

El BichiGol

UTN FRSF - El Rejunte

2017



Índice

1. C/C++	3
1.1. I/O	3
1.1.1. scanf Format Strings	3
1.1.2. printf Format Strings	3
2. Template del Rejunte	5
3. Estructuras de datos	5
3.1. Set Mejorado	5
3.2. Union Find	5
3.3. Hash Table	6
3.4. RMQ	6
3.4.1. RMQ (static)	6
3.4.2. RMQ (dynamic)	6
3.4.3. RMQ (lazy)	6
3.4.4. RMQ (persistente)	7
3.5. BIGInt	7

4. Algoritmos	8
4.1. Longest Increasing Subsequence	8
4.2. Mo's	9
5. Strings	9
5.1. KMP	9
5.2. Z function	9
5.3. Trie	10
5.4. Manacher	10
5.5. Aho Corasick	10
6. Geometría	11
6.1. Punto	11
6.2. Orden Radial de Puntos	11
6.3. Línea	11
6.4. Segmento	11
6.5. Rectángulo	12
6.6. Círculo	12
6.7. Área de polígono	13
6.8. Punto en polígono	13
6.9. Punto en Polígono Convexo	13
6.10. Chequeo de Convex	13
6.11. Convex Hull	13
6.12. Cortar polígono	13
6.13. Intersección de Círculos	14
6.14. Rotar Matriz	14
7. Matemática	15
7.1. Identidades	15
7.2. Ec. Característica	15
7.3. Teorema Chino del Resto	15
7.4. GCD & LCM	15

7.5. Euclides Extendido	15	10. Juegos	28
7.6. Combinatoria	15	10.1. Nim Game	28
7.7. Exponenciación de Matrices y Fibonacci	16	10.1.1. Misere Game	28
7.8. Operaciones Modulares	16	10.2. Ajedrez	29
7.9. Funciones de Primos	17	10.2.1. Non-Attacking N Queen	29
7.10. Phollard's Rho	18	11. Utils	29
7.11. Inversos	18	11.1. Convertir string a num e viceversa	29
7.12. Fracciones	18	11.2. Truquitos para entradas/salidas	29
7.13. Simpson	19	11.3. Comparación de Double	29
7.14. Tablas y cotas (Primos, Divisores, Factoriales, etc)	19	11.4. Iterar subconjuntos	29
7.15. Números Catalanés	19	11.5. Límites	29
7.15.1. Primeros 25 Catalanés	20	11.6. Mejorar Lectura de Enteros	29
8. Grafos	20	11.7. Tablita de relación de Complejidades	30
8.1. Dijkstra	20	11.8. Compilar C++11 con g++	30
8.2. Bellman-Ford	20	11.9. Build de C++11 para Sublime Text	30
8.3. Floyd-Warshall	21	11.10 Funciones Útiles	30
8.4. Kruskal	21		
8.5. Prim	21		
8.6. 2-SAT + Tarjan SCC	21		
8.7. Puntos de Articulación	22		
8.8. Least Common Ancestor + Climb	22		
8.9. Heavy Light Decomposition	22		
8.10. Centroid Decomposition	23		
8.11. Ciclo Euleriano	23		
8.12. Diámetro Árbol	23		
8.13. Componentes Biconexas y Puentes	23		
8.14. Hungarian	24		
8.15. Dynamic Connectivity	25		
9. Flow	26		
9.1. Edmond Karp	26		
9.2. Min Cut	26		
9.3. Push Relabel	26		
9.4. Dinic	27		
9.5. Min cost - Max flow	28		

1. C/C++

1.1. I/O

1.1.1. scanf Format Strings

%[*][width][length]specifier

spec	Tipo	Descripción
i	int	Dígitos dec. [0-9], oct. (0) [0-7], hexa (0x 0X) [0-9a-fA-F]. Con signo.
d, u	int, unsigned	Dígitos dec. [+0-9].
o	unsigned	Dígitos oct. [+0-7].
x	unsigned	Dígitos hex. [+0-9a-fA-F]. Prefijo 0x, 0X opcional.
f, e, g	float	Dígitos dec. c/punto flotante [+-.0-9]. Prefijo 0x, 0X y sufijo e, E opcionales.
c, [width]c	char, char*	Siguiente carácter. Lee width chars y los almacena contiguamente. No agrega \0.
s	char*	Secuencia de chars hasta primer espacio. Agrega \0.
p	void*	Secuencia de chars que representa un puntero.
[chars]	Scanset, char*	Caracteres especificados entre corchetes.] debe ser primero en la lista, - primero o último. Agrega \0
[^chars]	!Scanset, char*	Caracteres no especificados entre corchetes.
n	int	No consume entrada. Almacena el número de chars leídos hasta el momento.
%		%% consume un %

sub-specifier	Descripción
*	Indica que se leerá el dato pero se ignorará. No necesita argumento.
width	Cantidad máxima de caracteres a leer.
length	Uno de hh, h, l, ll, j, z, t, L. Ver tabla siguiente.

length	d i	u o x
(none)	int*	unsigned int*
hh	signed char*	unsigned char*
h	short int*	unsigned short int*
l	long int*	unsigned long int*
ll	long long int*	unsigned long long int*

Continuación		
length	d i	u o x
j	intmax_t*	uintmax_t*
z	size_t*	size_t*
t	ptrdiff_t*	ptrdiff_t*
L		

length	f e g a	c s [] [^]	p	n
(none)	float*	char*	void**	int*
hh				signed char*
h				short int*
l	double*	wchar_t*		long int*
ll				long long int*
j				intmax_t*
z				size_t*
t				ptrdiff_t*
L	long double*			

1.1.2. printf Format Strings

%[flags][width][.precision][length]specifier

specifier	Descripción	Ejemplo
d or i	Entero decimal con signo	392
u	Entero decimal sin signo	7235
o	Entero octal sin signo	610
x	Entero hexadecimal sin signo	7fa
X	Entero hexadecimal sin signo (mayúsculas)	7FA
f	Decimal punto flotante (minúsculas)	392.65
F	Decimal punto flotante (mayúsculas)	392.65
e	Notación científica (mantisa/exponente), (minúsculas)	3.9265e+2
E	Notación científica (mantisa/exponente), (mayúsculas)	3.9265E+2
g	Utilizar la representación más corta: %e ó %f	392.65
G	Utilizar la representación más corta: %E ó %F	392.65
a	Hexadecimal punto flotante (minúsculas)	-0xc.90fep-2
A	Hexadecimal punto flotante (mayúsculas)	-0XC.90FEP-2
c	Caracter	a
s	String de caracteres	sample

Continuación		
specifier	Descripción	Ejemplo
p	Dirección de puntero	b8000000
n	No imprime nada. El argumento debe ser int*, almacena el número de caracteres imprimidos hasta el momento.	
%	Un % seguido de otro % imprime un solo %	%

flag	Descripción
-	Justificación a la izquierda dentro del campo width (ver width sub-specifier).
+	Forza a preceder el resultado de texttt+ o texttt-.
(espacio)	Si no se va a escribir un signo, se inserta un espacio antes del valor.
#	Usado con o, x, X specifiers el valor es precedido por 0, 0x, 0X respectivamente para valores distintos de 0.
0	Rellena el número con texttt0 a la izquierda en lugar de espacios cuando se especifica width.

width	Descripción
(número)	Número mínimo de caracteres a imprimir. Si el valor es menor que número, el resultado es rellando con espacios. Si el valor es mayor, no es truncado.
*	No se especifica width, pero se agrega un argumento entero precediendo al argumento a ser formateado. Ej. printf(“---%d----\n”, 3, 2); ⇒ “---- 5----”.

precision	Descripción
.(número)	Para d, i, o, u, x, X: número mínimo de dígitos a imprimir. Si el valor es más chico que número se rellena con 0. Para a, A, e, E, f, F: número de dígitos a imprimir después de la coma (default 6). Para g, G: Número máximo de cifras significativas a imprimir. Para s: Número máximo de caracteres a imprimir. Trunca.
.*	No se especifica precision pero se agrega un argumento entero precediendo al argumento a ser formateado.

length	d i	u o x X
(none)	int	unsigned int
hh	signed char	unsigned char
h	short int	unsigned short int
l	long int	unsigned long int
ll	long long int	unsigned long long int

Continuación		
length	d i	u o x X
j	intmax_t	uintmax_t
z	size_t	size_t
t	ptrdiff_t	ptrdiff_t
L		

length	f F e E g G a A	c	s	p	n
(none)	double	int	char*	void*	int*
hh					signed char*
h					short int*
l		wint_t	wchar_t*		long int*
ll					long long int*
j					intmax_t*
z					size_t*
t					ptrdiff_t*
L	long double				

2. Template del Rejunte

```

1 #include <bits/stdc++.h>
2 #define sqr(a) ((a)*(a))
3 #define rsz resize
4 #define forr(i,a,b) for(int i=(a);i<(b);i++)
5 #define forn(i,n) forr(i,0,n)
6 #define dforr(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 mp make_pair
12 #define lb lower_bound
13 #define ub upper_bound
14 #define fst first
15 #define snd second
16 #define PI 3.1415926535897932384626
17
18 using namespace std;
19
20 typedef long long ll;
21 typedef pair<int,int> ii;
22 typedef vector<int> vi;
23 typedef vector<ii> vii;
24
25 int main()
26 {
27     //freopen("input","r",stdin);
28     //freopen("output","w",stdout);
29     ios::sync_with_stdio(false);
30     cin.tie(NULL);
31     cout.tie(NULL);
32     return 0;
33 }
```

3. Estructuras de datos

3.1. Set Mejorado

Esto solo compila en C++11.

