n);
26     if ((x == 1) || (x+1 == n)) return true;
27
28     forn (i, s-1){
29         x = mulmod(x, x, n);
30         if (x == 1) return false;
31         if (x+1 == n) return true;
32     }
33     return false;
34 }
35
36 bool rabin (ll n){ //devuelve true si n es primo
37     if (n == 1) return false;
38     const int ar[] = {2,3,5,7,11,13,17,19,23};
39     forn (j,9)
40         if (!es_primo_prob(n,ar[j]))
41             return false;
42     return true;
43 }
44
45 ll rho(ll n){
46     if( (n & 1) == 0 ) return 2;
47     ll x = 2 , y = 2 , d = 1;
48     ll c = rand() % n + 1;
49     while( d == 1 ){
50         x = (mulmod( x , x , n ) + c)%n;
51         y = (mulmod( y , y , n ) + c)%n;
52         y = (mulmod( y , y , n ) + c)%n;
53         if( x - y >= 0 ) d = gcd( x - y , n );
54         else d = gcd( y - x , n );
55     }
56     return d==n? rho(n):d;
57 }
58
59 map<ll,ll> prim;

```

```

60 void factRho (ll n){ //0 (lg n)^3. un solo numero
61     if (n == 1) return;
62     if (rabin(n)){
63         prim[n]++;
64         return;
65     }
66     ll factor = rho(n);
67     factRho(factor);
68     factRho(n/factor);
69 }

```

### 6.15 GCD

```

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

```

### 6.16 Extended Euclid

```

1 number ext_gcd(number a, number b, number &x, number &y){
2     if(b == 0) {
3         x = 1; y = 0; return a;
4     }
5     number g = ext_gcd(b,a % b,y,x);
6     y -= a / b * x;
7     return g;
8 }
9 pair<number, number> solve(number a, number b, number c, number n,
10     number m, number g, number tk) {
11     pair<number, number> ans(LLONG_MAX, LLONG_MAX);
12     for (number k = tk - 3; k < tk + 3; k++) {
13         number x = (n * c) / g + (b * k) / g;
14         number y = (m * c) / g - (a * k) / g;
15         if (x >= 1 && y >= 1)
16             ans = min(ans, mp(x, y));
17     }
18     return ans;
19 }
20 int main() {
21     //Encuentra a y b de a*x + b*x = c
22     //tal que a > 0 && b > 0 y minimiza (a, b)
23     number a, b, c;
24     scanf("%lld%lld%lld", &a, &b, &c);
25     number n, m;
26     number g = ext_gcd(a, b, n, m);
27     number L = (g - n * c) / b, R = (m * c - g) / a;

```

```

27 auto ans =min(solve(a, b, c, n, m, g, L),solve(a, b, c, n, m, g, R));
28 printf("%lld%lld\n", ans.first, ans.second);
29 return 0;
30 }

```

### 6.17 LCM

```

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

```

### 6.18 Inversos

```

1 #define MAXMOD 15485867
2 ll inv[MAXMOD]; //inv[i]*i=1 mod MOD
3 void calc(int p){ //0(p)
4     inv[1]=1;
5     forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
6 }
7 int inverso(int x){ //0(log x)
8     return expmod(x, eulerphi(MOD)-2); //si mod no es primo(sacar a mano)
9     return expmod(x, MOD-2); //si mod es primo
10 }

```

### 6.19 Simpson

```

1 double integral(double a, double b, int n=10000) { //0(n), n=cantdiv
2     double area=0, h=(b-a)/n, fa=f(a), fb;
3     forn(i, n){
4         fb=f(a+h*(i+1));
5         area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
6     }
7     return area*h/6.;}

```

### 6.20 Fraction

```

1 | tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
2 struct frac{
3     tipo p,q;
4     frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
5     void norm(){
6         tipo a = mcd(p,q);
7         if(a) p/=a, q/=a;
8         else q=1;
9         if (q<0) q=-q, p=-p;}
10     frac operator+(const frac& o){

```

```

11     tipo a = mcd(q,o.q);
12     return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
13 frac operator-(const frac& o){
14     tipo a = mcd(q,o.q);
15     return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
16 frac operator*(frac o){
17     tipo a = mcd(q,o.p), b = mcd(o.q,p);
18     return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
19 frac operator/(frac o){
20     tipo a = mcd(q,o.q), b = mcd(o.p,p);
21     return frac((p/b)*(o.q/a), (q/a)*(o.p/b));}
22 bool operator<(const frac &o) const{return p*o.q < o.p*q;}
23 bool operator==(frac o){return p==o.p&&q==o.q;}
24 };

```

## 6.21 Polinomio

```

1     int m = sz(c), n = sz(o.c);
2     vector<tipo> res(max(m,n));
3     forn(i, m) res[i] += c[i];
4     forn(i, n) res[i] += o.c[i];
5     return poly(res);    }
6 poly operator*(const tipo cons) const {
7     vector<tipo> res(sz(c));
8     forn(i, sz(c)) res[i]=c[i]*cons;
9     return poly(res);    }
10 poly operator*(const poly &o) const {
11     int m = sz(c), n = sz(o.c);
12     vector<tipo> res(m+n-1);
13     forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[j];
14     return poly(res);    }
15 tipo eval(tipo v) {
16     tipo sum = 0;
17     dforn(i, sz(c)) sum=sum*v + c[i];
18     return sum; }
19 //poly contains only a vector<int> c (the coefficients)
20 //the following function generates the roots of the polynomial
21 //it can be easily modified to return float roots
22 set<tipo> roots(){
23     set<tipo> roots;
24     tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
25     vector<tipo> ps,qs;
26     forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);

```

```

27     forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
28     forall(pt,ps)
29         forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
30             tipo root = abs((*pt) / (*qt));
31             if (eval(root)==0) roots.insert(root);
32         }
33     return roots; }
34 };
35 pair<poly,tipo> ruffini(const poly p, tipo r) {
36     int n = sz(p.c) - 1 ;
37     vector<tipo> b(n);
38     b[n-1] = p.c[n];
39     dforn(k,n-1) b[k] = p.c[k+1] + r*b[k+1];
40     tipo resto = p.c[0] + r*b[0];
41     poly result(b);
42     return make_pair(result,resto);
43 }
44 poly interpolate(const vector<tipo>& x,const vector<tipo>& y) {
45     poly A; A.c.pb(1);
46     forn(i,sz(x)) { poly aux; aux.c.pb(-x[i]), aux.c.pb(1), A = A * aux;
47     }
48     poly S; S.c.pb(0);
49     forn(i,sz(x)) { poly Li;
50         Li = ruffini(A,x[i]).fst;
51         Li = Li * (1.0 / Li.eval(x[i])); // here put a multiple of the
52         coefficients instead of 1.0 to avoid using double
53         S = S + Li * y[i]; }
54     return S;
55 }
56 int main(){
57     return 0;
58 }

```

## 6.22 Ec. Lineales

```

1 bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
2     int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
3     vector<int> p; forn(i,m) p.push_back(i);
4     forn(i, rw) {
5         int uc=i, uf=i;
6         forr(f, i, n) forr(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;
            uc=c;}

```

```

7   if (freq(a[uf][uc], 0)) { rw = i; break; }
8   forn(j, n) swap(a[j][i], a[j][uc]);
9   swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
10  tipo inv = 1 / a[i][i]; //aca divide
11  forr(j, i+1, n) {
12      tipo v = a[j][i] * inv;
13      forr(k, i, m) a[j][k] -= v * a[i][k];
14      y[j] -= v*y[i];
15  }
16  } // rw = rango(a), aca la matriz esta triangulada
17  forr(i, rw, n) if (!freq(y[i],0)) return false; // chequeo de
    compatibilidad
18  x = vector<tipo>(m, 0);
19  dforn(i, rw){
20      tipo s = y[i];
21      forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
22      x[p[i]] = s / a[i][i]; //aca divide
23  }
24  ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
25  forn(k, m-rw) {
26      ev[k][p[k+rw]] = 1;
27      dforn(i, rw){
28          tipo s = -a[i][k+rw];
29          forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
30          ev[k][p[i]] = s / a[i][i]; //aca divide
31      }
32  }
33  return true;
34  }

```

## 6.23 Karatsuba

