

El Bichi

UTN FRSF - El Rejunte

2017



Índice

1. C/C++	2		
1.1. I/O	2		
1.1.1. scanf Format Strings	2		
1.1.2. printf Format Strings	3		
2. Template del Rejunte	4		
3. Estructuras de datos	4		
3.1. Set Mejorado	4		
3.2. Union Find	4		
3.3. Hash Table	5		
3.4. RMQ	5		
3.4.1. RMQ (static)	5		
3.4.2. RMQ (dynamic)	5		
3.4.3. RMQ (lazy)	5		
3.4.4. RMQ (persistente)	6		
4. Strings	6		
4.1. KMP	6		
		4.2. Z function	6
		4.3. Trie	7
		5. Geometría	7
		6. Matemática	7
		6.1. Identidades	7
		6.2. Ec. Característica	7
		6.3. Teorema Chino del Resto	7
		6.4. GCD & LCM	7
		6.5. Euclides Extendido	8
		6.6. Combinatoria	8
		6.7. Exponenciación de Matrices y Fibonacci	8
		6.8. Operaciones Modulares	8
		6.9. Funciones de Primos	9
		6.10. Phollard's Rho	10
		6.11. Inversos	10
		6.12. Fracciones	10
		6.13. Simpson	11
		6.14. Tablas y cotas (Primos, Divisores, Factoriales, etc)	11
		7. Grafos	12
		8. Flow	12
		8.1. Edmond Karp	12
		8.2. Push Relabel	12
		9. Juegos	13
		9.1. Ajedrez	13
		9.1.1. Non-Attacking N Queen	13

10. Utils	13
10.1. Convertir string a num e viceversa	13
10.2. Contar bits	13

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 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 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

```

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)
13    {
14        bool con=comp(i)==comp(j);
15        if(!con)
16        {
17            cantSets--;
18            f[comp(i)]=comp(j);
19            setSize[comp(j)]+=setSize[comp(i)];
20            setSize[comp(i)]=setSize[comp(j)]; //no suma, solo asigna
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 }

```

4. Strings

4.1. 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 }

```

4.2. 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-1]);
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 }

```

4.3. 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. Geometría

6. Matemática

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. 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.

6.3. Teorema Chino del Resto

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

6.4. 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));}

```

6.5. Euclides Extendido

```

1 void extendedEuclid (ll a, ll b){ //a * x + b * y = d
2     if (!b) {x=1; y=0; d=a; return;}
3     extendedEuclid (b,a%b);
4     ll x1=y;
5     ll y1=x-(a/b)*y;
6     x=x1; y=y1;
7 }

```

6.6. 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 }

```

6.7. 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 }

```

6.8. 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); q=mulMod(q,q,m);
16     return e%2? mulMod(b,q,m) : q;
17 }
18 ll sumMod(ll a,ll b,ll m=MOD)
19 {
20     return (a %m+b %m) %m;
21 }

```



```

22 ll difMod(ll a,ll b,ll m=MOD)
23 {
24     ll ret=a%m-b%m;
25     if(ret<0) ret+=m;
26     return ret;
27 }
28 ll divMod(ll a,ll b,ll m=MOD)
29 {
30     return mulMod(a,inverso(b),m);
31 }

```

6.9. 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     {
50         divs.pb(n);
51         return;
52     }
53     ll p=it->fst, k=it->snd; ++it;
54     forn(_, k+1) divisores(f, divs, it, n), n*=p;
55 }
56 ll cantDivs(map<ll,ll> &f)
57 {
58     ll ret=1;
59     forall(it, f) ret*=(it->second+1);
60     return ret;
61 }
62 ll sumDivs(map<ll,ll> &f)
63 {
64     ll ret=1;
65     forall(it, f)
66     {
67         ll pot=1, aux=0;
68         forn(i, it->snd+1) aux+=pot, pot*=it->fst;
69         ret*=aux;
70     }
71     return ret;
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 }

```

```

43     f[n]++;
44     return;
45 }
46 ll factor = rho(n);
47 factRho(factor, f);
48 factRho(n/factor, f);
49 }

```

6.10. 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     for(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     for(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     {

```

6.11. 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     for(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 }

```

6.12. 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 };

```

6.13. 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 }

```

6.14. 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
 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
 $\pi(10^{10}) = 455.052.511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

7. Grafos

8. Flow

8.1. 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(VE^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 }

```

8.2. 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. Juegos

9.1. Ajedrez

9.1.1. 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));

```

10. Utils

10.1. 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 }

```

10.2. Contar bits

```

1 int x = 5328; // 000000000000000000001010011010000
2
3 //cuenta los zeros a la izquierda del numero
4 __builtin_clz(x); // 19
5
6 //cuenta los zeros a la derecha del numero
7 __builtin_ctz(x); // 4
8
9 //cuenta los unos en el numero
10 __builtin_popcount(x); // 5
11
12 //cuenta la paridad de cantidad de unos
13 __builtin_parity(x); // 1

```