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 //<key,mapped type,comparator,...>
5 typedef tree<int,null_type,less<int>,rb_tree_tag,
6     tree_order_statistics_node_update> ordered_set;
7 //find_by_order(i) devuelve iterador al i-esimo elemento
8 //order_of_key(k): devuelve la pos del lower bound de k
9 //Ej: 12, 100, 505, 1000, 10000.
10 //order_of_key(10) == 0, order_of_key(100) == 1,
11 //order_of_key(707) == 3, order_of_key(9999999) == 5
```

3.2. Union Find

```

1 struct UnionFind{
2     vector<int> f,setSize; //the array f contains the parent of each node
3     int cantSets;
4     void init(int n)
5     {
6         f.clear(); setSize.clear();
7         cantSets=n;
8         f.rsz(n,-1);
9         setSize.rsz(n,1);
10    }
11    int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));} //O(1)
12    bool join(int i,int j) //devuelve true si ya estaban juntos
13    {
14        bool con=comp(i)==comp(j);
15        if(!con)
16        {
17            cantSets--;
18            setSize[comp(j)]+=setSize[comp(i)];
19            setSize[comp(i)]=setSize[comp(j)]; //no suma, solo asigna
20            f[comp(i)]=comp(j);
21        }
22        return con;
23    }
24 };
```

3.3. Hash Table

```

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

```

3.4. RMQ

3.4.1. 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(n log n)
10        int mp = 31-__builtin_clz(n);
11        forn(p, mp) forn(x, n-(1<<p))
12            vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
13    };

```

3.4.2. RMQ (dynamic)

```

1 //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
  sobre el rango [i, j].
2 #define MAXN 100000
3 #define operacion(x, y) max(x, y)

```

```

4 const int neutro=0;
5 struct RMQ{
6     int sz;
7     tipo t[4*MAXN];
8     tipo &operator[](int p){return t[sz+p];}
9     void init(int n) { //O(n log n)
10        sz = 1 << (32-__builtin_clz(n));
11        forn(i, 2*sz) t[i]=neutro;
12    }
13    void upcall() { //O(n)
14        dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);
15    }
16    tipo get(int i, int j){return get(i, j, 1, 0, sz);}
17    tipo get(int i, int j, int n, int a, int b) { //O(log n)
18        if(j<=a || i>=b) return neutro;
19        if(i<=a && b<=j) return t[n];
20        int c=(a+b)/2;
21        return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
22    }
23    void set(int p, tipo val) { //O(log n)
24        for(p+=sz; p>0 && t[p]!=val;){
25            t[p]=val;
26            p/=2;
27            val=operacion(t[p*2], t[p*2+1]);
28        }
29    }
30 }rmq;
31 //Usage:
32 cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.upcall();

```

3.4.3. RMQ (lazy)

```

1 //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
  sobre el rango [i, j].
2 typedef int Elem; //Elem de los elementos del arreglo
3 typedef int Alt; //Elem de la alteracion
4 #define operacion(x, y) x+y
5 const Elem neutro=0; const Alt neutro2=0;
6 #define MAXN 100000 //Cambiar segun el N del problema
7 struct RMQ{
8     int sz;
9     Elem t[4*MAXN];
10    Alt dirty[4*MAXN]; //las alteraciones pueden ser de distinto Elem
11    Elem &operator[](int p){return t[sz+p];}
12    void init(int n) { //O(n log n)
13        sz = 1 << (32-__builtin_clz(n));
14        forn(i, 2*sz) t[i]=neutro;
15        forn(i, 2*sz) dirty[i]=neutro2;
16    }
17    void push(int n, int a, int b) { //propaga el dirty a sus hijos
18        if(dirty[n]!=0){
19            t[n]+=dirty[n]*(b-a); //altera el nodo

```

```

20     if(n<sz){
21         dirty[2*n]+=dirty[n];
22         dirty[2*n+1]+=dirty[n];
23     }
24     dirty[n]=0;
25 }
26 }
27 Elem get(int i, int j, int n, int a, int b){//O(lgn)
28     if(j<=a || i>=b) return neutro;
29     push(n, a, b);//corrige el valor antes de usarlo
30     if(i<=a && b<=j) return t[n];
31     int c=(a+b)/2;
32     return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
33 }
34 Elem get(int i, int j){return get(i,j,1,0,sz);}
35 //altera los valores en [i, j) con una alteracion de val
36 void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
37     push(n, a, b);
38     if(j<=a || i>=b) return;
39     if(i<=a && b<=j){
40         dirty[n]+=val;
41         push(n, a, b);
42         return;
43     }
44     int c=(a+b)/2;
45     alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
46     t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
47 }
48 void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
49 }rmq;

```

3.4.4. RMQ (persistente)

```

1 typedef int tipo;
2 tipo oper(const tipo &a, const tipo &b){
3     return a+b;
4 }
5 struct node{
6     tipo v; node *l,*r;
7     node(tipo v):v(v), l(NULL), r(NULL) {}
8     node(node *l, node *r) : l(l), r(r){
9         if(!l) v=r->v;
10        else if(!r) v=l->v;
11        else v=oper(l->v, r->v);
12    }
13 };
14 node *build (tipo *a, int tl, int tr) { //modificar para que tome tipo a
15     if (tl+1==tr) return new node(a[tl]);
16     int tm=(tl + tr)>>1;
17     return new node(build(a, tl, tm), build(a, tm, tr));
18 }

```

```

19 node *update(int pos, int new_val, node *t, int tl, int tr){
20     if (tl+1==tr) return new node(new_val);
21     int tm=(tl+tr)>>1;
22     if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
23     else return new node(t->l, update(pos, new_val, t->r, tm, tr));
24 }
25 tipo get(int l, int r, node *t, int tl, int tr){
26     if(l==tl && tr==r) return t->v;
27     int tm=(tl + tr)>>1;
28     if(r<=tm) return get(l, r, t->l, tl, tm);
29     else if(l>=tm) return get(l, r, t->r, tm, tr);
30     return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
31 }

```

3.5. BIGInt

```

1 #define BASEXP 6
2 #define BASE 1000000
3 #define LMAX 1000
4 struct bint{
5     int l;
6     ll n[LMAX];
7     bint(ll x=0){
8         l=1;
9         forn(i, LMAX){
10             if (x) l=i+1;
11             n[i]=x%BASE;
12             x/=BASE;
13         }
14     }
15 }
16 bint(string x){
17     l=(x.size()-1)/BASEXP+1;
18     fill(n, n+LMAX, 0);
19     ll r=1;
20     forn(i, sz(x)){
21         n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
22         r*=10; if(r==BASE) r=1;
23     }
24 }
25 void out(){
26     cout << n[l-1];
27     dform(i, l-1) printf("%6.6llu", n[i]); //6=BASEXP!
28 }
29 void invar(){
30     fill(n+1, n+LMAX, 0);
31     while(l>1 && !n[l-1]) l--;
32 }
33 };
34 bint operator+(const bint&a, const bint&b){
35     bint c;

```

```

36     c.l = max(a.l, b.l);
37     ll q = 0;
38     forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
39     if(q) c.n[c.l++] = q;
40     c.invar();
41     return c;
42 }
43 pair<bint, bool> lresta(const bint& a, const bint& b)    // c = a - b
44 {
45     bint c;
46     c.l = max(a.l, b.l);
47     ll q = 0;
48     forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/BASE
49         -1;
49     c.invar();
50     return make_pair(c, !q);
51 }
52 bint& operator==(const bint& a, const bint& b){return a==lresta(a, b).first;}
53 bint& operator-(const bint& a, const bint& b){return lresta(a, b).first;}
54 bool operator<(const bint& a, const bint& b){return !lresta(a, b).second;}
55 bool operator<=(const bint& a, const bint& b){return lresta(b, a).second;}
56 bool operator==(const bint& a, const bint& b){return a <= b && b <= a;}
57 bint operator*(const bint& a, ll b){
58     bint c;
59     ll q = 0;
60     forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
61     c.l = a.l;
62     while(q) c.n[c.l++] = q %BASE, q/=BASE;
63     c.invar();
64     return c;
65 }
66 bint operator*(const bint& a, const bint& b){
67     bint c;
68     c.l = a.l+b.l;
69     fill(c.n, c.n+b.l, 0);
70     forn(i, a.l){
71         ll q = 0;
72         forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q/=
73             BASE;
73         c.n[i+b.l] = q;
74     }
75     c.invar();
76     return c;
77 }
78 pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
79     bint c;
80     ll rm = 0;
81     dfor(i, a.l){
82         rm = rm * BASE + a.n[i];
83         c.n[i] = rm / b;
84         rm %= b;
85     }
86     c.l = a.l;

```

```

87     c.invar();
88     return make_pair(c, rm);
89 }
90 bint operator/(const bint& a, ll b){return ldiv(a, b).first;}
91 ll operator%(const bint& a, ll b){return ldiv(a, b).second;}
92 pair<bint, bint> ldiv(const bint& a, const bint& b){
93     bint c;
94     bint rm = 0;
95     dfor(i, a.l){
96         if (rm.l==1 && !rm.n[0])
97             rm.n[0] = a.n[i];
98         else{
99             dfor(j, rm.l) rm.n[j+1] = rm.n[j];
100             rm.n[0] = a.n[i];
101             rm.l++;
102         }
103         ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
104         ll u = q / (b.n[b.l-1] + 1);
105         ll v = q / b.n[b.l-1] + 1;
106         while (u < v-1){
107             ll m = (u+v)/2;
108             if (b*m <= rm) u = m;
109             else v = m;
110         }
111         c.n[i]=u;
112         rm-=b*u;
113     }
114     c.l=a.l;
115     c.invar();
116     return make_pair(c, rm);
117 }
118 bint operator/(const bint& a, const bint& b){return ldiv(a, b).first;}
119 bint operator%(const bint& a, const bint& b){return ldiv(a, b).second;}

```

4. Algoritmos

4.1. Longest Increasing Subsequence

```

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
3 //Solution:Min number of colors=Length of the longest increasing subsequence
4 int N, a[MAXN]; //secuencia y su longitud
5 ii d[MAXN+1]; //d[i]=ultimo valor de la subsecuencia de tamano i
6 int p[MAXN]; //padres
7 vector<int> R; //respuesta
8 void rec(int i){
9     if(i==1) return;
10    R.push_back(a[i]);
11    rec(p[i]);

```



```

12 }
13 int lis() { // O(n log n)
14     d[0] = ii(-INF, -1); forn(i, N) d[i+1] = ii(INF, -1);
15     forn(i, N) {
16         int j = upper_bound(d, d+N+1, ii(a[i], INF)) - d;
17         if (d[j-1].first < a[i] && a[i] < d[j].first) {
18             p[i] = d[j-1].second;
19             d[j] = ii(a[i], i);
20         }
21     }
22     R.clear();
23     dforn(i, N+1) if (d[i].first != INF) {
24         rec(d[i].second); // reconstruir
25         reverse(R.begin(), R.end());
26         return i; // longitud
27     }
28     return 0;
29 }