```

1  // g++ -std=c++11 "karatsuba.cpp" -o hld
2
3  /**
4  ===== <karatsuba> =====
5  Complexity: O(N^1.7)
6  Call to karatsuba function paramter two vectors
7  * INPUT: two vectors A,B cointains the coeficients of the polynomail
8  * OUTPUT a vector coitains the coeficients of A * B
9  */
10
11 int p,k;

```

```

12 vector<int> b,r;
13
14 void trim(vector<int>& a){
15     while (a.size() > 0 && a.back() == 0) a.pop_back();
16 }
17
18 vector<int> multiply(const vector<int>& a, const vector<int>& b){
19     vector<int> c(a.size() + b.size() + 1, 0);
20     for (int i = 0; i < a.size(); i++) {
21         for (int j = 0; j < b.size(); j++) {
22             c[i+j] += a[i] * b[j];
23         }
24     }
25     trim(c);
26     return c;
27 }
28 // a = a + b*(10^k)
29 void addTo(vector<int>& a, const vector<int>& b, int k){
30     if (a.size() < b.size() + k) a.resize(b.size() + k);
31     for (int i = 0; i < b.size(); i++) a[i+k] += b[i];
32 }
33 void subFrom(vector<int>& a, const vector<int>& b){
34     for (int i = 0; i < b.size(); i++) a[i] -= b[i];
35 }
36 // a = a + b
37 void addTo(vector<int>& a, const vector<int>& b){
38     addTo(a, b, 0);
39 }
40 vector<int> karatsuba(const vector<int>& a, const vector<int>& b)
41 {
42     int alen = a.size();
43     int blen = b.size();
44     if (alen == 0 || blen == 0) return vector<int>();
45     if (alen < blen) return karatsuba(b, a);
46     if (alen < 50) return multiply(a, b);
47
48     int half = alen / 2;
49     vector<int> a0(a.begin(), a.begin() + half);
50     vector<int> a1(a.begin() + half, a.end());
51     vector<int> b0(b.begin(), b.begin() + min<int>(blen, half));
52     vector<int> b1(b.begin() + min<int>(blen, half), b.end());
53
54     vector<int> z0 = karatsuba(a0, b0);

```

```

55     vector<int> z2 = karatsuba(a1, b1);
56     addTo(a0, a1);
57     addTo(b0, b1);
58     vector<int> z1 = karatsuba(a0, b0);
59     subFrom(z1, z0);
60     subFrom(z1, z2);
61
62     vector<int> res;
63     addTo(res, z0);
64     addTo(res, z1, half);
65     addTo(res, z2, half + half);
66
67     trim(res);
68     return res;
69 }

```

## 6.24 FFT

```

1  #define lowbit(x) (((x) ^ (x-1)) & (x))
2  typedef complex<long double> Complex;
3
4  void FFT(vector<Complex> &A, int s){
5      int n = A.size();
6      int p = __builtin_ctz(n);
7
8      vector<Complex> a = A;
9
10     for(int i = 0; i < n; ++i){
11         int rev = 0;
12         for(int j = 0; j < p; ++j){
13             rev <<= 1;
14             rev |= ((i >> j) & 1);
15         }
16         A[i] = a[rev];
17     }
18
19     Complex w, wn;
20
21     for(int i = 1; i <= p; ++i){
22         int M = (1 << i), K = (M >> 1);
23         wn = Complex(cos(s*2.0*M_PI/(double)M), sin(s*2.0*M_PI/(double)M));
24     }

```

```

25     for(int j = 0; j < n; j += M){
26         w = Complex(1.0, 0.0);
27         for(int l = j; l < K+j; ++l){
28             Complex t = w;
29             t *= A[l + K];
30             Complex u = A[l];
31             A[l] += t;
32             u -= t;
33             A[l + K] = u;
34             w *= wn;
35         }
36     }
37 }
38
39 if(s == -1){
40     for(int i = 0; i < n; ++i)
41         A[i] /= (double)n;
42 }
43 }
44
45 vector<Complex> FFT_Multiply(vector<Complex> &P, vector<Complex> &Q){
46     int n = P.size() + Q.size();
47     while(n != lowbit(n)) n += lowbit(n);
48
49     P.resize(n, 0);
50     Q.resize(n, 0);
51
52     FFT(P, 1);
53     FFT(Q, 1);
54
55     vector<Complex> R;
56     for(int i = 0; i < n; ++i) R.push_back(P[i] * Q[i]);
57
58     FFT(R, -1);
59
60     return R;
61 }
62
63 // Para multiplicacion de enteros grandes
64 const long long B = 100000;
65 const int D = 5;

```



## 6.25 Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales	
0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 ( $\in \text{int}$ )
2! = 2	13! = 6.227.020.800
3! = 6	14! = 87.178.291.200
4! = 24	15! = 1.307.674.368.000
5! = 120	16! = 20.922.789.888.000
6! = 720	17! = 355.687.428.096.000
7! = 5.040	18! = 6.402.373.705.728.000
8! = 40.320	19! = 121.645.100.408.832.000
9! = 362.880	20! = 2.432.902.008.176.640.000 ( $\in \text{tint}$ )
10! = 3.628.800	21! = 51.090.942.171.709.400.000
max signed tint = 9.223.372.036.854.775.807	
max unsigned tint = 18.446.744.073.709.551.615	

### Primos

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

### Primos cercanos a $10^n$

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

### Cantidad de primos menores que $10^n$

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

### Divisores

Cantidad de divisores ( $\sigma_0$ ) para *algunos*  $n/\neg\exists n' < n, \sigma_0(n') \geq \sigma_0(n)$   
 $\sigma_0(60) = 12$  ;  $\sigma_0(120) = 16$  ;  $\sigma_0(180) = 18$  ;  $\sigma_0(240) = 20$  ;  $\sigma_0(360) = 24$   
 $\sigma_0(720) = 30$  ;  $\sigma_0(840) = 32$  ;  $\sigma_0(1260) = 36$  ;  $\sigma_0(1680) = 40$  ;  $\sigma_0(10080) = 72$   
 $\sigma_0(15120) = 80$  ;  $\sigma_0(50400) = 108$  ;  $\sigma_0(83160) = 128$  ;  $\sigma_0(110880) = 144$   
 $\sigma_0(498960) = 200$  ;  $\sigma_0(554400) = 216$  ;  $\sigma_0(1081080) = 256$  ;  $\sigma_0(1441440) = 288$   
 $\sigma_0(4324320) = 384$  ;  $\sigma_0(8648640) = 448$

Suma de divisores ( $\sigma_1$ ) para *algunos*  $n/\neg\exists n' < n, \sigma_1(n') \geq \sigma_1(n)$   
 $\sigma_1(96) = 252$  ;  $\sigma_1(108) = 280$  ;  $\sigma_1(120) = 360$  ;  $\sigma_1(144) = 403$  ;  $\sigma_1(168) = 480$   
 $\sigma_1(960) = 3048$  ;  $\sigma_1(1008) = 3224$  ;  $\sigma_1(1080) = 3600$  ;  $\sigma_1(1200) = 3844$   
 $\sigma_1(4620) = 16128$  ;  $\sigma_1(4680) = 16380$  ;  $\sigma_1(5040) = 19344$  ;  $\sigma_1(5760) = 19890$   
 $\sigma_1(8820) = 31122$  ;  $\sigma_1(9240) = 34560$  ;  $\sigma_1(10080) = 39312$  ;  $\sigma_1(10920) = 40320$   
 $\sigma_1(32760) = 131040$  ;  $\sigma_1(35280) = 137826$  ;  $\sigma_1(36960) = 145152$  ;  $\sigma_1(37800) = 148800$   
 $\sigma_1(60480) = 243840$  ;  $\sigma_1(64680) = 246240$  ;  $\sigma_1(65520) = 270816$  ;  $\sigma_1(70560) = 280098$   
 $\sigma_1(95760) = 386880$  ;  $\sigma_1(98280) = 403200$  ;  $\sigma_1(100800) = 409448$   
 $\sigma_1(491400) = 2083200$  ;  $\sigma_1(498960) = 2160576$  ;  $\sigma_1(514080) = 2177280$   
 $\sigma_1(982800) = 4305280$  ;  $\sigma_1(997920) = 4390848$  ;  $\sigma_1(1048320) = 4464096$   
 $\sigma_1(4979520) = 22189440$  ;  $\sigma_1(4989600) = 22686048$  ;  $\sigma_1(5045040) = 23154768$   
 $\sigma_1(9896040) = 44323200$  ;  $\sigma_1(9959040) = 44553600$  ;  $\sigma_1(9979200) = 45732192$

## 7 Grafos

### 7.1 Bellman-Ford

