

# Cartilla de código piola

UTN FRSF - El Rejunte

2017

## Índice

<b>1. C/C++</b>	<b>2</b>	<b>8. Grafos</b>	<b>7</b>
1.1. Typedefs . . . . .	2	<b>9. Flow</b>	<b>7</b>
1.2. Macros . . . . .	2	<b>10. Juegos</b>	<b>7</b>
1.3. I/O . . . . .	2	10.1. Ajedrez . . . . .	7
1.3.1. scanf Format Strings . . . . .	2	10.1.1. Non-Attacking N Queen . . . . .	7
1.3.2. printf Format Strings . . . . .	3	<b>11. Utils</b>	<b>8</b>
1.3.3. Fast C++ Input . . . . .	4		
<b>2. Template del Rejunte</b>	<b>4</b>		
<b>3. Constantes y Tablas</b>	<b>4</b>		
3.1. Constantes . . . . .	4		
<b>4. Estructuras de datos</b>	<b>4</b>		
4.1. Disjoint Sets . . . . .	4		
4.1.1. Union Find (OOP) . . . . .	4		
4.1.2. Union Find (C Style/Static) . . . . .	5		
4.2. Segment Trees . . . . .	5		
4.2.1. Range Sum Query (lazy) . . . . .	5		
4.2.2. Range Min/Max Query (lazy) . . . . .	6		
<b>5. Strings</b>	<b>7</b>		
<b>6. Geometría</b>	<b>7</b>		
<b>7. Matemática</b>	<b>7</b>		
7.1. GCD & LCM . . . . .	7		
7.1.1. GCD Extendido (Extended Euclid) . . . . .	7		
7.2. Primos . . . . .	7		
7.2.1. Criba de Eratóstenes . . . . .	7		

# 1. C/C++

## 1.1. Typedefs

```
1 typedef long long ll;
2 typedef long double ld;
3 typedef pair<int,int> pii;
4 typedef vector<pii> vpii;
5 typedef vector<int> vi;
6 typedef vector<vi> vvi;
```

## 1.2. Macros

```
1 //buffers
2 #define MEMZ(mem) memset(mem, 0, sizeof mem) // memset => <cstring / string.
   h >
3 #define MEMX(mem,x) memset(mem, (x), sizeof mem)
4
5 //loops
6 #define forall(i,a,b) for(int (i)=(a);(i)<(b);++(i))
7 #define foreach(i,v) for(typeof((v).begin()) (i) = (v).begin(); (i) != (v).
   end(); ++(i))
8
9 //map/pairs
10 #define mp make_pair
11 #define fi first
12 #define se second
13
14 //vector
15 #define pb push_back
16
17 //queries
18 #define in(a,v) ((v).find((a))!=(v).end()) // <algorithm>
```

## 1.3. I/O

### 1.3.1. 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.

Continuación		
spec	Tipo	Descripción
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.
lenght	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*
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*

Continuación				
length	f e g a	c s [ ] [^]	p	n
<b>l</b>	double*	wchar_t*		long int*
<b>ll</b>				long long int*
<b>j</b>				intmax_t*
<b>z</b>				size_t*
<b>t</b>				ptrdiff_t*
<b>L</b>	long double*			

1.3.2. printf Format Strings

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

specifier	Descripción	Ejemplo
d o r 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
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).

Continuación	
flag	Descripción
+	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
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*

Continuación					
length	f F e E g G a A	c	s	p	n
l		wint_t	wchar_t*		long int*
ll					long long int*
j					intmax_t*
z					size_t*
t					ptrdiff_t*
L	long double				