```

4.2. Mo's

$$O(q * \sqrt{n})$$

```

1 int n, sq;
2 struct Qu { // queries [l, r]
3     // intervalos cerrado abiertos !!! importante!!
4     int l, r, id;
5 } qs[MAXN];
6 int ans[MAXN], curans; // ans[i] = ans to ith query
7 bool bymos(const Qu &a, const Qu &b) {
8     if (a.l/sq != b.l/sq) return a.l < b.l;
9     return (a.l/sq & 1 ? a.r < b.r : a.r > b.r);
10 }
11 void mos() {
12     forn(i, t) qs[i].id = i;
13     sort(qs, qs+t, bymos);
14     int cl=0, cr=0;
15     sq = sqrt(n);
16     curans = 0;
17     forn(i, t) { // intervalos cerrado abiertos !!! importante!!
18         Qu &q = qs[i];
19         while (cl > q.l) add(--cl);
20         while (cr < q.r) add(cr++);
21         while (cl < q.l) remove(cl++);
22         while (cr > q.r) remove(--cr);
23         ans[q.id] = curans;
24     }
25 }

```

5. Strings

5.1. KMP

```

1 vector<int> b; // back table b[i] maximo borde de [0..i)
2 void kmppre(string &P) // by gabina with love
3 {
4     b.clear();
5     b.resize(P.size());
6     int i = 0, j = -1; b[0] = -1;
7     while (i < sz(P))
8     {
9         while (j >= 0 && P[i] != P[j]) j = b[j];
10        i++, j++;
11        b[i] = j;
12    }
13 }
14 void kmp(string &T, string &P) // Text, Pattern -- O(|T| + |P|)
15 {
16     kmppre(P);
17     int i = 0, j = 0;
18     while (i < sz(T))
19     {
20         while (j >= 0 && T[i] != P[j]) j = b[j];
21         i++, j++;
22         if (j == sz(P))
23         {
24             // P encontrado en T empezando en [i-j, i)
25             j = b[j];
26         }
27     }
28 }

```

5.2. Z function

```

1 // z[i] = length of longest substring starting from s[i] that is prefix of s
2 vector<int> z;
3 void zFunction(string &s)
4 {
5     int n = s.size();
6     for (int i = 1, l = 0, r = 0; i < n; i++)
7     {
8         if (i <= r)
9             z[i] = min(r - i + 1, z[i - l]);
10        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
11            z[i]++;
12        if (i + z[i] - 1 > r)
13            l = i, r = i + z[i] - 1;
14    }
15 }

```

```

16 void match(string &T, string &P) //Text, Pattern -- O(|T|+|P|)
17 {
18     string s=P;
19     s+='$'; //here append a character that is not present in T
20     s.append(T);
21     z.clear();
22     z.rsz(s.size(),0);
23     zFunction(s);
24     forr(i,P.size()+1,s.size())
25         if(z[i]==P.size()) //match found, idx = i-P.size()-1
26 }

```

5.3. Trie

```

1 struct trie{
2     map<char, trie> m;
3     void add(const string &s, int p=0)
4     {
5         if(s[p]) m[s[p]].add(s, p+1);
6     }
7     void dfs()
8     {
9         //Do stuff
10        forall(it, m)
11            it->second.dfs();
12    }
13 };

```

5.4. Manacher

```

1 string s;
2 int d1[MAXN]; //d1[i]=long del maximo palindromo impar con centro en i
3 int d2[MAXN]; //d2[i]=analogo pero para longitud par
4 //0 1 2 3 4
5 //a a b a a <--d1[2]=3
6 //a a a a <--d2[2]=2 (estan uno antes)
7 void manacher() // O(|S|) - find longest palindromic substring
8 {
9     int l=0, r=-1, n=sz(s);
10    forn(i, n)
11    {
12        int k=(i>r? 1 : min(d1[l+r-i], r-i));
13        while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) ++k;
14        d1[i] = k--;
15        if(i+k > r) l=i-k, r=i+k;
16    }
17    l=0, r=-1;
18    forn(i, n)

```

```

19 {
20     int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
21     while(i+k-1<n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
22     d2[i] = --k;
23     if(i+k-1 > r) l=i-k, r=i+k-1;
24 }
25 }

```

5.5. 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 es
6     //hoja
7     Trie *padre, *link, *nxthoja;
8     char pch; //caracter que conecta con padre
9     //Trie(): tran(), idhoja(), padre(), link() {}
10    //coment linea de arriba porque me daba errores usarla.
11    void insert(const string &s, int id=1, int p=0) //id>0!!!
12    {
13        if(p<sz(s))
14        {
15            Trie &ch=next[s[p]];
16            tran[(int)s[p]]=&ch;
17            ch.padre=this, ch.pch=s[p];
18            ch.insert(s, id, p+1);
19        }
20        else idhoja=id, szhoja=sz(s);
21    }
22    Trie* get_link()
23    {
24        if(!link)
25        {
26            if(!padre) link=this; //es la raiz
27            else if(!padre->padre) link=padre; //hijo de la raiz
28            else link=padre->get_link()->get_tran(pch);
29        }
30        return link;
31    }
32    Trie* get_tran(int c)
33    {
34        if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
35        return tran[c];
36    }
37    Trie *get_nxthoja()
38    {
39        if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
40        return nxthoja;
41    }

```

```

41 void print(int p)
42 {
43     if(idhoja) cout << "found " << idhoja << " at position " << p-szhoja <<
        endl;
44     if(get_nxthoja()) get_nxthoja()->print(p);
45 }
46 void matching(const string &s, int p=0) //O(|s| + tamaño palabras)
47 {
48     print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1);
49 }
50 };

```

6. Geometría

6.1. Punto

```

1 struct pto{
2     double x, y;
3     pto(double x=0, double y=0):x(x),y(y){}
4     pto operator+(pto a){return pto(x+a.x, y+a.y);}
5     pto operator-(pto a){return pto(x-a.x, y-a.y);}
6     pto operator+(double a){return pto(x+a, y+a);}
7     pto operator*(double a){return pto(x*a, y*a);}
8     pto operator/(double a){return pto(x/a, y/a);}
9     //dot product, producto interno:
10    double operator*(pto a){return x*a.x+y*a.y;}
11    //module of the cross product or vectorial product:
12    //if a is less than 180 clockwise from b, a^b>0
13    double operator^(pto a){return x*a.y-y*a.x;}
14    //returns true if this is at the left side of line qr
15    bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
16    bool operator<(const pto &a) const{return x<a.x-EPS || (abs(x-a.x)<EPS &&
        y<a.y-EPS);}
17    bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
18    double norm(){return sqrt(x*x+y*y);}
19    double norm_sq(){return x*x+y*y;}
20 };
21 double dist(pto a, pto b){return (b-a).norm();}
22 typedef pto vec;
23
24 double angle(pto a, pto o, pto b){
25     pto oa=a-o, ob=b-o;
26     return atan2(oa^ob, oa*ob);}
27
28 //rotate p by theta rads CCW w.r.t. origin (0,0)
29 pto rotate(pto p, double theta){
30     return pto(p.x*cos(theta)-p.y*sin(theta),
31         p.x*sin(theta)+p.y*cos(theta));
32 }

```

6.2. Orden Radial de Puntos

```

1 struct Cmp{//orden total de puntos alrededor de un punto r
2     pto r;
3     Cmp(pto r):r(r) {}
4     int cuad(const pto &a) const{
5         if(a.x > 0 && a.y >= 0)return 0;
6         if(a.x <= 0 && a.y > 0)return 1;
7         if(a.x < 0 && a.y <= 0)return 2;
8         if(a.x >= 0 && a.y < 0)return 3;
9         assert(a.x ==0 && a.y==0);
10        return -1;
11    }
12    bool cmp(const pto&p1, const pto&p2)const{
13        int c1 = cuad(p1), c2 = cuad(p2);
14        if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
15        else return c1 < c2;
16    }
17    bool operator()(const pto&p1, const pto&p2) const{
18        return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
19    }
20 };

```

6.3. Línea

```

1 int sgn(ll x){return x<0? -1 : !!x;}
2 struct line{
3     line() {}
4     double a,b,c;//Ax+By=C
5     //pto MUST store float coordinates!
6     line(double a, double b, double c):a(a),b(b),c(c){}
7     line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
8     int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
9 };
10 bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
11 pto inter(line l1, line l2){//intersection
12     double det=l1.a*l2.b-l2.a*l1.b;
13     if(abs(det)<EPS) return pto(INF, INF);//parallels
14     return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
15 }

```

6.4. Segmento

```

1 struct segm{
2     pto s,f;
3     segm(pto s, pto f):s(s), f(f) {}
4     pto closest(pto p) {//use for dist to point
5         double l2 = dist_sq(s, f);

```

```

6   if(l2==0.) return s;
7   double t = ((p-s)*(f-s))/l2;
8   if (t<0.) return s; //not write if is a line
9   else if(t>1.) return f; //not write if is a line
10  return s+((f-s)*t);
11  }
12  bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS;}
13  };
14
15  //NOTA: Si los segmentos son colineales solo devuelve un punto de
16  //interseccion
17  pto inter(segm s1, segm s2){
18      if(s1.inside(s2.s)) return s2.s; //Fix cuando son colineales
19      if(s1.inside(s2.f)) return s2.f; //Fix cuando son colineales
20      pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
21      if(s1.inside(r) && s2.inside(r)) return r;
22      return pto(INF, INF);
23  }

```

6.5. Rectangulo

```

1  struct rect{
2      //lower-left and upper-right corners
3      pto lw, up;
4  };
5  //returns if there's an intersection and stores it in r
6  bool inter(rect a, rect b, rect &r){
7      r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
8      r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
9      //check case when only a edge is common
10     return r.lw.x<r.up.x && r.lw.y<r.up.y;
11 }

```

6.6. Circulo

```

1  vec perp(vec v){return vec(-v.y, v.x);}
2  line bisector(pto x, pto y){
3      line l=line(x, y); pto m=(x+y)/2;
4      return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5  }
6  struct Circle{
7      pto o;
8      double r;
9      Circle(pto x, pto y, pto z){
10         o=inter(bisector(x, y), bisector(y, z));
11         r=dist(o, x);
12     }
13     pair<pto, pto> ptosTang(pto p){

```