```

1 int negative_cycle(vector<vector<int> > &G, vector<vector<int> > &cost)
2 {
3     //write your code here
4     bool nc = false;
5     int n = G.size();
6     vector<int> dist(n, INT_MAX / 2);
7     dist[0] = 0;
8     for(int i = 0; i < n - 1; i++)
9         for(int u = 0; u < n; u++)
10             for(int j = 0; j < G[u].size(); j++){
11                 int v = G[u][j];
12                 int w = cost[u][j];
13                 dist[v] = min(dist[v], dist[u] + w);
14             }
15 }
```



```

13     }
14     for(int u = 0; u < n;u++){
15         for(int j = 0; j < G[u].size();j++){
16             int v = G[u][j];
17             int w = cost[u][j];
18             if(dist[v] > dist[u] + w)nc = true;
19         }
20     }
21     return nc;
22 }

```

## 7.2 dijkstra grafos densos

### 7.3 2 SAT definitivamente no con Tarjan

```

1 // g++ -std=c++11 "twosat.cpp" -o run
2 /**
3 ===== <Two Sat> =====
4 Complexity: O(N)
5 Input: number of variables, then number of clause clauses in format (u
        or v)
6 if u,v > 0 then is equivalent to u,v
7 if u,v < 0 then is equivalent to u , v
8 Output: UNSATISFIABLE can't find a solution
9 SATISFIABLE if exist a solution then print the assignment of all
        variables (negative for xi = false)
10
11 Examples:
12 Input:
13 3 3
14 1 -3
15 -1 2
16 -2 -3
17 Output
18 SATISFIABLE
19 1 2 -3
20 *
21 Input
22 1 2
23 1 1
24 -1 -1
25 Output
26 UNSATISFIABLE
27 */

```

```

28 #include <bits/stdc++.h>
29 using namespace std;
30 vector<int>G[2][2000010],G2[2000010];
31 int n, m;
32 int scc[2000010];
33 bool vis[2000010];
34 vector<int>comp[2000010];
35 int assign[2000010];
36 int cc = 0;
37 stack<int>st;
38 vector<int>sta;
39 void dfs(int u,int type){
40     if(scc[u] != -1)return;
41     scc[u] = cc;
42     for(int v:G[type][u]){
43         dfs(v,type);
44     }
45     if(!type)st.push(u);
46 }
47 void topo(int u){
48     if(vis[u])return;
49     vis[u] = true;
50     for(int v:G2[u])topo(v);
51     sta.push_back(u);
52 }
53 void buildGraphWitouthLoop(){
54     for(int i = 0;i < 2 * n;i++){
55         for(int j = 0;j < G[0][i].size();j++){
56             if(scc[i] != scc[G[0][i][j]])
57                 G2[scc[i]].push_back(scc[G[0][i][j]]);
58         }
59     }
60 }
61 int main() {
62     ios::sync_with_stdio(false);cin.tie(0);
63     cin >> n >> m;
64     for(int i = 0,u,v; i < m;i++){
65         cin >> u >> v;
66         int uu = (u > 0?(u - 1) * 2:(-u - 1) * 2 + 1);
67         int vv = (v > 0?(v - 1) * 2:(-v - 1) * 2 + 1);
68         // cout << uu << " " << (uu ^ 1) << "\n";
69         G[0][uu ^ 1].push_back(vv);
70         G[0][vv ^ 1].push_back(uu);

```

```

71     G[1][vv].push_back(uu ^ 1);
72     G[1][uu].push_back(vv ^ 1);
73 }
74 memset(scc,-1,sizeof scc);
75 for(int i = 0; i < 2 * n;i++){
76     if(scc[i] == -1)dfs(i,0);
77 }
78 memset(scc,-1,sizeof scc);
79 while(!st.empty()){
80     int u = st.top();st.pop();
81     if(scc[u] == -1){
82         dfs(u,1);
83         cc++;
84     }
85 }
86 bool unsat = false;
87 for(int i = 0; i < 2 * n;i++){
88     if(scc[i] == scc[i ^ 1])unsat = true;
89     comp[scc[i]].push_back(i);
90 }
91 if(unsat){
92     return cout << "UNSATISFIABLE",0;
93 }
94 cout << "SATISFIABLE\n";
95 buildGraphWithouthLoop();
96 for(int i = 0; i < 2 * n;i++){
97     if(!vis[i])topo(i);
98 }
99 for(int u:sta){//inverse of topological sort
100     for(int v:comp[u]){//transitivite Skew-Symmetry
101         if(!assign[v]){
102             assign[v] = 1;
103             assign[v ^ 1] = -1;
104         }
105     }
106 }
107 for(int i = 0,j = 1; i < 2 * n; i += 2,j++){
108     cout << (j) * (assign[i]) << "□";
109 }
110 return 0;
111 }

```

## 7.4 Prim

### 7.5 Articulation Points (desgraciadamente tarjan)

```

1 // g++ -std=c++11 "articulationpointsandbridges.cpp" -o run
2 /**
3 ===== <Articulation points and bridges c++ version>
4 =====
5 Given a graph return a vector of paris with the bridges and a bool array
6 art[]
7 true if the node is an articulation point
8 * false otherwise
9 Graph nodes: 0 to N - 1
10 */
11 using namespace std;
12 vector<int>G[10010];
13 int low[10010],num[10010],parent[10010],cc;
14 //cc is my timer
15 int art[10010];//bool for detect art point, int for detect how many
16     nodes are connected to my articulation point
17 int root,rC;
18 int n;
19 vector<pair<int,int> >bridges;
20 void dfs(int u){
21     low[u]=num[u]=cc++;
22     for(int v:G[u]){
23         if(num[v]==-1){
24             parent[v]=u;
25             if(u==root)rC++;
26             dfs(v);
27             if(low[v]>=num[u])art[u]++;//is a articulation point
28             if(low[v]>num[u])bridges.push_back({u,v});//this is a bridge
29             low[u]=min(low[u],low[v]);
30         }
31         else if(v!=parent[u]){
32             low[u]=min(low[u],num[v]);
33         }
34     }
35 }
36 void init(){
37     bridges.clear();
38     for(int i=0;i<n;i++){
39         art[i]=low[i]=0;

```

```

37     num[i]=parent[i]=-1;
38     G[i].clear();
39 }
40 cc=0;
41 }
42 void callARTBRID(){
43     for(int i=0;i<n;i++){
44         if(num[i]==-1){
45             root=i,rC=0;dfs(i);
46             art[root]=(rC>1);
47         }
48     }
49 }

```

## 7.6 componentes biconexas y puentes (block cut tree)

```

1 int V;
2 vector<int> G[MAXN];
3 int dfn[MAXN],low[MAXN];
4 vector< vector<int> > C;
5 stack< pair<int, int> > stk;
6 void cache_bc(int x, int y){
7     vector<int> com;
8     int tx,ty;
9     do{
10         tx = stk.top().first, ty = stk.top().second;
11         stk.pop();
12         com.push_back(tx), com.push_back(ty);
13     }while(tx!=x || ty!=y);
14     C.push_back(com);
15 }
16
17 void DFS(int cur, int prev, int number){
18     dfn[cur] = low[cur] = number;
19     for(int i = G[cur].size()-1;i>=0;--i){
20         int next = G[cur][i];
21         if(next==prev) continue;
22         if(dfn[next]==-1){
23             stk.push(make_pair(cur,next));
24             DFS(next,cur,number+1);
25             low[cur] = min(low[cur], low[next]);
26             if(low[next]>=dfn[cur]) cache_bc(cur,next);
27         }else low[cur] = min(low[cur],dfn[next]);

```

