Cartilla de código piola

UTN FRSF - El Rejunte

2016

Índice

1. C/C++

1.1. Typedefs

```
typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef vector<pii> vpii;
typedef vector<int> vi;
typedef vector<vi> vvi;
```

1.2. Macros

```
//buffers
#define MEMZ(mem) memset(mem, 0, sizeof mem) // memset => <cstring / string.h >
#define MEMX(mem,x) memset(mem, (x), sizeof mem)

//loops
#define forall(i,a,b) for(int (i)=(a);(i)<(b);++(i))
#define foreach(i,v) for(typeof((v).begin()) (i) = (v).begin(); (i) != (v).end(); ++(i)
    )

//map/pairs
#define mp make_pair
#define fi first
#define se second

//vector
#define pb push_back

//queries
#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.
`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.

Continuación

spec	Tipo	Descripción
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*
l	double*	wchar_t*		long int*
ll				long long int*
j				intmax_t*
z				size_t*
t				ptrdiff_t*
L	long double*			

1.3.2. 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

Continuación

specifier	Descripción	Ejemplo
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 representaciíon más corta: %e ó %f	392.65
G	Utilizar la representaciíon 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).
+	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.

Continuación	
precision	**Descripción**
.*	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*
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

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

2. Constantes y Tablas

2.1. Constantes

```cpp
#define INF 1000000000 // 1 billon, entra en int
#define EPS 1e-12
#define PI 3.14159265358979323846
```

3. Estructuras de datos

3.1. Disjoint Sets

3.1.1. 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.

```cpp
class UnionFind
{
private:
  vector<int> p, rank, setSize;
  int numSets;
public:
  UnionFind(int N)
  {
    setSize.assign(N, 1);
    numSets = N;
    rank.assign(N, 0);
    p.assign(N, 0);
    for (int i = 0; i < N; i++) p[i] = i;
  }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j)
  {
    int x = findSet(i), y = findSet(j);
    if (!(x==y))
    {
      numSets--;
      if (rank[x] > rank[y])
      {
        p[y] = x;
        setSize[x] += setSize[y];
      }
      else
      {
        p[x] = y;
        setSize[y] += setSize[x];
        if (rank[x] == rank[y]) rank[y]++;
      }
    }
  }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};
```

3.1.2. Union Find (C Style/Static)

Utiliza: <cstring>

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

```cpp
#define MAXN ...
int p[MAXN], rank[MAXN], setSize[MAXN], numSets;

inline void init(int n)
{
    memset(rank,0, sizeof(int)*n);
    for(int i=0;i<n;++i) p[i]=i, setSize[i]=1;
    numSets = n;
}
inline int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
inline bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
void unionSet(int i, int j)
{
```

```
    int x = findSet(i), y = findSet(j);
    if (!(x==y))
    {
        numSets--;
        if (rank[x] > rank[y])
        {
            p[y] = x;
            setSize[x] += setSize[y];
        }
        else
        {
            p[x] = y;
            setSize[y] += setSize[x];
            if (rank[x] == rank[y]) rank[y]++;
        }
    }
}
inline int numDisjointSets() { return numSets; }
inline int sizeOfSet(int i) { return setSize[findSet(i)]; }
```

3.2. Segment Trees

3.2.1. Range Sum Query (lazy)

Utiliza: <cstring>
Notas: Limpiar st y lazy en cada testcase. En main():

- $(0 \leq i,j < N)$ 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].

```
const int MAXN = ...;
const int MAXST = (int) (2*exp2(ceil(log2(MAXN)))+1);

#define OP(a,b) ((a) + (b))
#define NEUTRO 0LL

#define left(x) ((x)<<1)
#define right(x) (((x)<<1)+1)
#define middle(a,b) (((a)+(b))>>1)

ll st[MAXST], lazy[MAXST]; int arr[MAXN+1];

void build(int node, int a, int b)
{
    if(a>b) return;
    if(a==b){st[node] = arr[a]; return;}
    int l=left(node), r=right(node), m=middle(a,b);
    build(l,a,m);
    build(r,m+1,b);
    st[node]=OP(st[l],st[r]);
}

void update(int node, int a, int b, int i, int j, ll value)
{
    int l=left(node), r=right(node), m=middle(a,b);

    if(lazy[node]!=0LL)
    {
        st[node] = (b-a+1)*lazy[node]; // SET
        //      += (b-a+1)*lazy[node]; // ADD
        if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
        //       lazy[l]+=lazy[node],lazy[r]+=lazy[node]; // ADD
        lazy[node] = 0LL;
    }

    if(a>b || a>j || b<i) return;

    if(a>=i && b<=j)
    {
        st[node] = (b-a+1)*value; // SET
        //      += (b-a+1)*value; // ADD
        if(a!=b) lazy[l] = lazy[r] = value;
        //       lazy[l]+=value,lazy[r]+=value; // ADD
        return;
    }

    update(l,a,m,i,j,value);
    update(r,m+1,b,i,j,value);

    st[node] = OP(st[l],st[r]);
}

ll query(int node, int a, int b, int i, int j)
{
    if(a>b || a>j || b<i) return NEUTRO;

    int l=left(node), r=right(node), m=middle(a,b);

    if(lazy[node]!=0)
    {
        st[node] = (b-a+1)*lazy[node];
        if(a!=b) lazy[l] = lazy[r] = lazy[node];
        lazy[node] = 0;
    }
    if(a>=i && b<=j) return st[node];

    int ql = query(l,a,m,i,j);
    int qr = query(r,m+1,b,i,j);
    return OP(ql,qr);
}
```

3.2.2. Range Min/Max Query (lazy)

```
const int MAXN  = ...;
const int MAXST = (int) (2*exp2(ceil(log2(MAXN)))+1);

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))

#define OP(a,b) (max(a,b)) // operador binario asociativo
#define NEUTRO 0       // elemento neutro del operador

#define left(x) ((x)<<1)
#define right(x) (((x)<<1)+1)
#define middle(a,b) (((a)+(b))>>1)

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
62   if(lazy[node]!=0)
63   {
64     st[node] = lazy[node]; // SET [a,b] to value
65     //      += lazy[node]; // ADD value to [a,b]
66     if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
67     //     lazy[l] +=lazy[node], lazy[r] += lazy[node]; // ADD
68     lazy[node] = 0;
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 }
```

4. Teoría de Números

4.1. 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));}
```

4.1.1. GCD Extendido (Extended Euclid)

Notas: ecuaciones diofánticas ($ax + by = d$ con $d = gcd(a, b)$). 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 }
```

4.2. Primos

4.2.1. 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)^2. 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 }
```