```

14     pto m=(p+o)/2;
15     tipo d=dist(o, m);
16     tipo a=r*r/(2*d);
17     tipo h=sqrt(r*r-a*a);
18     pto m2=o+(m-o)*a/d;
19     vec per=perp(m-o)/d;
20     return make_pair(m2-per*h, m2+per*h);
21 }
22 };
23 //finds the center of the circle containing p1 and p2 with radius r
24 //as there may be two solutions swap p1, p2 to get the other
25 bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
26     double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
27     if(det<0) return false;
28     c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
29     return true;
30 }
31 #define sqr(a) ((a)*(a))
32 #define feq(a,b) (fabs((a)-(b))<EPS)
33 pair<tipo, tipo> ecCua(tipo a, tipo b, tipo c){ //a*x*x+b*x+c=0
34     tipo dx = sqrt(b*b-4.0*a*c);
35     return make_pair((-b + dx)/(2.0*a), (-b - dx)/(2.0*a));
36 }
37 pair<pto, pto> interCL(Circle c, line l){
38     bool sw=false;
39     if((sw=feq(0, l.b))){
40         swap(l.a, l.b);
41         swap(c.o.x, c.o.y);
42     }
43     pair<tipo, tipo> rc = ecCua(
44         sqr(l.a)+sqr(l.b),
45         2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
46         sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.o.y
47     );
48     pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
49                     pto(rc.second, (l.c - l.a * rc.second) / l.b) );
50     if(sw){
51         swap(p.first.x, p.first.y);
52         swap(p.second.x, p.second.y);
53     }
54     return p;
55 }
56 pair<pto, pto> interCC(Circle c1, Circle c2){
57     line l;
58     l.a = c1.o.x-c2.o.x;
59     l.b = c1.o.y-c2.o.y;
60     l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
61           -sqr(c2.o.y))/2.0;
62     return interCL(c1, l);
63 }

```

6.7. Area de poligono

```

1 double area(vector<pto> &p){//O(sz(p))
2     double area=0;
3     forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4     //if points are in clockwise order then area is negative
5     return abs(area)/2;
6 }
7 //Area ellipse = M_PI*a*b where a and b are the semi axis lengths
8 //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2

```

6.8. Punto en poligono

```

1 //checks if v is inside of P, using ray casting
2 //works with convex and concave.
3 //excludes boundaries, handle it separately using segment.inside()
4 bool inPolygon(pto v, vector<pto>& P) {
5     bool c = false;
6     forn(i, sz(P)){
7         int j=(i+1)%sz(P);
8         if((P[j].y>v.y) != (P[i].y > v.y) &&
9         (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
10            c = !c;
11     }
12     return c;
13 }

```

6.9. Punto en Poligono Convexo

 $O(\log n)$

```

1 void normalize(vector<pto> &pt) //delete collinear points first!
2 {
3     //this makes it clockwise:
4     if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
5     int n=sz(pt), pi=0;
6     forn(i, n)
7         if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
8             pi=i;
9     vector<pto> shift(n); //puts pi as first point
10    forn(i, n) shift[i]=pt[(pi+i)%n];
11    pt.swap(shift);
12 }
13 bool inPolygon(pto p, const vector<pto> &pt)
14 {
15     //call normalize first!
16     if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
17     int a=1, b=sz(pt)-1;
18     while(b-a>1)

```

```

19 {
20     int c=(a+b)/2;
21     if(!p.left(pt[0], pt[c])) a=c;
22     else b=c;
23 }
24 return !p.left(pt[a], pt[a+1]);
25 }

```

6.10. Chequeo de Convex

```

1
2 bool isConvex(vector<int> &p){//O(N), delete collinear points!
3     int N=sz(p);
4     if(N<3) return false;
5     bool isLeft=p[0].left(p[1], p[2]);
6     forn(i, 1, N)
7         if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
8             return false;
9     return true; }

```

6.11. Convex Hull

```

1 //stores convex hull of P in S, CCW order
2 //left must return >=0 to delete collinear points!
3 void CH(vector<pto>& P, vector<pto> &S){
4     S.clear();
5     sort(P.begin(), P.end()); //first x, then y
6     forn(i, sz(P)){ //lower hull
7         while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
8         S.pb(P[i]);
9     }
10    S.pop_back();
11    int k=sz(S);
12    dforn(i, sz(P)){ //upper hull
13        while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
14        S.pb(P[i]);
15    }
16    S.pop_back();
17 }

```

6.12. 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 }

```

6.13. Intersección de Círculos

```

1  struct event {
2      double x; int t;
3      event(double xx, int tt) : x(xx), t(tt) {}
4      bool operator <(const event &o) const { return x < o.x; }
5  };
6  typedef vector<Circle> VC;
7  typedef vector<event> VE;
8  int n;
9  double cuenta(VE &v, double A, double B)
10 {
11     sort(v.begin(), v.end());
12     double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
13     int contador = 0;
14     forn(i, sz(v))
15     { //interseccion de todos (contador == n), union de todos (contador > 0)
16         //conjunto de puntos cubierto por exacta k Círculos (contador == k)
17         if (contador == n) res += v[i].x - lx;
18         contador += v[i].t, lx = v[i].x;
19     }
20     return res;
21 }
22 // Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
23 inline double primitiva(double x, double r)
24 {
25     if (x >= r) return r*r*M_PI/4.0;
26     if (x <= -r) return -r*r*M_PI/4.0;
27     double raiz = sqrt(r*r-x*x);
28     return 0.5 * (x * raiz + r*r*atan(x/raiz));
29 }
30 double interCircle(VC &v)
31 {
32     vector<double> p; p.reserve(v.size() * (v.size() + 2));
33     forn(i, sz(v)) p.push_back(v[i].c.x + v[i].r), p.push_back(v[i].c.x - v[i].r);
34     forn(i, sz(v)) forn(j, i)
35     {
36         Circle &a = v[i], b = v[j];
37         double d = (a.c - b.c).norm();
38         if (fabs(a.r - b.r) < d && d < a.r + b.r)
39         {
40             double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r));

```

```

41         pto vec = (b.c - a.c) * (a.r / d);
42         p.pb((a.c + rotate(vec, alfa)).x), p.pb((a.c + rotate(vec, -alfa)).x);
43     }
44 }
45 sort(p.begin(), p.end());
46 double res = 0.0;
47 forn(i, sz(p)-1)
48 {
49     const double A = p[i], B = p[i+1];
50     VE ve; ve.reserve(2 * v.size());
51     forn(j, sz(v))
52     {
53         const Circle &c = v[j];
54         double arco = primitiva(B-c.c.x, c.r) - primitiva(A-c.c.x, c.r);
55         double base = c.c.y * (B-A);
56         ve.push_back(event(base + arco, -1));
57         ve.push_back(event(base - arco, 1));
58     }
59     res += cuenta(ve, A, B);
60 }
61 return res;
62 }

```

6.14. Rotar Matriz

```

1 //rotates matrix t 90 degrees clockwise
2 //using auxiliary matrix t2(faster)
3 void rotate()
4 {
5     forn(x, n) forn(y, n)
6         t2[n-y-1][x]=t[x][y];
7     memcpy(t, t2, sizeof(t));
8 }

```

7. Matemática

7.1. Identidades

$$\sum_{i=0}^n \binom{n}{i} = 2^n$$

$$\sum_{i=0}^n i \binom{n}{i} = n * 2^{n-1}$$

$$\sum_{i=m}^n i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$

$$\sum_{i=0}^n i = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

$$\sum_{i=0}^n i(i-1) = \frac{8}{6} \left(\frac{n}{2}\right) \left(\frac{n}{2} + 1\right) (n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par}$$

$$\sum_{i=0}^n i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^n i]^2$$

$$\sum_{i=0}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$

$$\sum_{i=0}^n i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^p \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1}$$

$$r = e - v + k + 1$$

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

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

7.2. Ec. Característica

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

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

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.

7.3. Teorema Chino del Resto

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

```

1 //Chinese remainder theorem (special case): find z such that
2 //z % m1 = r1, z % m2 = r2. Here, z is unique modulo M = lcm(m1, m2).
3 //Return (z, M). On failure, M = -1.
4 ii chinese_remainder_theorem(int m1, int r1, int m2, int r2)
5 { //{xx,yy,d} son variables globales usadas en extendedEuclid
6   extendedEuclid(m1, m2);
7   if (r1%d != r2%d) return make_pair(0,-1);

```

```

8   return mp(sumMod(xx*r2*m1, yy*r1*m2, m1*m2) / d, m1*m2 / d);
9 }
10 //Chinese remainder theorem: find z such that z % m[i] = r[i] for all i.
11 //Note that the solution is unique modulo M = lcm_i (m[i]).
12 //Return (z, M). On failure, M = -1.
13 //Note that we do not require the a[i]'s to be relatively prime.
14 ii chinese_remainder_theorem(const vector<int> &m, const vector<int> &r)
15 {
16   ii ret=mp(r[0], m[0]);
17   forr(i,1,m.size())
18   {
19     ret=chinese_remainder_theorem(ret.snd, ret.fst, m[i], r[i]);
20     if (ret.snd==-1) break;
21   }
22   return ret;
23 }

```

7.4. GCD & LCM

```

1 int gcd(int a, int b) {return b? gcd(b,a%b) : a;}
2 int lcm(int a, int b) {return a*(b/gcd(a,b));}

```

7.5. Euclides Extendido

```

1 //ecuacin diofntica lineal
2 //sea d=gcd(a,b); la ecuacin a * x + b * y = c tiene soluciones enteras si
3 //d|c. La siguiente funcin nos sirve para esto. De forma general ser :
4 //x = x0 + (b/d)n      x0 = xx*c/d
5 //y = y0 - (a/d)n      y0 = yy*c/d
6 ll xx,yy,d;
7 void extendedEuclid(ll a, ll b) //a * xx + b * yy = d
8 {
9   if (!b) {xx=1; yy=0; d=a; return;}
10  extendedEuclid (b,a%b);
11  ll x1=yy;
12  ll y1=xx-(a/b)*yy;
13  xx=x1; yy=y1;
14 }

```

7.6. Combinatoria

```

1 void cargarComb() //O(MAXN^2)
2 {
3   forn(i, MAXN+1) //comb[i][k]=i tomados de a k = i!/(k!*(i-k)!)
4   {
5     comb[0][i]=0;