```

28     }
29 }
30
31 void biconn_comp(){
32     memset(dfn,-1,sizeof(dfn));
33     C.clear();
34     DFS(0,0,0);
35     int comp = C.size();
36     printf("%d\n",comp);
37     for(int i = 0;i < comp;++i){
38         sort(C[i].begin(),C[i].end());
39         C[i].erase(unique(C[i].begin(),C[i].end()),C[i].end());
40         int m = C[i].size();
41         for(int j = 0;j < m;++j) printf("%d_",1 + C[i][j]);
42         printf("\n");
43     }
44 }

```

## 7.7 LCA saltitos potencias de 2

## 7.8 LCA sparse table query O(1)

## 7.9 HLD

```

1 // g++ -std=c++11 "hld.cpp" -o hld
2
3 /**
4 ===== <HLD> =====
5 Complexity: O(N*log (N))
6 Given a tree and asociative operation in the paths of this tree ask for
7     many queries, and updates
8 in nodes or edges
9 Input of this example:
10 N number of nodes, then N elements values in each node
11 then n - 1 conexions
12 Q queries if T == 1 query on the path u,v
13 else update node U with value val.
14
15 Example problems: Spoj QTREE1 to QTREE6, toby and tree UVA
16 */
17 #include <bits/stdc++.h>
18 using namespace std;
19 const int maxn = 1e5;

```

```

20 const int NEUTRO = 0; // a null value for my ST
21 int vec[maxn];
22 vector<int>G[maxn]; //the graph
23 //int idx[maxn]; // case with value in the edge
24 int op(int u,int v){// an operation for my path (using ST)
25     //return __gcd(u,v);
26     //return max(u,v);
27     return u + v;
28 }
29 int n;
30 //ask to Branimir for information about this
31 struct SegmentTree{
32     int T[2*maxn];
33     void init(){
34         memset(T,0,sizeof T);
35     }
36     void set(int pos,int val){
37         pos += n;
38         T[pos] = val;
39         for(pos >= 1; pos > 0; pos >= 1){
40             T[pos] = op( T[pos << 1] , T[(pos << 1)|1] );
41         }
42     }
43     int query(int l,int r){
44         l += n;
45         r += n;
46         int ans = NEUTRO;
47         while( l < r){
48             if ( l & 1 ) ans = op(ans, T[l++] );
49             if ( r & 1 ) ans = op( ans, T[--r] );
50             l >>= 1;
51             r >>= 1;
52         }
53         return ans;
54     }
55 };
56 struct hld{
57     int ncad; // store actual number of chain
58     int root; // the root of a tree generally 0 or 1
59     int pos; // pos of node in chain
60
61     int sz[maxn]; // store the subsize of subtrees
62     int depth[maxn]; //depth of the node, useful for LCA via HLD

```

```

63     int parent[maxn]; // useful for LCA
64     int where[maxn]; // where chain is the node?
65     //int edgepos[maxn]; // if the value is on the edge: stored in a node
66     int chainIdx[maxn]; // position in the chain of the node
67     int head[maxn]; // the head of the i-th chain
68     //int val[maxn]; // if the value is on the edge
69     SegmentTree tree; // this ST allow operations in the path
70
71     void init(){//settings value, and process de HLD
72         root = 0;
73         ncad = 0;
74         pos = 0;
75         for(int i = 0; i <=n; i++){
76             where[i] = head[i] = -1;
77         }
78         depth[root] = 0;
79         dfs(root , -1);
80         descompose(root);
81         tree.init();
82         /* case with values in edges
83         for(int i=0;i<n;i++){
84             tree.set(i,val[i]);
85         }
86         */
87     }
88
89     ///init descomposition
90     void dfs(int u,int pu){
91         sz[u] = 1; //init the sz of this subtree
92         parent[u] = pu; // assign the parent
93         for(int i = 0; i < G[u].size(); i++){
94             int v = G[u][i];
95             if ( v == pu )continue;
96             //edgepos[idx[u][i]] = v;
97             depth[v] = depth[u] + 1;
98             dfs(v,u);
99             sz[u] += sz[v];
100         }
101     }
102 }
103 //descompose graph in HLD descomposition
104 void descompose(int u){
105     if( head[ncad] == -1)head[ncad] = u; // the head of ncad is u

```

```

106 where[u] = ncad; // assign where tu node
107 //val[pos] = cost; cost another parameter in descompose for graphs
    with values in edges
108 chainIdx[u] = pos++; //assing pos to this node
109 int maxi = -1, sc = -1; //finding a special child
110 for(int v:G[u]){
111     if( sz[v] > maxi && where[v] == -1){
112         maxi = sz[v];
113         sc = v;
114     }
115 }
116 if(sc != -1)descompose(sc);
117 //light nodes here:
118 for(int v:G[u]){
119     if(where[v] == -1){
120         ncad++;
121         descompose(v);
122     }
123 }
124 }
125 ///end descomposition
126
127 int lca(int u,int v){
128     while(where[u]!=where[v]){
129         if(depth[ head[ where[u] ] ] > depth[ head[ where[v] ] ] )u =
            parent[ head[ where[u] ] ];
130         else v = parent[ head[ where[v] ] ];
131     }
132     return depth[u] < depth[v] ? u:v;
133 }
134
135 void update(int u, int val){
136     tree.set(chainIdx[u],val);
137 }
138
139 int query(int u,int v){
140     // if ( u == v) return NEUTRO; value in edges
141     int vChain = where[v];
142     int ans = NEUTRO;
143     while(true){
144         int uChain = where[u];
145         if(uChain == vChain){
146             // return op(ans, tree.query( chainIdx[v] + 1, chainIdx[u] + 1)

```

```

    ); value in edges
147     return op(ans, tree.query( chainIdx[v], chainIdx[u] + 1) );
148 }
149 int hu = head[uChain];
150 ans = op( ans, tree.query(chainIdx[hu], chainIdx[u] + 1) );
151 u = parent[hu];
152 }
153 }
154
155 int Q(int u,int v){
156     int L = lca(u,v);
157     return op( query(u,L) , query(v,L) );
158 }
159 }HLD;
160 int main(){
161     //ios::sync_with_stdio(false);cin.tie(0);
162     while(cin >> n){
163         for(int i = 0; i < n; i++)G[i].clear();
164         for(int i = 0; i < n; i++){
165             cin >> vec[i];
166         }
167         for(int i = 1, u,v ; i < n; i++){
168             cin >> u >> v;
169             G[u].push_back(v);
170             G[v].push_back(u);
171             /* case with value in edges
172              G[u].push_back(make_pair(v,w));
173              idx[u].push_back(i-1);
174              G[v].push_back(make_pair(u,w));
175              idx[v].push_back(i-1);
176
177              */
178         }
179         HLD.init();
180         for(int i = 0; i < n; i++){
181             HLD.update(i, vec[i]);
182         }
183         int question;
184         cin >> question;
185         for(int i = 0, t, u ,v; i < question; i++){
186             cin >> t >> u >> v;
187             if( t == 1){
188                 cout << HLD.Q(u,v) << "\n";

```

```

189     }
190     else HLD.update(u,v);
191 }
192 }
193 }

```

## 7.10 centroid decomposition

### 7.11 euler cycle

```

1 int n,m,ars[MAXE], eq;
2 vector<int> G[MAXN]; //fill G,n,m,ars,eq
3 list<int> path;
4 int used[MAXN];
5 bool usede[MAXE];
6 queue<list<int>::iterator> q;
7 int get(int v){
8     while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
9     return used[v];
10 }
11 void explore(int v, int r, list<int>::iterator it){
12     int ar=G[v][get(v)]; int u=v^ars[ar];
13     usede[ar]=true;
14     list<int>::iterator it2=path.insert(it, u);
15     if(u!=r) explore(u, r, it2);
16     if(get(v)<sz(G[v])) q.push(it);
17 }
18 void euler(){
19     zero(used), zero(usede);
20     path.clear();
21     q=queue<list<int>::iterator>();
22     path.push_back(0); q.push(path.begin());
23     while(sz(q)){
24         list<int>::iterator it=q.front(); q.pop();
25         if(used[*it]<sz(G[*it])) explore(*it, *it, it);
26     }
27     reverse(path.begin(), path.end());
28 }
29 void addEdge(int u, int v){
30     G[u].pb(eq), G[v].pb(eq);
31     ars[eq++]=u^v;
32 }

```

## 7.12 diámetro y centro de un árbol