1.3.3. Fast C++ Input

```
1 ios_base::sync_with_stdio(false); cin.tie(NULL);
```

2. Template del Rejunte

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define sqr(a) ((a)*(a))
5 #define RAND(a,b) ((a)+rand()%((b)-(a)+1))
6 #define rsz resize
7 #define forr(i,a,b) for(int i=(a);i<(b);i++)
8 #define forn(i,n) forr(i,0,n)
9 #define dforr(i,n) for(int i=n-1;i>=0;i--)
10 #define forall(it,v) for(auto it=v.begin();it!=v.end();it++)
11 #define pb push_back
12 #define mp make_pair
13 #define lb lower_bound
14 #define ub upper_bound
15 #define fst first
16 #define snd second
17 #define INF 1000000000
18
19 typedef long long ll;
20 typedef pair<int,int> ii;
21 typedef vector<int> vi;
22 typedef vector<ii> vii;
23
24 #define max(a,b) ((a)>(b)?(a):(b));
25 #define min(a,b) ((a)<(b)?(a):(b));
26
27 int main()
28 {
29     freopen("input","r",stdin);
30     //freopen("output","w",stdout);
31
32     return 0;
```

```
33 }
```

3. Constantes y Tablas

3.1. Constantes

```
1 #define INF 1000000000 // 1 billon, entra en int
2 #define EPS 1e-12
3 #define PI 3.1415926535897932384626
```

4. Estructuras de datos

4.1. Disjoint Sets

4.1.1. Union Find (OOP)

Utiliza: <vector>

Notas: Rangos [i, j] (0 based). No recomendable si se tienen que crear y destruir muchos objetos. Probar funcionamiento en casos límites.

```
1 class UnionFind
2 {
3 private:
4     vector<int> p, rank, setSize;
5     int numSets;
6 public:
7     UnionFind(int N)
8     {
9         setSize.assign(N, 1);
10        numSets = N;
11        rank.assign(N, 0);
12        p.assign(N, 0);
13        for (int i = 0; i < N; i++) p[i] = i;
14    }
15    int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
16    bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
17    void unionSet(int i, int j)
18    {
19        int x = findSet(i), y = findSet(j);
20        if (!(x==y))
21        {
22            numSets--;
23            if (rank[x] > rank[y])
24            {
25                p[y] = x;
26                setSize[x] += setSize[y];
```

```

27     }
28     else
29     {
30         p[x] = y;
31         setSize[y] += setSize[x];
32         if (rank[x] == rank[y]) rank[y]++;
33     }
34 }
35 }
36 int numDisjointSets() { return numSets; }
37 int sizeOfSet(int i) { return setSize[findSet(i)]; }
38 };

```

### 4.1.2. Union Find (C Style/Static)

Utiliza: <cstring>

Notas: Rangos [i, j] (0 based). En init(n),  $n \leq \text{MAXN}$

```

1  #define MAXN ...
2  int p[MAXN], rank[MAXN], setSize[MAXN], numSets;
3
4  inline void init(int n)
5  {
6      memset(rank, 0, sizeof(int)*n);
7      for(int i=0; i<n; ++i) p[i]=i, setSize[i]=1;
8      numSets = n;
9  }
10 inline int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
11
12 inline bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
13 void unionSet(int i, int j)
14 {
15     int x = findSet(i), y = findSet(j);
16     if (x==y)
17     {
18         numSets--;
19         if (rank[x] > rank[y])
20         {
21             p[y] = x;
22             setSize[x] += setSize[y];
23         }
24         else
25         {
26             p[x] = y;
27             setSize[y] += setSize[x];
28             if (rank[x] == rank[y]) rank[y]++;
29         }
30     }
31 }
32 inline int numDisjointSets() { return numSets; }
33 inline int sizeOfSet(int i) { return setSize[findSet(i)]; }

```

## 4.2. Segment Trees

### 4.2.1. Range Sum Query (lazy)

Utiliza: <cmath>

Notas: Limpiar st y lazy en cada testcase. En main():

- N es la cantidad de elementos del problema.
- $(0 \leq i, j < N)$  build(1, 0, N-1), update(1, 0, N-1, i, j, value), query(1, 0, N-1, i, j). Los elementos en arr tienen que estar desde arr[0].
- $(1 \leq i, j \leq N)$  build(1, 1, N), update(1, 1, N, i, j, value), query(1, 1, N, i, j). Los elementos en arr tienen que estar desde arr[1].