```

```

6     comb[i][0]=comb[i][i]=1;
7     forr(k, 1, i) comb[i][k]=(comb[i-1][k-1]+comb[i-1][k]) %MOD;
8 }
9 }
10 ll lucas (ll n, ll k, int p)
11 { //Calcula (n,k) %p teniendo comb[p][p] precalculado.
12     ll aux = 1;
13     while (n + k)
14     {
15         aux = (aux * comb[n %p][k %p]) %p;
16         n/=p, k/=p;
17     }
18     return aux;
19 }

```

7.7. Exponenciación de Matrices y Fibonacci

```

1 #define SIZE 350
2 int NN;
3 void mul(double a[SIZE][SIZE], double b[SIZE][SIZE])
4 {
5     double res[SIZE][SIZE] = {{0}};
6     forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
7     forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
8 }
9 void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE])
10 {
11     forn(i, NN) forn(j, NN) res[i][j]=(i==j);
12     while(n)
13     {
14         if(n&1) mul(res, a), n--;
15         else mul(a, a), n/=2;
16     }
17 }
18
19 struct M22{    // |a b|
20     tipo a,b,c,d; // |c d| -- TIPO
21     M22 operator*(const M22 &p) const {
22         return (M22){a*p.a+b*p.c, a*p.b+b*p.d, c*p.a+d*p.c,c*p.b+d*p.d};}
23 };
24 M22 operator^(const M22 &p, int n)
25 { //VER COMO SE PUEDE PONER DENTRO DEL STRUCT
26     if(!n) return (M22){1, 0, 0, 1}; //identidad
27     M22 q=p^(n/2); q=q*q;
28     return n %2? p * q : q;
29 }
30
31 ll fibo(ll n)//calcula el fibonacci enesimo en O(logN)
32 {
33     M22 mat=(M22){0, 1, 1, 1}^n;
34     return mat.a*f0+mat.b*f1;//f0 y f1 son los valores iniciales

```

```

35 }

```

7.8. Operaciones Modulares

```

1 ll mulMod(ll a,ll b,ll m=MOD) //O(log b)
2 { //returns (a*b) %c, and minimize overflow
3     ll x=0, y=a %m;
4     while(b>0)
5     {
6         if(b%2==1) x=(x+y) %m;
7         y=(y*2) %m;
8         b/=2;
9     }
10    return x %m;
11 }
12 ll expMod(ll b,ll e,ll m=MOD) //O(log b)
13 {
14     if(!e) return 1;
15     ll q=expMod(b,e/2,m);
16     q=mulMod(q,q,m);
17     return e%2? mulMod(b,q,m) : q;
18 }
19 ll sumMod(ll a,ll b,ll m=MOD)
20 {
21     a %m;
22     b %m;
23     if(a<0) a+=m;
24     if(b<0) b+=m;
25     return (a+b) %m;
26 }
27 ll difMod(ll a,ll b,ll m=MOD)
28 {
29     a %m;
30     b %m;
31     if(a<0) a+=m;
32     if(b<0) b+=m;
33     ll ret=a-b;
34     if(ret<0) ret+=m;
35     return ret;
36 }
37 ll divMod(ll a,ll b,ll m=MOD)
38 {
39     return mulMod(a,inverso(b),m);
40 }

```


7.9. 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 #define MAXP 100000 //no necesariamente primo
2 int criba[MAXP+1];
3 void crearCriba()
4 {
5     int w[] = {4,2,4,2,4,6,2,6};
6     for(int p=25;p<=MAXP;p+=10) criba[p]=5;
7     for(int p=9;p<=MAXP;p+=6) criba[p]=3;
8     for(int p=4;p<=MAXP;p+=2) criba[p]=2;
9     for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
10     for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
11 }
12 vector<int> primos;
13 void buscarPrimos()
14 {
15     crearCriba();
16     forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
17 }
18
19 //factoriza bien numeros hasta MAXP^2
20 void fact(ll n,map<ll,ll> &f) //O (cant primos)
21 { //llamar a buscarPrimos antes
22     forall(p, primos){
23         while(!(n %p))
24         {
25             f[*p]++; //divisor found
26             n/=*p;
27         }
28     }
29     if(n>1) f[n]++;
30 }
31
32 //factoriza bien numeros hasta MAXP
33 void fact2(ll n,map<ll,ll> &f) //O (lg n)
34 { //llamar a crearCriba antes
35     while (criba[n])
36     {
37         f[criba[n]]++;
38         n/=criba[n];
39     }
40     if(n>1) f[n]++;
41 }
42
43 //Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
44 void divisores(map<ll,ll> &f,vector<ll> &divs,map<ll,ll>::iterator it,ll n
45 =1)
46 {
47     if(it==f.begin()) divs.clear();
48     if(it==f.end())
49     {

```

```

49     divs.pb(n);
50     return;
51 }
52 ll p=it->fst, k=it->snd; ++it;
53 forn(_, k+1) divisores(f, divs, it, n), n*=p;
54 }
55 ll cantDivs(map<ll,ll> &f)
56 {
57     ll ret=1;
58     forall(it, f) ret*=(it->second+1);
59     return ret;
60 }
61 ll sumDivs(map<ll,ll> &f)
62 {
63     ll ret=1;
64     forall(it, f)
65     {
66         ll pot=1, aux=0;
67         forn(i, it->snd+1) aux+=pot, pot*=it->fst;
68         ret*=aux;
69     }
70     return ret;
71 }
72
73 ll eulerPhi(ll n) // con criba: O(lg n)
74 {
75     map<ll,ll> f;
76     fact(n,f);
77     ll ret=n;
78     forall(it, f) ret-=ret/it->first;
79     return ret;
80 }
81 ll eulerPhi2(ll n) // O (sqrt n)
82 {
83     ll r = n;
84     forr (i,2,n+1)
85     {
86         if((ll)i*i>n) break;
87         if(n%i==0)
88         {
89             while(n%i==0) n/=i;
90             r -= r/i;
91         }
92     }
93     if (n != 1) r-= r/n;
94     return r;
95 }

```

7.10. Phollard's Rho

```

1 bool es_primo_prob(ll n, int a)
2 {
3     if(n==a) return true;
4     ll s=0,d=n-1;
5     while(d%2==0) s++,d/=2;
6     ll x=expMod(a,d,n);
7     if((x==1) || (x+1==n)) return true;
8     forn(i,s-1)
9     {
10         x=mulMod(x, x, n);
11         if(x==1) return false;
12         if(x+1==n) return true;
13     }
14     return false;
15 }
16 bool rabin (ll n) //devuelve true si n es primo
17 {
18     if(n==1) return false;
19     const int ar[]={2,3,5,7,11,13,17,19,23};
20     forn(j,9) if(!es_primo_prob(n,ar[j])) return false;
21     return true;
22 }
23 ll rho(ll n)
24 {
25     if((n&1)==0) return 2;
26     ll x=2,y=2,d=1;
27     ll c=rand() %n+1;
28     while(d==1)
29     {
30         x=(mulMod(x,x,n)+c) %n;
31         y=(mulMod(y,y,n)+c) %n;
32         y=(mulMod(y,y,n)+c) %n;
33         if(x-y>=0) d=gcd(n,x-y);
34         else d=gcd(n,y-x);
35     }
36     return d==n? rho(n):d;
37 }
38 void factRho (ll n,map<ll,ll> &f) //O (lg n)^3 un solo numero
39 {
40     if (n == 1) return;
41     if (rabin(n))
42     {
43         f[n]++;
44         return;
45     }
46     ll factor = rho(n);
47     factRho(factor,f);
48     factRho(n/factor,f);
49 }

```

7.11. Inversos

```

1 #define MAXMOD 15485867
2 ll inv[MAXMOD]; //inv[i]*i=1 mod MOD
3 void calc(int p) //O(p)
4 {
5     inv[1]=1;
6     forr(i,2,p) inv[i]=p-((p/i)*inv[p%i]) %p;
7 }
8 int inverso(int x) //O(log x)
9 {
10     return expMod(x, eulerPhi(MOD)-2); //si mod no es primo(sacar a mano)
11     return expMod(x, MOD-2); //si mod es primo
12 }

```

7.12. Fracciones

```

1 struct frac{
2     int p,q;
3     frac(int p=0,int q=1):p(p),q(q) {norm();}
4     void norm()
5     {
6         int a=gcd(q,p);
7         if(a) p/=a, q/=a;
8         else q=1;
9         if (q<0) q=-q, p=-p;
10    }
11    frac operator+(const frac& o)
12    {
13        int a=gcd(o.q,q);
14        return frac(p*(o.q/a)+o.p*(q/a),q*(o.q/a));
15    }
16    frac operator-(const frac& o)
17    {
18        int a=gcd(o.q,q);
19        return frac(p*(o.q/a)-o.p*(q/a),q*(o.q/a));
20    }
21    frac operator*(frac o)
22    {
23        int a=gcd(o.p,q), b=gcd(p,o.q);
24        return frac((p/b)*(o.p/a),(q/a)*(o.q/b));
25    }
26    frac operator/(frac o)
27    {
28        int a=gcd(o.q,q), b=gcd(p,o.p);
29        return frac((p/b)*(o.q/a),(q/a)*(o.p/b));
30    }
31    bool operator<(const frac &o) const{return p*o.q < o.p*q;}
32    bool operator==(frac o){return p==o.p&&q==o.q;}
33 };

```

7.13. Simpson

```

1 double integral(double a, double b, int n=10000) //O(n), n=cantdiv
2 {
3     double area=0, h=(b-a)/n, fa=f(a), fb;
4     forn(i, n)
5     {
6         fb=f(a+h*(i+1));
7         area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
8     }
9     return area*h/6.;
10 }

```

7.14. Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales	
0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 (∈ 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 (∈ 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
 9999941 9999959 9999971 9999989 10000007 10000037 10000039
 10000049
 99999893 99999929 99999937 100000007 100000009 100000021
 100000033

Cantidad de primos menores que 10^n

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

7.15. Números Catalanés

Útiles para problemas de Combinatoria

$$Cat(n) = \frac{\binom{2n}{n}}{n+1} = \frac{(2n)!}{n!(n+1)!}$$

Con $Cat(0) = 1$.

Diferentes aplicaciones:

1. Contar la cantidad de diferentes arboles binarios con n nodos que se pueden armar.

2. Contar las formas en que un polígono convexo de $n + 2$ lados puede ser triangulado.
3. Contar la cantidad de caminos monotonos a lo largo de los lados de una grilla $n * n$, que no cruzan la diagonal.
4. Contar el número de expresiones que contienen n pares de paréntesis correctamente colocados

7.15.1. Primeros 25 Catalanés

1 1 2 5 14 42 132 429 1430 4862 16796 58786 208012 742900 2674440
 9694845 35357670 129644790 477638700 1767263190 6564120420
 24466267020 91482563640 343059613650 1289904147324 4861946401452

8. Grafos

8.1. Dijkstra