```

1  /**
2  ===== <Diameter and center of a tree> =====
3  //Problem: Given a tree get the center (or centers)
4  /* the nodes in the tree that minimize the length of the longest path
5     from it to any other node.
6  * *Finding tree centers:
7  * If diameter length is even, then we have one tree center. If odd,
8     then we have 2 centers.
9  * E.g. 1-2-3-4-5 -> center is 3
10 * E.g. 1-2-3-4-5-6 -> center is 3, 4
11 * On other side, we can get the worst nodes through the center nodes.
12 * A worst node is one that is an end of a diameter, so it has the worst
13     tree height
14 Input:
15 * No
16 Output:
17 * No
18 dfs: calculate the diameter of the tree
19 * maxi stores the diameter
20 findingCenters() return the centers
21 Nodes in graph 1 to N careful with this
22 Complexity: O(N)
23 */
24
25 vector<int>G[5010];
26 int maxi=-1,far;
27 int n;
28 int pre[5010];
29 int Queue[5010];
30
31 void dfs(int path,int u,int parent){
32     pre[u]=parent;
33     if(path>maxi){
34         maxi=path;
35         far=u;
36     }
37     for(int v:G[u]){
38         if(parent!=v){
39             dfs(path+1,v,u); //path + w if the graph as weighted
40         }
41     }
42 }

```

```

39 }
40 }
41 pair<int,int> findingCenters(){
42     maxi=-1;
43     dfs(0,1,-1);
44     dfs(0,far,-1);
45     int t=far,L=0;
46     while(t!=-1){
47         Queue[L]=t;
48         t=pre[t];
49         ++L;
50     }
51     int a=-1,b=-1;
52     if(L&1){
53         a=Queue[L/2];
54     }
55     else{
56         a=min(Queue[L/2-1],Queue[L/2]),b=max(Queue[L/2-1],Queue[L/2]);
57     }
58     return {a,b};
59 }

```

### 7.13 algoritmo hungaro

### 7.14 union find dinámico

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define dprint(v) cerr << #v"=" << v << endl //;)
4 #define forr(i,a,b) for(int i=(a); i<(b); i++)
5 #define forn(i,n) forr(i,0,n)
6 #define dforn(i,n) for(int i=n-1; i>=0; i--)
7 #define forall(it,v) for(auto it=v.begin();it!=v.end();++it)
8 #define sz(c) ((int)c.size())
9 #define zero(v) memset(v, 0, sizeof(v))
10 #define pb push_back
11 #define fst first
12 #define snd second
13 #define mkp make_pair
14 typedef long long ll;
15 typedef pair<int,int> ii;
16
17 struct UnionFind {
18     int n, comp;

```

```

19     vector<int> pre,si,c;
20     UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
21         forn(i,n) pre[i] = i; }
22     int find(int u){return u==pre[u]?u:find(pre[u]);}
23     bool merge(int u, int v) {
24         if((u=find(u))==v) return false;
25         if(si[u]<si[v]) swap(u, v);
26         si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
27         return true;
28     }
29     int snap(){return sz(c);}
30     void rollback(int snap){
31         while(sz(c)>snap){
32             int v = c.back(); c.pop_back();
33             si[pre[v]] -= si[v], pre[v] = v, comp++;
34         }
35     }
36 };
37 enum {ADD,DEL,QUERY};
38 struct Query {int type,u,v;};
39 struct DynCon {
40     vector<Query> q;
41     UnionFind dsu;
42     vector<int> match,res;
43     map<ii,int> last;//se puede no usar cuando hay identificador para
                        cada arista (mejora poco)
44     DynCon(int n=0):dsu(n){}
45     void add(int u, int v) {
46         if(u>v) swap(u,v);
47         q.pb((Query){ADD, u, v}); match.pb(-1);
48         last[ii(u,v)] = sz(q)-1;
49     }
50     void remove(int u, int v) {
51         if(u>v) swap(u,v);
52         q.pb((Query){DEL, u, v});
53         int prev = last[ii(u,v)];
54         match[prev] = sz(q)-1;
55         match.pb(prev);
56     }
57     void query() { //podria pasarle un puntero donde guardar la respuesta
58         q.pb((Query){QUERY, -1, -1}); match.pb(-1);}
59     void process() {
60         forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] =

```

```

        sz(q);
        go(0,sz(q));
    }
    void go(int l, int r) {
        if(l+1==r){
            if (q[l].type == QUERY)//Aqui responder la query usando el
                dsu!
                res.pb(dsu.comp);//aqui query=cantidad de componentes
                    conexas
            return;
        }
        int s=dsu.snap(), m = (l+r) / 2;
        forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i]
            ].v);
        go(l,m);
        dsu.rollback(s);
        s = dsu.snap();
        forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[
            i].v);
        go(m,r);
        dsu.rollback(s);
    }
}dc;

// Problema ejemplo: http://codeforces.com/gym/100551/problem/A

int n,k;

int main() {
    //~ freopen("in", "r", stdin);
    freopen("connect.in", "r", stdin);
    freopen("connect.out", "w", stdout);
    ios::sync_with_stdio(0);
    while(cin >> n >> k){
        dc=DynCon(n);
        forn(_,k) { string ord; cin >> ord;
            if (ord=="?") {
                dc.query();
            } else if (ord=="+") { int a,b; cin>>a>>b; a--;b--;
                dc.add(a,b);
            } else if (ord=="-") { int a,b; cin>>a>>b; a--;b--;
                dc.remove(a,b);
            } else assert(false);
        }
    }
}

```

```

    }
    if(!k) continue;//k==0 WTF
    dc.process();
    forn(i,sz(dc.res)) cout << dc.res[i] << endl;
}
return 0;
}

```

## 7.15 truquitos estúpidos por ejemplo second MST es con LCA

### 7.16 erdos galloi

```

1 // g++ -std=c++11 "erdosgalloi.cpp" -o run
2 /**
3 ===== <Erdosgalloi c++ version> =====
4 Given the grades of each node of a graph return if this form a valid
5 graph
6 includes: algorithm, functional, numeric, forn
7 // Receives a sorted degree sequence (non ascending)
8 O(NlgN)
9 */
10 bool isGraphicSequence(const vector<int> &seq) // O(n lg n)
11 {
12     vector<int> sum;
13     int n = seq.size();
14
15     if (n == 1 && seq[0] != 0) return false;
16
17     sum.reserve(n + 1);
18     sum.push_back(0);
19     for (int i = 0; i < n; ++i) sum.push_back(sum[i] + seq[i]);
20     if ((sum[n] & 1) == 1) return false;
21
22     for (long long k = 1; k <= n - 1 && seq[k - 1] >= k; ++k) {
23         int j = distance(seq.begin(), upper_bound(seq.begin() + k, seq.end()
24             , k,
25                                     greater<int>())) +
26             1;
27         long long left = sum[k];
28         long long right = k * (k - 1) + (j - k - 1) * k + (sum[n] - sum[j -
29             1]);
30
31         if (left > right) return false;
32     }
33 }

```



```

30 }
31
32 return true;
33 }

```

## 7.17 grafo funcional hallar k-esimo partiendo de un nodo

### 7.18 konig

### 7.19 min-vertex cover bipartitos

### 7.20 max-flow (min cost versión)

```

1 // g++ -std=c++11 "maxflowmincost.cpp" -o run
2 /**
3 ===== <Max flow-min cost c++ version> =====
4 Given a graph with edges with a capacity C and weight D
5 * compute the max-flow min cost
6 Edmond karps idea
7 * Complexity O(v *E*log(v))
8 Problem for practice: Dijkstra Dijkstra uva
9 */
10 #define REP(i,j,k) for(int (i)=(j);(i)<(k);++(i))
11 #define MP make_pair
12
13 using namespace std;
14
15 #define MAXN 500
16 #define MAXM MAXN * 5
17 typedef vector<int> VI;
18 typedef long long ll;
19 const int INF = 1E9; // $infinity$: be careful to make this big enough
20
21 int S; // source
22 int T; // sink
23 int FN; // number of nodes
24 int FM; // number of edges (initialize this to 0)
25 // ra[a]: edges connected to a (NO MATTER WHICH WAY!!!); clear this in
26 // the beginning
27 VI ra[MAXN];
28 int kend[MAXM], cap[MAXM], cost[MAXM]; // size: TWICE the number of
29 // edges
30
31 // Adds an edge from a to b with capacity c and cost d and returns the