```

1  const int MAXN = ...;
2  const int MAXST = (int) (2*exp2(ceil(log2(MAXN)))+1);
3
4  #define OP(a,b) ((a) + (b))
5  #define NEUTRO 0LL
6
7  #define left(x) ((x)<<1)
8  #define right(x) (((x)<<1)+1)
9  #define middle(a,b) (((a)+(b))>>1)
10
11 ll st[MAXST], lazy[MAXST]; int arr[MAXN+1];
12
13 void build(int node, int a, int b)
14 {
15     if(a>b) return;
16     if(a==b){st[node] = arr[a]; return;}
17     int l=left(node), r=right(node), m=middle(a,b);
18     build(l,a,m);
19     build(r,m+1,b);
20     st[node]=OP(st[l],st[r]);
21 }
22
23 void update(int node, int a, int b, int i, int j, ll value)
24 {
25     int l=left(node), r=right(node), m=middle(a,b);
26
27     if(lazy[node]!=0LL)
28     {
29         st[node] = (b-a+1)*lazy[node]; // SET
30         // += (b-a+1)*lazy[node]; // ADD
31         if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
32         // lazy[l]+=lazy[node], lazy[r]+=lazy[node]; // ADD
33         lazy[node] = 0LL;
34     }
35
36     if(a>b || a>j || b<i) return;
37
38     if(a>=i && b<=j)
39     {

```

```

40     st[node] = (b-a+1)*value; // SET
41     // += (b-a+1)*value; // ADD
42     if(a!=b) lazy[l] = lazy[r] = value;
43     // lazy[l]+=value, lazy[r]+=value; // ADD
44     return;
45 }
46
47 update(l,a,m,i,j,value);
48 update(r,m+1,b,i,j,value);
49
50 st[node] = OP(st[l],st[r]);
51 }
52
53 ll query(int node, int a, int b, int i, int j)
54 {
55     if(a>b || a>j || b<i) return NEUTRO;
56
57     int l=left(node), r=right(node), m=middle(a,b);
58
59     if(lazy[node]!=0)
60     {
61         st[node] = (b-a+1)*lazy[node];
62         if(a!=b) lazy[l] = lazy[r] = lazy[node];
63         lazy[node] = 0;
64     }
65     if(a>=i && b<=j) return st[node];
66
67     int ql = query(l,a,m,i,j);
68     int qr = query(r,m+1,b,i,j);
69     return OP(ql,qr);
70 }

```

#### 4.2.2. Range Min/Max Query (lazy)

**Utiliza:** <cmath>

**Notas:** Limpiar st y lazy en cada testcase. En main():

- N es la cantidad de elementos del problema.
- $(0 \leq i, j < N)$  build(1,0,N-1), update(1,0,N-1,i,j,value), query(1,0,N-1,i,j). Los elementos en arr tienen que estar desde arr[0].
- $(1 \leq i, j \leq N)$  build(1,1,N), update(1,1,N,i,j,value), query(1,1,N,i,j). Los elementos en arr tienen que estar desde arr[1].

```

1 const int MAXN = ...;
2 const int MAXST = (int) (2*exp2(ceil(log2(MAXN)))+1);
3
4 #define min(a,b) ((a)<(b)?(a):(b))
5 #define max(a,b) ((a)>(b)?(a):(b))
6
7 #define OP(a,b) (max(a,b)) // operador binario asociativo

```