```

1 #define INF 1e9
2 int N;
3 #define MAX_V 250001
4 vector<ii> G[MAX_V];
5 //To add an edge use
6 #define add(a, b, w) G[a].pb(make_pair(w, b))
7 ll dijkstra(int s, int t){//O(|E| log |V|)
8     priority_queue<ii, vector<ii>, greater<ii> > Q;
9     vector<ll> dist(N, INF); vector<int> dad(N, -1);
10    Q.push(make_pair(0, s)); dist[s] = 0;
11    while(sz(Q)){
12        ii p = Q.top(); Q.pop();
13        if(p.snd == t) break;
14        forall(it, G[p.snd])
15            if(dist[p.snd]+it->first < dist[it->snd]){
16                dist[it->snd] = dist[p.snd] + it->fst;
17                dad[it->snd] = p.snd;
18                Q.push(make_pair(dist[it->snd], it->snd));
19            }
20    }
21    return dist[t];
22    if(dist[t]<INF)//path generator
23        for(int i=t; i!=-1; i=dad[i])
24            printf("%d%c", i, (i==s?'\\n':' '));}

```

8.2. Bellman-Ford

```

1 //Mas lento que Dijkstra, pero maneja arcos con peso negativo
2 vector<ii> G[MAX_N]; //ady. list with pairs (weight, dst)
3 int dist[MAX_N];
4 void bford(int src){//O(VE)
5     dist[src]=0;
6     forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
7         dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
8 }
9
10 bool hasNegCycle(){
11     forn(j, N) if(dist[j]!=INF) forall(it, G[j])
12         if(dist[it->snd]>dist[j]+it->fst) return true;
13     //inside if: all points reachable from it->snd will have -INF distance (do
14         bfs)
15     return false;
16 }

```

8.3. Floyd-Warshall

```

1 // Camino minimo en grafos dirigidos ponderados, en todas las parejas de
  // nodos.
2 //G[i][j] contains weight of edge (i, j) or INF
3 //G[i][i]=0
4 int G[MAX_N][MAX_N];
5 void floyd(){//O(N^3)
6   forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
7     G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
8 }
9 bool inNegCycle(int v){
10   return G[v][v]<0;}
11 //checks if there's a neg. cycle in path from a to b
12 bool hasNegCycle(int a, int b){
13   forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
14     return true;
15   return false;
16 }

```

8.4. Kruskal

```

1 // Minimum Spanning Tree in O(e log e)
2 bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
3 vector<Ar> E;
4 ll kruskal(){
5   ll cost=0;
6   sort(E.begin(), E.end()); //ordenar aristas de menor a mayor
7   uf.init(n);
8   forall(it, E){
9     if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
10      uf.unir(it->a, it->b); //conectar
11      cost+=it->w;
12    }
13  }
14  return cost;
15 }

```

8.5. Prim

```

1 vector<ii> G[MAXN];
2 bool taken[MAXN];
3 priority_queue<ii, vector<ii>, greater<ii> > pq; //min heap
4 void process(int v){
5   taken[v]=true;
6   forall(e, G[v])
7     if(!taken[e->second]) pq.push(*e);
8 }

```

```

9 // Minimum Spanning Tree in O(n^2)
10 ll prim(){
11   zero(taken);
12   process(0);
13   ll cost=0;
14   while(sz(pq)){
15     ii e=pq.top(); pq.pop();
16     if(!taken[e.second]) cost+=e.first, process(e.second);
17   }
18   return cost;
19 }

```

8.6. 2-SAT + Tarjan SCC

```

1 //We have a vertex representing a var and other for his negation.
2 //Every edge stored in G represents an implication. To add an equation of
  // the form a||b, use addor(a, b)
3 //MAX=max cant var, n=cant var
4 #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
5 vector<int> G[MAX*2];
6 //idx[i]=index assigned in the dfs
7 //lw[i]=lowest index(closer from the root) reachable from i
8 int lw[MAX*2], idx[MAX*2], qidx;
9 stack<int> q;
10 int qcmp, cmp[MAX*2];
11 //verdad[cmp[i]]=valor de la variable i
12 bool verdad[MAX*2+1];
13
14 int neg(int x) { return x>=n? x-n : x+n;}
15 void tjn(int v){
16   lw[v]=idx[v]=++qidx;
17   q.push(v), cmp[v]=-2;
18   forall(it, G[v]){
19     if(!idx[*it] || cmp[*it]==-2){
20       if(!idx[*it]) tjn(*it);
21       lw[v]=min(lw[v], lw[*it]);
22     }
23   }
24   if(lw[v]==idx[v]){
25     int x;
26     do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
27     verdad[qcmp]=(cmp[neg(v)]<0);
28     qcmp++;
29   }
30 }
31 //remember to CLEAR G!!!
32 bool satisf(){//O(n)
33   memset(idx, 0, sizeof(idx)), qidx=0;
34   memset(cmp, -1, sizeof(cmp)), qcmp=0;
35   forn(i, n){
36     if(!idx[i]) tjn(i);

```

```

37     if(!idx[neg(i)]) tjn(neg(i));
38 }
39 forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
40 return true;
41 }

```

8.7. Puntos de Articulación

```

1  int N;
2  vector<int> G[1000000];
3  //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4  int qV, V[1000000], L[1000000], P[1000000];
5  void dfs(int v, int f){
6      L[v]=V[v]=++qV;
7      forall(it, G[v])
8          if(!V[*it]){
9              dfs(*it, v);
10             L[v] = min(L[v], L[*it]);
11             P[v] += L[*it]>=V[v];
12         }
13     else if(*it!=f)
14         L[v]=min(L[v], V[*it]);
15 }
16 int cantart(){ //O(n)
17     qV=0;
18     zero(V), zero(P);
19     dfs(1, 0); P[1]--;
20     int q=0;
21     forn(i, N) if(P[i]) q++;
22     return q;
23 }

```

8.8. Least Common Ancestor + Climb

```

1  const int MAXN=100001;
2  const int LOGN=20;
3  //f[v][k] holds the 2^k father of v
4  //L[v] holds the level of v
5  int N, f[MAXN][LOGN], L[MAXN];
6  //call before build:
7  void dfs(int v, int fa=-1, int lvl=0){//generate required data
8      f[v][0]=fa, L[v]=lvl;
9      forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1); }
10 void build(){//f[i][0] must be filled previously, O(nlgn)
11     forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
12 #define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
13 int climb(int a, int d){//O(lgn)
14     if(!d) return a;

```

```

15     dforn(i, lg(L[a])+1) if(1<=i<=d) a=f[a][i], d-=1<=i;
16     return a;}
17 int lca(int a, int b){//O(lgn)
18     if(L[a]<L[b]) swap(a, b);
19     a=climb(a, L[a]-L[b]);
20     if(a==b) return a;
21     dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
22     return f[a][0]; }
23 int dist(int a, int b) {//returns distance between nodes
24     return L[a]+L[b]-2*L[lca(a, b)];}

```

8.9. Heavy Light Decomposition

```

1  vector<int> G[MAXN];
2  int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
3  int dad[MAXN];//dad[v]=padre del nodo v
4  void dfs1(int v, int p=-1){//pre-dfs
5      dad[v]=p;
6      treesz[v]=1;
7      forall(it, G[v]) if(*it!=p){
8          dfs1(*it, v);
9          treesz[v]+=treesz[*it];
10     }
11 }
12 //PONER Q EN 0 !!!!
13 int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
14 //Las cadenas aparecen continuas en el recorrido!
15 int cantcad;
16 int homecad[MAXN];//dada una cadena devuelve su nodo inicial
17 int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
18 void heavylight(int v, int cur=-1){
19     if(cur==1) homecad[cur=cantcad++]=v;
20     pos[v]=q++;
21     cad[v]=cur;
22     int mx=-1;
23     forn(i, sz(G[v])) if(G[v][i]!=dad[v])
24         if(mx==1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
25     if(mx!=1) heavylight(G[v][mx], cur);
26     forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
27         heavylight(G[v][i], -1);
28 }
29 //ejemplo de obtener el maximo numero en el camino entre dos nodos
30 //RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
31 //esta funcion va trepando por las cadenas
32 int query(int an, int v){//O(logn)
33     //si estan en la misma cadena:
34     if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
35     return max(query(an, dad[homecad[cad[v]]]),
36               rmq.get(pos[homecad[cad[v]]], pos[v]+1));
37 }

```

8.10. Centroid Decomposition

```

1 vector<int> G[MAXN];
2 bool taken[MAXN]; //poner todos en FALSE al principio!!
3 int padre[MAXN]; //padre de cada nodo en el centroid tree
4
5 int szt[MAXN];
6 void calcsz(int v, int p) {
7     szt[v] = 1;
8     forall(it, G[v]) if (*it!=p && !taken[*it])
9         calcsz(*it, v), szt[v]+=szt[*it];
10 }
11 void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) { //O(nlogn)
12     if(tam== -1) calcsz(v, -1), tam=szt[v];
13     forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
14         {szt[v]=0; centroid(*it, f, lvl, tam); return;}
15     taken[v]=true;
16     padre[v]=f;
17     forall(it, G[v]) if(!taken[*it])
18         centroid(*it, v, lvl+1, -1);
19 }

```

8.11. Ciclo Euleriano

```

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 }

```

8.12. Diametro Árbol

```

1 vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
2 int bfs(int r, int *d) {
3     queue<int> q;
4     d[r]=0; q.push(r);
5     int v;
6     while(sz(q)) { v=q.front(); q.pop();
7         forall(it, G[v]) if (d[*it]==-1)
8             d[*it]=d[v]+1, p[*it]=v, q.push(*it);
9     }
10    return v; //ultimo nodo visitado
11 }
12 vector<int> diams; vector<ii> centros;
13 void diametros(){
14     memset(d,-1,sizeof(d));
15     memset(d2,-1,sizeof(d2));
16     diams.clear(), centros.clear();
17     forn(i, n) if(d[i]==-1){
18         int v,c;
19         c=v=bfs(bfs(i, d2), d);
20         forn(_, d[v]/2) c=p[c];
21         diams.pb(d[v]);
22         if(d[v]&1) centros.pb(ii(c, p[c]));
23         else centros.pb(ii(c, c));
24     }
25 }

```

8.13. Componentes Biconexas y Puentes