```

number of the new edge

```

29
30 int addedge(int a, int b, int c, int d) {
31     int i = 2*FM;
32     kend[i] = b;
33     cap[i] = c;
34     cost[i] = d;
35     ra[a].push_back(i);
36     kend[i+1] = a;
37     cap[i+1] = 0;
38     cost[i+1] = -d;
39     ra[b].push_back(i+1);
40     FM++;
41     return i;
42 }
43
44 int n;
45 int dst[MAXM], pre[MAXM], pret[MAXM];
46 //finding the shortest path via finding duan, also it works with bellman
47 //ford
48 //or dijkstra (careful of negative cycles)
49 bool spfa(){
50     REP(i,0,FN) dst[i] = INF;
51     dst[S] = 0;
52     queue<int> que; que.push(S);
53     while(!que.empty()){
54         int x = que.front(); que.pop();
55         for (int t : ra[x]){
56             int y = kend[t], nw = dst[x] + cost[t];
57             if(cap[t] > 0 && nw<dst[y]){
58                 dst[y] = nw; pre[y] = x; pret[y] = t; que.push(y);
59             }
60         }
61     }
62     return dst[T]!=INF;
63 }
64
65 // returns the maximum flow and the minimum cost for this flow
66 pair<ll,ll> solve(){
67     ll totw = 0, totf = 0;
68     while(spfa()){
69         int minflow = INF;
70         for (int x = T; x!=S; x = pre[x]){
71             minflow = min(minflow, cap[pret[x]]);
72         }
73     }

```

```

70     for (int x = T; x!=S; x = pre[x]){
71         cap[pret[x]] -= minflow;
72         cap[pret[x]^1] += minflow;
73     }
74     totf += minflow;
75     totw += minflow*dst[T];
76 }
77 return make_pair(totf, totw);
78 }
79 void init(){
80     FN=4*n+15; //make this big n=number of nodes of the graph
81     FM=0;
82     S=0, T=n+1;
83     for(int i=0; i<FN; i++) ra[i].clear(); //clear the graph be careful
84 }

```

## 7.21 max-flow corto con matriz

```

1 // g++ "maxflowMVEK.cpp" -o run
2 /**
3 ===== <Max Flow with matriz Edmonds karp c++ version>
4 =====
5 //Given a graph with capacitys find the max-flow
6
7 Nodes indexed 1 to N
8 * Complexity O(N *E)
9 Problem for practice: UVA 820
10 */
11 #define N 500
12 int cap[N][N], pre[N], n;
13 int s; //source
14 int t; //destination
15 bool bfs() {
16     queue<int> q;
17     q.push(s);
18     memset(pre, -1, sizeof pre);
19     pre[s] = s;
20     while(!q.empty()){
21         int u = q.front(); q.pop();
22         if(u == t) return true;
23         for(int i = 1; i <= n; i++){ //nodes 1 to n
24             if(pre[i] == -1 && cap[u][i]) pre[i] = u, q.push(i);
25         }
26     }
27 }

```

```

25 }
26 return false;
27 }
28
29 int maxFlow() {
30     int mf=0, f, v; //max flow, flow for a path, the vertex
31     while(bfs()){ //while encountered a path source to destination
32         v = t; //min
33         f = INT_MAX; //make this big enough
34         while(pre[v] != v){ f = min(f, cap[pre[v]][v]), v = pre[v]; } //finding the
35                                 min capacity
36         v = t; mf += f;
37         while(pre[v] != v){ cap[pre[v]][v] -= f, cap[v][pre[v]] += f, v = pre[v]; }
38                                 //update the flow
39     }
40     return mf;
41 }
42 void init(){
43     memset(cap, 0, sizeof cap);
44     //cap[u][v] += capacidad, cap[v][u] += capacidad
45 }

```

## 7.22 max-flow sin matriz

```

1 // g++ -std=c++11 "maxflowNMEK.cpp" -o run
2 /**
3 ===== <Max Flow with-out matriz Edmonds karp c++ version>
4 =====
5 //Given a graph with capacitys find the max-flow
6
7 Nodes indexed 1 to N
8 * Complexity O(N *E)
9 Problem for practice: UVA 820
10 * Input N number of nodes,
11 * M edges conections
12 * compute the flow with source 1 and sink N
13 */
14 using namespace std;
15 const int N = 110;
16 const int M = 10010 * 2;
17 vector<int> G[N];
18 int kend[M], cap[M], cost[M];
19 int edge = 0;

```

```

19 int s,t;
20 void add(int u,int v,int c){
21     int forward = edge * 2, backward = edge * 2 + 1;
22     kend[forward] = v;
23     cap[forward] = c;
24     G[u].push_back(forward);
25     kend[backward] = u;
26     cap[backward] = 0;
27     G[v].push_back(backward);
28     edge++;
29 }
30 int vis[M],pre[M],pret[M];
31 bool bfs(){
32     for(int i = 0; i <= 100;i++)vis[i] = false;
33     vis[s] = true;
34     queue<int>q;
35     q.push(s);
36     while(!q.empty()){
37         int u = q.front();q.pop();
38         for(int edge:G[u]){
39             int v = kend[edge];
40             if(cap[edge] > 0 && !vis[v]){
41                 vis[v] = true;
42                 pre[v] = u;
43                 pret[v] = edge;//the edge store the information
44                 q.push(v);
45             }
46         }
47     }
48     return vis[t];
49 }
50 int max_flow(){
51     int totf = 0LL;
52     while(bfs()){
53         int minflow = INT_MAX;
54         for(int x = t; x != s; x = pre[x]){
55             minflow = min(minflow,cap[pret[x]]);
56         }
57         for(int x = t; x != s; x = pre[x]){
58             cap[pret[x]] -= minflow;
59             cap[pret[x] ^ 1] += minflow;
60         }
61         totf += minflow;

```

```

62     }
63     return totf;
64 }
65 int main(){
66     int n,m;
67     scanf("%d%d",&n,&m);
68     for(int i = 0,u,v,ca; i < m;i++){
69         scanf("%d%d%d",&u,&v,&ca);
70         add(u,v,ca);
71     }
72     s = 1, t = n;
73     printf("%lld\n",max_flow());
74 }

```

## 7.23 Dinic

```

1
2 const int MAX = 300;
3 // Corte minimo: vertices con dist[v]>=0 (del lado de src) VS. dist[v]
4 // ==-1 (del lado del dst)
5 // Para el caso de la red de Bipartite Matching (Sean V1 y V2 los
6 // conjuntos mas proximos a src y dst respectivamente):
7 // Reconstruir matching: para todo v1 en V1 ver las aristas a vertices
8 // de V2 con it->f>0, es arista del Matching
9 // Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con
10 // dist[v]>0
11 // Max Independent Set: tomar los vertices NO tomados por el Min Vertex
12 // Cover
13 // Max Clique: construir la red de G complemento (debe ser bipartito!) y
14 // encontrar un Max Independet Set
15 // Min Edge Cover: tomar las aristas del matching + para todo vertices
16 // no cubierto hasta el momento, tomar cualquier arista de el
17 int nodes, src, dst;
18 int dist[MAX], q[MAX], work[MAX];
19 struct Edge {
20     int to, rev;
21     ll f, cap;
22     Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(
23         cap) {}
24 };
25 vector<Edge> G[MAX];
26 void addEdge(int s, int t, ll cap){
27     G[s].pb(Edge(t, sz(G[t]), 0, cap)), G[t].pb(Edge(s, sz(G[s])-1, 0,