```

8 #define NEUTRO 0 // elemento neutro del operador
9
10 #define left(x) ((x)<<1)
11 #define right(x) (((x)<<1)+1)
12 #define middle(a,b) (((a)+(b))>>1)
13
14 int st[MAXST], lazy[MAXST], arr[MAXN+1];
15
16 void build(int node, int a, int b)
17 {
18     if(a>b) return;
19     if(a==b) {st[node] = arr[a]; return;}
20     int l = left(node), r = right(node), m = middle(a,b);
21     build(l,a,m);
22     build(r,m+1,b);
23     st[node] = OP(st[l], st[r]);
24 }
25
26 void update(int node, int a, int b, int i, int j, int value)
27 {
28     int l = left(node), r = right(node), m = middle(a,b);
29
30     if(lazy[node]!=0)
31     {
32         st[node] = lazy[node]; // SET [a,b] to value
33         // += lazy[node]; // ADD value to [a,b]
34         if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
35         // lazy[l] += lazy[node], lazy[r] += lazy[node]; // ADD
36         lazy[node] = 0;
37     }
38
39     if(a>b || a>j || b<i) return;
40
41     if(a>=i && b<=j)
42     {
43         st[node] = value; // SET
44         // += value; // ADD
45         if(a!=b) lazy[l] = lazy[r] = value; // SET
46         // lazy[l] += value, lazy[r] += value; // ADD
47         return;
48     }
49
50     update(l,a,m,i,j,value);
51     update(r,m+1,b,i,j,value);
52
53     st[node] = OP(st[l], st[r]);
54 }
55
56 int query(int node, int a, int b, int i, int j)
57 {
58     if(a>b || a>j || b<i) return NEUTRO;
59
60     int l = left(node), r = right(node), m = middle(a,b);

```

```

61 if(lazy[node]!=0)
62 {
63     st[node] = lazy[node]; // SET [a,b] to value
64     //      += lazy[node]; // ADD value to [a,b]
65     if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
66     //      lazy[l] +=lazy[node], lazy[r] += lazy[node]; // ADD
67     lazy[node] = 0;
68 }
69
70
71 if(a>=i && b<=j) return st[node];
72
73 int ql = query(l,a,m,i,j);
74 int qr = query(r,m+1,b,i,j);
75
76 return OP(ql, qr);
77 }

```

## 5. Strings

## 6. Geometría

## 7. Matemática

### 7.1. GCD & LCM

```

1 int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
2 int lcm(int a, int b) { return a * (b / gcd(a, b)); }

```

#### 7.1.1. GCD Extendido (Extended Euclid)

**Notas:** ecuaciones diofánticas ( $ax + by = d$  con  $d = \text{gcd}(a, b)$ ). Da la primer solución  $(x_0, y_0)$ , el resto mediante  $x = x_0 + (b/d)n, y = y_0 - (a/d)n$  con  $n \geq 1$ .

```

1 void gcd_ext(int a, int b, int&x, int& y, int& d)
2 {
3     if (b == 0) { x = 1; y = 0; d = a; return; }
4     gcd_ext(b, a % b, x, y, d);
5     int x1 = y;
6     int y1 = x - (a / b) * y;
7     x = x1;
8     y = y1;
9 }

```

## 7.2. Primos

### 7.2.1. Criba de Eratóstenes

**Utiliza:** <vector>, <bitset>

**Notas:** ll=long long, vi=vector<int>. Guarda los primos [0-upperbound] en primes.isPrime funciona sólo para  $N \leq$  (último primo en primes)<sup>2</sup>. Cuidado con bs si el tamaño es  $> 10^7$

```

1 ll sieve_size;
2 bitset<10000010> bs; // 10^7
3 vi primes;
4
5 void sieve(ll upperbound)
6 {
7     sieve_size = upperbound + 1;
8     bs.set(); bs[0] = bs[1] = 0;
9     for (ll i = 2; i <= sieve_size; i++)
10         if (bs[i])
11         {
12             for (ll j = i * i; j <= sieve_size; j += i)
13                 bs[j] = 0;
14             primes.push_back((int)i);
15         }
16 }
17
18 bool isPrime(ll N)
19 {
20     if (N <= sieve_size) return bs[N];
21     for (int i = 0; i < (int)primes.size(); i++)
22         if (N % primes[i] == 0) return false;
23     return true;
24 }

```

## 8. Grafos

## 9. Flow

## 10. Juegos

### 10.1. Ajedrez

#### 10.1.1. 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));
```

## 11. Utils