```

1 vector<int> G[MAXN];
2
3 struct edge{
4     int u,v, comp;
5     bool bridge;
6 };
7 vector<edge> e;
8 void addEdge(int u, int v)
9 {
10     G[u].pb(sz(e)), G[v].pb(sz(e));
11     e.pb((edge){u,v,-1,false});
12 }

```

```

13 //d[i]=id de la dfs
14 //b[i]=lowest id reachable from i
15 int d[MAXN], b[MAXN], t;
16 int nbc;//cant componentes
17 int comp[MAXN]; //comp[i]=cant comp biconexas a la cual pertenece i
18 void initDfs(int n)
19 {
20     zero(G), zero(comp);
21     e.clear();
22     forn(i,n) d[i]=-1;
23     nbc = t = 0;
24 }
25 stack<int> st;
26 void dfs(int u,int pe) //O(n + m)
27 {
28     b[u]=d[u]=t++;
29     comp[u]=(pe!=-1);
30     forall(ne,G[u]) if(*ne!=pe)
31     {
32         int v=e[*ne].u ^ e[*ne].v ^ u;
33         if(d[v]==-1)
34         {
35             st.push(*ne);
36             dfs(v,*ne);
37             if(b[v]>d[u]) e[*ne].bridge=true; // bridge
38             if(b[v]>=d[u]) // art
39             {
40                 int last;
41                 do
42                 {
43                     las=st.top(); st.pop();
44                     e[last].comp=nbc;
45                 }while(last!=*ne);
46                 nbc++;
47                 comp[u]++;
48             }
49             b[u]=min(b[u],b[v]);
50         }
51         else if(d[v]<d[u]) // back edge
52         {
53             st.push(*ne);
54             b[u]=min(b[u], d[v]);
55         }
56     }
57 }

```

8.14. Hungarian

```

1 //Dado un grafo bipartito completo con costos no negativos, encuentra el
  matching perfecto de minimo costo.
2 #define tipo double

```

```

3 tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar: cost=matriz de adyacencia
4 int n, max_match, xy[N], yx[N], slackx[N], prev2[N]; //n=cantidad de nodos
5 bool S[N], T[N]; //sets S and T in algorithm
6 void add_to_tree(int x, int prevx) {
7     S[x] = true, prev2[x] = prevx;
8     forn(y, n) if (lx[x] + ly[y] - cost[x][y] < slack[y] - EPS)
9         slack[y] = lx[x] + ly[y] - cost[x][y], slackx[y] = x;
10 }
11 void update_labels(){
12     tipo delta = INF;
13     forn(y, n) if (!T[y]) delta = min(delta, slack[y]);
14     forn(x, n) if (S[x]) lx[x] -= delta;
15     forn(y, n) if (T[y]) ly[y] += delta; else slack[y] -= delta;
16 }
17 void init_labels(){
18     zero(lx), zero(ly);
19     forn(x,n) forn(y,n) lx[x] = max(lx[x], cost[x][y]);
20 }
21 void augment() {
22     if (max_match == n) return;
23     int x, y, root, q[N], wr = 0, rd = 0;
24     memset(S, false, sizeof(S)), memset(T, false, sizeof(T));
25     memset(prev2, -1, sizeof(prev2));
26     forn(x, n) if (xy[x] == -1){
27         q[wr++] = root = x, prev2[x] = -2;
28         S[x] = true; break; }
29     forn(y, n) slack[y] = lx[root] + ly[y] - cost[root][y], slackx[y] = root;
30     while (true){
31         while (rd < wr){
32             x = q[rd++];
33             for (y = 0; y < n; y++) if (cost[x][y] == lx[x] + ly[y] && !T[y]){
34                 if (yx[y] == -1) break; T[y] = true;
35                 q[wr++] = yx[y], add_to_tree(yx[y], x); }
36             if (y < n) break; }
37         if (y < n) break;
38         update_labels(), wr = rd = 0;
39         for (y = 0; y < n; y++) if (!T[y] && slack[y] == 0){
40             if (yx[y] == -1){x = slackx[y]; break;}
41             else{
42                 T[y] = true;
43                 if (!S[yx[y]]) q[wr++] = yx[y], add_to_tree(yx[y], slackx[y]);
44             }
45             if (y < n) break; }
46         if (y < n){
47             max_match++;
48             for (int cx = x, cy = y, ty; cx != -2; cx = prev2[cx], cy = ty)
49                 ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
50             augment(); }
51     }
52 }
53 tipo hungarian(){
54     tipo ret = 0; max_match = 0, memset(xy, -1, sizeof(xy));
55     memset(yx, -1, sizeof(yx)), init_labels(), augment(); //steps 1-3
56     forn(x,n) ret += cost[x][xy[x]]; return ret;

```


8.15. Dynamic Connectivity

```

56 }

1 struct UnionFind {
2     int n, comp;
3     vector<int> pre, si, c;
4     UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
5         forn(i,n) pre[i] = i; }
6     int find(int u){return u==pre[u]?u:find(pre[u]);}
7     bool merge(int u, int v)
8     {
9         if((u=find(u))== (v=find(v))) return false;
10        if(si[u]<si[v]) swap(u, v);
11        si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
12        return true;
13    }
14    int snap(){return sz(c);}
15    void rollback(int snap)
16    {
17        while(sz(c)>snap)
18        {
19            int v = c.back(); c.pop_back();
20            si[pre[v]] -= si[v], pre[v] = v, comp++;
21        }
22    }
23 };
24 enum {ADD, DEL, QUERY};
25 struct Query {int type, u, v;};
26 struct DynCon{//bidirectional graphs; create vble as DynCon name(cant_nodos)
27     vector<Query> q;
28     UnionFind dsu;
29     vector<int> match, res;
30     map<ii, int> last;//se puede no usar cuando hay identificador para cada
        arista (mejora poco)
31     DynCon(int n=0):dsu(n){}
32     void add(int u, int v) //to add an edge
33     {
34         if(u>v) swap(u,v);
35         q.pb((Query){ADD, u, v}), match.pb(-1);
36         last[ii(u,v)] = sz(q)-1;
37     }
38     void remove(int u, int v) //to remove an edge
39     {
40         if(u>v) swap(u,v);
41         q.pb((Query){DEL, u, v});
42         int prev = last[ii(u,v)];
43         match[prev] = sz(q)-1;
44         match.pb(prev);
45     }
46     void query() //to add a question (query) type of query

```

```

47 {
48     q.pb((Query){QUERY, -1, -1}), match.pb(-1);
49 }
50 void process() //call this to process queries in the order of q
51 {
52     forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz(q);
53     go(0,sz(q));
54 }
55 void go(int l, int r)
56 {
57     if(l+1==r)
58     {
59         if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
60         res.pb(dsu.comp);//aqui query=cantidad de componentes conexas
61         return;
62     }
63     int s=dsu.snap(), m = (l+r) / 2;
64     forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v);
65     go(l,m);
66     dsu.rollback(s);
67     s = dsu.snap();
68     forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].v);
69     go(m,r);
70     dsu.rollback(s);
71 }
72 };

```

9. Flow

9.1. Edmond Karp

```

1 #define MAX_V 1000
2 #define INF 1e9
3 //special nodes
4 #define SRC 0
5 #define SNK 1
6 map<int, int> G[MAX_V]; //limpiar esto -- unordered_map mejora
7 //To add an edge use
8 #define add(a, b, w) G[a][b]=w
9 int f, p[MAX_V];
10 void augment(int v, int minE)
11 {
12     if(v==SRC) f=minE;
13     else if(p[v]!=-1)
14     {
15         augment(p[v], min(minE, G[p[v]][v]));
16         G[p[v]][v]-=f, G[v][p[v]]+=f;
17     }
18 }
19 ll maxflow() //O(min(V E^2, Mf * E))
20 {
21     ll Mf=0;
22     do
23     {
24         f=0;
25         char used[MAX_V]; queue<int> q; q.push(SRC);
26         zero(used), memset(p, -1, sizeof(p));
27         while(sz(q))
28         {
29             int u=q.front(); q.pop();
30             if(u==SNK) break;
31             forall(it, G[u])
32                 if(it->snd>0 && !used[it->fst])
33                     used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
34         }
35         augment(SNK, INF);
36         Mf+=f;
37     }while(f);
38     return Mf;
39 }

```

9.2. Min Cut

```

1 //Suponemos un grafo con el formato definido en Edmond Karp o Push relabel
2 bitset<MAX_V> type,used; //reset this
3 void dfs1(int nodo)
4 {

```

```

5     type.set(nodo);
6     forall(it,G[nodo]) if(!type[it->fst] && it->snd>0) dfs1(it->fst);
7 }
8 void dfs2(int nodo)
9 {
10     used.set(nodo);
11     forall(it,G[nodo])
12     {
13         if(!type[it->fst])
14         {
15             //edge nodo -> (it->fst) pertenece al min_cut
16             //y su peso original era: it->snd + G[it->fst][nodo]
17             //si no existia arista original al revs
18         }
19         else if(!used[it->fst]) dfs2(it->fst);
20     }
21 }
22 void minCut() //antes correr algn maxflow()
23 {
24     dfs1(SRC);
25     dfs2(SNK);
26     return;
27 }

```

9.3. Push Relabel

```

1 #define MAX_V 1000
2 int N; //valid nodes are [0...N-1]
3 #define INF 1e9
4 //special nodes
5 #define SRC 0
6 #define SNK 1
7 map<int, int> G[MAX_V]; //limpiar esto -- unordered_map mejora
8 //To add an edge use
9 #define add(a, b, w) G[a][b]=w
10 ll excess[MAX_V];
11 int height[MAX_V], active[MAX_V], cuenta[2*MAX_V+1];
12 queue<int> Q;
13
14 void enqueue(int v)
15 {
16     if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v);
17 }
18 void push(int a, int b)
19 {
20     int amt = min(excess[a], ll(G[a][b]));
21     if(height[a] <= height[b] || amt == 0) return;
22     G[a][b]-=amt, G[b][a]+=amt;
23     excess[b] += amt, excess[a] -= amt;
24     enqueue(b);
25 }

```