```

```

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

```

## 7.24 máximo emparejamiento bipartito

```

1 // g++ -std=c "bipartitematching.cpp" -o run
2 /**
3 ===== <MCBM max cardinality bipartite matching c++ version>
4 =====
5 Return the bipartite matching of a Graph
6 * Format of nodes: 1 to N
7 */
8 const int N = 100010;
9 vector<int>G[N];
10 bool v[N]; //for the greedy speed up
11 int match[N];
12 bool vis[N];
13 int n,m;
14 //calling augmenting path
15 bool aug(int u){
16     if(vis[u])return false;
17     vis[u]=true;
18     for(int i=0;i<(int)G[u].size();++i){
19         int r=G[u][i];
20         if(match[r]==-1 || aug(match[r])){
21             match[r]=u;match[u]=r;return true;
22         }
23     }
24     return 0;
25 }
26 int mc;
27 //findging all augmenting path's
28 int solve(){
29     bool check=true;
30     while(check){
31         check=false;
32         memset(vis,0,sizeof vis);
33         for(int i=1;i<=n;++i){
34             if(!v[i]&&match[i]==-1){
35                 bool op=aug(i);
36                 check|=op;
37                 mc+=op;
38             }
39         }
40     }
41     return mc;
42 }

```

```

43 void init(){
44     memset(v,0,sizeof v);
45     memset(vis,false,sizeof vis);
46     mc=0;
47     memset(match,-1,sizeof match);
48     for(int i=0;i<=n;i++)G[i].clear();
49 }
50 void greedySpeedUp(){
51     //greedy optimization, match with the first not matched
52     for(int i=1;i<=n;++i){
53         for(int j=0;j<(int)G[i].size();++j){
54             if(match[G[i][j]]==-1){
55                 match[G[i][j]]=i,match[i]=G[i][j],mc++,v[i]=true;break;
56             }
57         }
58     }
59 }

```

## 7.25 max-independent set en bipartitos

## 7.26 min-path cover (ver tópicos raros de halim)

## 7.27 min-cost arborescence

## 7.28 lema de diapositivas de nico de grafos funcionales

## 7.29 minimax y maximini con kruskal y dijkstra

```

1 // g++ -std=c++11 "maximini.cpp" -o run
2 /**
3 ===== <maximini c++ version> =====
4 Given a weighted graph return the maximini (the maximum of the minimum)
5 or the minimax (the minimum of the maximum) in the path a,b
6 *
7 Minimax as defined as: finding the minimum of maximum edge weight among
8     all possible paths
9 * between two vertices a to b, the cost for a path from a to b is
10    determined by maximum edge
11 * weight along this path. Among all these possible paths from a to b,
12    pick the one with the minimum
13 * ax-edge-weight
14 * Complexity  $O(E \cdot \log(E) + V + E)$ 
15 *
16 Problem for practice: UVA 534,544

```

```

14 */
15 int n;
16 pair<int,pair<int,int> >Edges[20000];
17 int t;
18 map<string,int>mp;
19 int parent[210];
20 pair<int,int>child[210];
21 bool vis[210];
22 vector<pair<int,int> >G[210];
23
24 int find(int u){return u==parent[u]?u:parent[u]=find(parent[u]);}
25 void Union(int u,int v){
26     int pu=find(u),pv=find(v);
27     if(pu!=pv){
28         parent[pv]=pu;
29     }
30 }
31 int mst(int a,int b){
32     sort(Edges,Edges+t);
33     reverse(Edges,Edges+t);//don't reverse for the minimax
34     for(int i=0;i<=200;i++)parent[i] = i;
35     int w,u,v, maximini = 1e8, minimax = 0;
36     for(int i=0;i<t;i++){
37         tie(w,u,v) = make_tuple(Edges[i].first, Edges[i].second.first, Edges
38             [i].second.second);
39         if(find(u) != find(v)){
40             Union(u,v);
41             G[u].push_back({v,w});
42             G[v].push_back({u,w});
43         }
44     }
45     queue<int>q;
46     q.push(a);
47     vis[a]=true;
48     while(!q.empty()){
49         int u = q.front();q.pop();
50         //if(u==1)break;
51         for(pair<int,double>node: G[u]){
52             if(!vis[node.first]){
53                 vis[node.first] = true;
54                 q.push(node.first);
55                 //maximini=max(maximini,node.second);
56                 child[node.first].first = u;

```

```

56     child[node.first].second = node.second;
57 }
58 }
59 }
60 for(int t = b;t != -1;t = child[t].first){
61     //cout<<t<<" "<<child[t].second<<"\n";
62     //minimax=max(minimax,child[t].second);
63     maximini = min(maximini,child[t].second);
64 }
65 return maximini;
66 }

```

## 8 Teoria de juegos

### 8.1 Teorema fundamental de los juegos optimos

```

1 boolean isWinning(position pos) {
2     moves[] = possible positions to which I can move from the position
3     pos;
4     for (all x in moves)
5         if (!isWinning(x)) return true;
6
7     return false;
8 }

```

### 8.2 Como calcular Grundy

```

1 int GrundyNumber(position pos) {
2     moves[] = possible positions to which I can move from pos
3     set s;
4     for (all x in moves) insert into s GrundyNumber(x);
5     //return the smallest non-negative integer not in the set s;
6     int ret=0;
7     while (s.contains(ret)) ret++;
8     return ret;
9 }

```

## 9 Algunas formulas de probabilidad

### Importante

Verificar la independencia de eventos.

### 9.1 Regla general de la probabilidad

$$P(E) = \frac{\text{Nro.casos favorables}}{\text{Nro.casos posibles}}$$

### 9.2 Teorema de Bayes (Probabilidad condicional)

$$P(A/B) = \frac{P(A \cap B)}{P(B)}$$

### 9.3 Regla de la suma

Se suman 2 probabilidades cuando los eventos no pueden ocurrir al mismo tiempo  
"Pasa el evento X o pasa el evento Y"

$$P(A) \cup P(B) = P(A) + P(B)$$

Regla de la suma con eventos compatibles e incompatibles (que pueden pasar al mismo tiempo o no)

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

### 9.4 Regla de la multiplicación

La misma idea de arriba, "Pasa el evento X y pasa el evento Y"

$$P(A \cap B) = P(A) * P(B)$$

### 9.5 Esperanza matemática

$$E[X] = \sum_{i=0}^n X * P(X)$$

En el caso continuo reemplazar sumatoria por integral

### 9.6 Ley de la esperanza total

Esto se usa cuando quieres calcular el valor en el que pasa el evento X  
ej. calcular el día esperado en el que muere el i-ésimo pez

$$E(X) = E(E(X|Y))$$

## 9.7 Esperanza de variables independientes

$$E(X * Y) = E(X) * E(Y)$$

## 9.8 Varianza

$$\sigma^2 = E(X^2) - (E(X))^2$$

## 9.9 Distribucion Binomial

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Ejemplo. Supongamos que se lanza un dado (con 6 caras) 51 veces y queremos conocer la probabilidad de que el número 3 salga 20 veces.

$$f(X = 20) = \binom{51}{20} \frac{1}{6}^{20} * \left(1 - \frac{1}{6}\right)^{51-20}$$

Esperanza:

$$E(x) = n * p$$

Varianza:

$$V(x) = np * (1 - p)$$

# 10 Otros/utilitarios

## 10.1 josephus

```

1 | int survivor(int n, int m){
2 |     int s = 0;
3 |     for (int i=1;i<=n;++i) s = (s+m)%i;
4 |     return (s+1);
5 | }
```

## 10.2 josephus k = 2

```

1 | ///////////////JAVA
2 | /**
3 | *
4 | * @param n the number of people standing in the circle
5 | * @return the safe position who will survive the execution
6 | *   f(N) = 2L + 1 where N = 2^M + L and 0 <= L < 2^M
7 | */
8 | public int getSafePosition(int n) {
9 |     // find value of L for the equation
10 |    int valueOfL = n - Integer.highestOneBit(n);
11 |    int safePosition = 2 * valueOfL + 1;
12 |    return safePosition;
13 | }
```

## 10.3 poker

## 10.4 iterar subconjuntos

```

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

## 10.5 como reconstruir una DP (normal)