```

26 void gap(int k)
27 {
28     forn(v, N)
29     {
30         if (height[v] < k) continue;
31         cuenta[height[v]]--;
32         height[v] = max(height[v], N+1);
33         cuenta[height[v]]++;
34         enqueue(v);
35     }
36 }
37 void relabel(int v)
38 {
39     cuenta[height[v]]--;
40     height[v] = 2*N;
41     forall(it, G[v])
42     if(it->snd) height[v] = min(height[v], height[it->fst] + 1);
43     cuenta[height[v]]++;
44     enqueue(v);
45 }
46 ll maxflow() //O(V^3)
47 {
48     zero(height), zero(active), zero(cuenta), zero(excess);
49     cuenta[0]=N-1; cuenta[N]=1;
50     height[SRC] = N;
51     active[SRC] = active[SNK] = true;
52     forall(it, G[SRC])
53     {
54         excess[SRC] += it->snd;
55         push(SRC, it->fst);
56     }
57     while(sz(Q))
58     {
59         int v = Q.front(); Q.pop();
60         active[v]=false;
61         forall(it, G[v]) push(v, it->fst);
62         if(excess[v] > 0)
63             cuenta[height[v]] == 1? gap(height[v]):relabel(v);
64     }
65     ll mf=0;
66     forall(it, G[SRC]) mf+=G[it->fst][SRC];
67     return mf;
68 }

```

9.4. Dinic

```

1 struct Edge {
2     int u, v;
3     ll cap, flow;
4     Edge() {}
5     Edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0) {}

```

```

6 };
7 struct Dinic {
8     int N;
9     vector<Edge> E;
10    vector<vector<int>>> g;
11    vector<int> d, pt;
12    Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {} //clear and init
13    void addEdge(int u, int v, ll cap)
14    {
15        if (u != v)
16        {
17            E.emplace_back(Edge(u, v, cap));
18            g[u].emplace_back(E.size() - 1);
19            E.emplace_back(Edge(v, u, 0));
20            g[v].emplace_back(E.size() - 1);
21        }
22    }
23    bool BFS(int S, int T)
24    {
25        queue<int> q({S});
26        fill(d.begin(), d.end(), N + 1);
27        d[S] = 0;
28        while(!q.empty())
29        {
30            int u = q.front(); q.pop();
31            if (u == T) break;
32            for (int k: g[u])
33            {
34                Edge &e = E[k];
35                if (e.flow < e.cap && d[e.v] > d[e.u] + 1)
36                {
37                    d[e.v] = d[e.u] + 1;
38                    q.emplace(e.v);
39                }
40            }
41        }
42        return d[T] != N + 1;
43    }
44    ll DFS(int u, int T, ll flow = -1)
45    {
46        if (u == T || flow == 0) return flow;
47        for (int &i = pt[u]; i < g[u].size(); ++i)
48        {
49            Edge &e = E[g[u][i]];
50            Edge &oe = E[g[u][i]^1];
51            if (d[e.v] == d[e.u] + 1)
52            {
53                ll amt = e.cap - e.flow;
54                if (flow != -1 && amt > flow) amt = flow;
55                if (ll pushed = DFS(e.v, T, amt))
56                {
57                    e.flow += pushed;
58                    oe.flow -= pushed;

```

```

59         return pushed;
60     }
61 }
62 }
63 return 0;
64 }
65 ll maxFlow(int S,int T)
66 {
67     ll total = 0;
68     while(BFS(S, T))
69     {
70         fill(pt.begin(), pt.end(), 0);
71         while (ll flow = DFS(S, T)) total += flow;
72     }
73     return total;
74 }
75 };

```

9.5. Min cost - Max flow

```

1  const int MAXN=10000;
2  typedef ll tf;
3  typedef ll tc;
4  const tf INFFLUJO = 1e14;
5  const tc INFCOSTO = 1e14;
6  struct edge {
7      int u, v;
8      tf cap, flow;
9      tc cost;
10     tf rem() { return cap - flow; }
11 };
12 int nodes; //numero de nodos
13 vector<int> G[MAXN]; // limpiar!
14 vector<edge> e; // limpiar!
15 void addEdge(int u, int v, tf cap, tc cost)
16 {
17     G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
18     G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
19 }
20 tc dist[MAXN], mnCost;
21 int pre[MAXN];
22 tf cap[MAXN], mxFlow;
23 bool in_queue[MAXN];
24 void flow(int s, int t)
25 {
26     zero(in_queue);
27     mxFlow=mnCost=0;
28     while(1)
29     {
30         fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
31         memset(pre, -1, sizeof(pre)); pre[s]=0;

```

```

32     zero(cap); cap[s] = INFFLUJO;
33     queue<int> q; q.push(s); in_queue[s]=1;
34     while(sz(q))
35     {
36         int u=q.front(); q.pop(); in_queue[u]=0;
37         for(auto it:G[u])
38         {
39             edge &E = e[it];
40             if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9) // ojo EPS
41             {
42                 dist[E.v]=dist[u]+E.cost;
43                 pre[E.v] = it;
44                 cap[E.v] = min(cap[u], E.rem());
45                 if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
46             }
47         }
48     }
49     if (pre[t] == -1) break;
50     mxFlow +=cap[t];
51     mnCost +=cap[t]*dist[t];
52     for (int v = t; v != s; v = e[pre[v]].u)
53     {
54         e[pre[v]].flow += cap[t];
55         e[pre[v]^1].flow -= cap[t];
56     }
57 }
58 }

```

10. Juegos

10.1. Nim Game

Juego en el que hay N pilas, con objetos. Cada jugador debe sacar al menos un objeto de una pila. GANA el jugador que saca el último objeto.

$$P_0 \oplus P_1 \oplus \dots \oplus P_n = R$$

Si $R \neq 0$ gana el jugador 1.

10.1.1. Misere Game

Es un juego con las mismas reglas que Nim, pero PIERDE el que saca el último objeto. Entonces teniendo el resultado de la suma R , y si todas las pilas tienen 1 solo objeto $todos1=true$, podemos decir que el jugador2 GANA si:

$$(R=0) \& \neg todos1 \parallel (R \neq 0) \& todos1$$

10.2. Ajedrez

10.2.1. Non-Attacking N Queen

Utiliza: <algorithm>

Notas: todo es $O(N \cdot N^2)$.

```

1 #define NQUEEN 8
2 #define abs(x) ((x)<0?(-(x)):(x))
3
4 int board[NQUEEN];
5 void inline init() {for(int i=0;i<NQUEEN;++i)board[i]=i;}
6 bool check() {
7     for(int i=0;i<NQUEEN;++i)
8         for(int j=i+1;j<NQUEEN;++j)
9             if(abs(i-j)==abs(board[i]-board[j]))
10                return false;
11     return true;
12 }
13 //en main
14 init();
15 do{
16     if(check()){
17         //process solution
18     }
19 }while(next_permutation(board,board+NQUEEN));

```

11. Utils

11.1. Convertir string a num e viceversa

```

1 #include <sstream>
2 string num_to_str(int x){
3     ostringstream convert;
4     convert << x;
5     return convert.str();
6 }
7
8 int str_to_num(string x){
9     int ret;
10    istringstream (x) >> ret;
11    return ret;
12 }

```

11.2. Truquitos para entradas/salidas

```

1 //Cantidad de decimales
2 cout << setprecision(2) << fixed;
3 //Rellenar con espacios(para justificar)
4 cout << setfill(' ') << setw(3) << 2 << endl;
5 //Leer hasta fin de linea
6 // hacer cin.ignore() antes de getline()
7 while(getline(cin, line)){
8     istringstream is(line);
9     while(is >> X)
10         cout << X << " ";
11     cout << endl;
12 }

```

11.3. Comparación de Double

```

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

```

11.4. Iterar subconjuntos

```

1 for(int sbm=0; sbm<1<N; sbm=(sbm+1)&1)

```

11.5. Limites

```

1 #include <limits>
2 numeric_limits<T>
3 ::max()
4 ::min()
5 ::epsilon()

```

11.6. Mejorar Lectura de Enteros

```

1 //Solo para enteros positivos
2 inline void Scanf(int& a)
3 {
4     char c = 0;
5     while(c<33) c = getc(stdin);
6     a = 0;
7     while(c>33) a = a*10 + c - '0', c = getc(stdin);
8 }

```

11.7. Tablita de relacion de Complejidades

n	Peor AC Complejidad	Comentario
$\leq [10.,11]$	$O(n!), O(n^6)$	ej. Enumerar permutaciones
$\leq [15.,18]$	$O(2^n \times n^2)$	ej. DP TSP
$\leq [18.,22]$	$O(2^n \times n)$	ej. DP con mascara de bits
≤ 100	$O(n^4)$	ej. DP con 3 dimensiones + $O(n)$ loops
≤ 400	$O(n^3)$	ej. Floyd Warshall
$\leq 2K$	$n^2 \log_2 n$	ej. 2 loops anidados + una busqueda en arbol en una estructura de datos
$\leq 10K$	$O(n^2)$	ej. Ordenamiento Burbuja/Selección/Inserción
$\leq 1M$	$O(n \log_2 n)$	ej. Merge Sort, armar Segment Tree
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	La mayoría de los problemas de contest tiene $n \leq 1M$ (cuello de botella en I/O)

11.8. Compilar C++11 con g++

Dos opciones, útil en Linux.

```

1 g++ -std=c++11 {file} -o {filename}
2
3 g++ -std=c++0x {file} -o {filename}

```

11.9. Build de C++11 para Sublime Text

```

1 {
2     "shell_cmd": "g++ -std=c++0x \"${file}\" -o \"${file_path}/${file_base_name}\"",
3     "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)??:? (.*)$",
4     "working_dir": "${file_path}",
5     "selector": "source.cpp, source.c++",
6     "variants":
7     [{
8         "name": "Run",

```

```

9     "shell_cmd": "g++ -std=c++0x \"${file}\" -o \"${file_path}/${file_base_name}\" && \"${file_path}/${file_base_name}\""
10  }

```

11.10. Funciones Utiles

Algo	Params	Función
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
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
lexicographical_compare	f1, l1, f2, l2	bool con [f1,l1]; [f2,l2]
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 1s en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.