```

1 | /*
2 | You just need to revisit your steps in the DP. In case of 0-1 knapsack,
3 | lets say the original DP function was solve, and the function
4 | reconstruct will give you the actual solution (I'm writing the code
5 | in C++):
6 |
7 | */
8 | int solve(int pos, int capacity){
9 |     if(pos == no_of_objects) return 0;
10 |    if(memo[pos][capacity] != -1) return memo[pos][capacity];
11 |    int r1 = solve(pos + 1, capacity); //dont take
12 |    int r2 = 0;
13 |    if(weight[pos] <= capacity){
14 |        r2 = solve(pos + 1, capacity - weight[pos]) + profit[pos]; //
15 |        take
16 |    }
17 |    return memo[pos][capacity] = max(r1, r2);
18 | }
19 | void reconstruct(int pos, int capacity){
20 |     if(pos == no_of_objects) return; //you have completed reconstruction
21 |     int r1 = memo[pos + 1][capacity]; //dont take
```

```

17     int r2 = 0;
18     if(weight[pos] <= capacity)r2 = memo[pos + 1][capacity - weight[pos]
    ]] + profit[pos]; //take
19     if(r1 > r2) {reconstruct(pos + 1, capacity);}
20     else{
21         cout << "Take_object_" << pos << endl;
22         reconstruct(pos + 1, capacity - weight[pos]) + profit[pos];
23     }
24 }

```

25 After executing reconstruct, it will print all those objects that give you the optimal solution. As you can see, at most no\_of\_objects calls will be made in the reconstruct function.

26 Similarly, you can reconstruct the solution of any DP greedily.

## 10.6 muajaja con j

```

1 #include <signal.h>
2 void divzero(int p){
3     while(true);}
4 void segm(int p){
5     exit(0);}
6 //in main
7 signal(SIGFPE, divzero);
8 signal(SIGSEGV, segm);

```

## Expandir pila

```

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

```

## 10.7 comparar doubles for noobs

```

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

```

## 10.8 infix to postfix

```

1 //infix to postfix with shunting yard, Halim interpretation
2 //plus eval function given a postfix return the result of the operation

```

```

3 //format: string like ( x o x ( x o x ) ) o=operation x=value
4 string s;
5 bool isOperator(string u){
6     return u=="+" || u=="-" || u=="*" || u=="/";
7 }
8 bool precede(string u){
9     if(u=="*" || u=="/")return true;
10    return false;
11 }
12 void solve(){
13     getline(cin,s);
14     stack<string>st;
15     vector<string>v;
16     stringstream ss;
17     ss<<s;
18     while(ss>>s){
19         if(isOperator(s)){
20             while(!st.empty()&&isOperator(st.top())&&precede(st.top())>=
                precede(s)){
21                 v.push_back(st.top());st.pop();
22             }
23             st.push(s);
24         }
25         else{
26             if(s=="("){
27                 st.push(s);
28             }
29             else{
30                 if(s=="){
31                     while(!st.empty()&&st.top()!="("){
32                         v.push_back(st.top());st.pop();
33                     }
34                     if(!st.empty()&&st.top()=="(")st.pop();
35                 }
36                 else {
37                     v.push_back(s);
38                 }
39             }
40         }
41     }
42     while(!st.empty()){
43         v.push_back(st.top());st.pop();
44     }

```



```

45 stack<double>stans;
46 double x;
47 for(string eva:v){
48     if(!isOperator(eva)){
49         stringstream nu;
50         nu<<eva;
51         nu>>x;
52         stans.push(x);
53     }
54     else{
55         double a=stans.top();stans.pop();
56         double b=stans.top();stans.pop();
57         if(eva=="*")b*=a;
58         if(eva=="/")b/=a;
59         if(eva=="+")b+=a;
60         if(eva=="-")b-=a;
61         stans.push(b);
62     }
63 }
64 cout<<fixed<<stans.top()<<"\n";
65 }

```

## 10.9 numeros romanos

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 map<int,string>cvt;
4
5 string aromano(int n){
6     cvt[1000] = "M";cvt[900] = "CM";cvt[500] = "D", cvt[400] = "CD";
7     cvt[100] = "C";cvt[90] = "XC"; cvt[50] = "L";
8     cvt[40] = "XL";cvt[10] = "X";cvt[9] = "IX";cvt[5] = "V"; cvt[4] = "IV"
9     ;
10    cvt[1] = "I";
11    string ans = "";
12    for(map<int,string>::reverse_iterator it = cvt.rbegin();it != cvt.rend
13        ();it++){
14        while(n >= it->first){
15            ans += it->second;
16            n -= it->first;
17        }
18    }
19    return ans;
20 }

```

```

18 map<string,int>crn;
19 int anumero(string R){
20     map<char, int> crn;
21     crn['I'] = 1;   crn['V'] = 5;   crn['X'] = 10;   crn['L'] = 50;
22     crn['C'] = 100; crn['D'] = 500; crn['M'] = 1000;
23     int value = 0;
24     for (int i = 0; R[i]; i++)
25         if (i + 1 < R.size() && crn[R[i]] < crn[R[i+1]]) {
26             value += crn[R[i+1]] - crn[R[i]];
27             i++;
28         }
29         else value += crn[R[i]];
30     return value;
31 }

```

## 10.10 get k-th permutacion

```

1 vector<int>v;
2 //finding the number of permutation 0.....n-1
3 int main()
4 {
5     string s;
6     while(getline(cin,s)){
7         stringstream ss;
8         ss<<s;
9         int pos=0,u;
10        v.clear();
11        while(ss>>u){
12            v.push_back(u-1);
13        }
14        vector<int>le(v.size(),0);
15        for(int i=0;i<v.size();i++){
16            for(int j=i+1;j<v.size();j++){
17                if(v[i]>v[j])le[i]++;
18            }
19        }
20        long long ans=0LL,fact=0LL,por=1LL;
21        for(int i=le.size()-1;i>=0;i--){
22            if(fact!=0LL)por*=fact;
23            fact++;
24            ans=ans+por*le[i];
25        }
26        cout<<ans+1<<"\n";

```

```

27     }
28     return 0;
29 }

```

### 10.11 sliding window

### 10.12 permutaciones de un dado

```

1 // izquierda, derecha, arriba, al frente, abajo, atras
2
3 int p[][6] = {
4     {0,1,2,3,4,5},
5     {0,1,3,4,5,2},
6     {0,1,4,5,2,3},
7     {0,1,5,2,3,4},
8     {1,0,2,5,4,3},
9     {1,0,3,2,5,4},
10    {1,0,4,3,2,5},
11    {1,0,5,4,3,2},
12    {2,4,5,1,3,0},
13    {2,4,1,3,0,5},
14    {2,4,3,0,5,1},
15    {2,4,0,5,1,3},
16    {3,5,2,1,4,0},
17    {3,5,1,4,0,2},
18    {3,5,4,0,2,1},
19    {3,5,0,2,1,4},
20    {4,2,5,0,3,1},
21    {4,2,0,3,1,5},
22    {4,2,3,1,5,0},
23    {4,2,1,5,0,3},
24    {5,3,2,0,4,1},
25    {5,3,0,4,1,2},
26    {5,3,4,1,2,0},
27    {5,3,1,2,0,4}
28 };

```

### 10.13 ternary search

### 10.14 liebre y el tortugo

### 10.15 como usar printf

### 10.16 java

### 10.17 python

### 10.18 template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /* no sirve en todos los jueces probar primero
5 int in() {
6     int num, c;
7     while((c = getchar_unlocked()) < '-');
8     num = c - '0';
9     while((c = getchar_unlocked()) >= '0') {
10         num = (num<<3) + (num<<1) + (c-'0');
11     }
12     return num;
13 }
14 */
15 int in(){int r=0,c;for(c=getchar();c<=32;c=getchar());if(c=='-') return
    -in();for(;c>32;r=(r<<1)+(r<<3)+c-'0',c=getchar());return r;}
16
17 #define DBG(x) cout << #x << ": " << x << "\n";
18 #define eb emplace_back
19 #define pb push_back
20 #define mp make_pair
21 #define mt make_tuple
22 int main(){
23     return 0;
24 }

```

### 10.19 file setup

```

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

```