Complejidad Algorítmica

Límites de tiempo

En las competencias de programación, tu programa tiene que terminar su ejecución en una cantidad de tiempo definida por el problema para considerarse aceptado.

La cantidad de tiempo es diferente en cada problema, por lo general, el tiempo límite en C++ es 1 segundo y en Java/Python se considera un incremento de 3 a 4 segundos a la de c++

Si tu programa falla en producir un resultado a tiempo, recibirás el veredicto de TLE o tiempo límite excedido.

Operaciones

La complejidad en tiempo de un programa depende del número de operaciones que realiza.

Puedes considerar una operación como una instrucción a la computadora. Por ejemplo, multiplicar dos números o imprimir el resultado serían operaciones.

La regla que generalmente se usa es que el juez virtual puede soportar 10^8 operaciones por segundo.

Calcular la complejidad

Necesitamos un método para calcular cuantas operaciones le toma a un algoritmo, en término del tamaño de la entrada N del problema.

Esto se puede hacer usando la notación **Big O,** el cual expresa la complejidad del peor caso en función de N.

En la notación Big O, denotamos la complejidad de una función como $O(f(n))$, donde los factores constantes o los términos de menor complejidad son omitidos.

Ejemplos

El siguiente código es $O(1)$ porque ejecuta un número constante de operaciones.

```
int a = 5;
int b = 7;
int c = 4;
int d = a + b + c + 153;
```

Ejemplos

La complejidad de los ciclos es el número de iteraciones que corre el ciclo. La complejidad del siguiente código es $O(n)$.

```
int i = 0;
while (i < n) {
    // constant time code here
    i++;
}
```

Ejemplos

Debido a que ignoramos factores constantes y términos de menor complejidad, los siguientes códigos también son $O(n)$.

```
for (int i = 1; i <= 5*n + 17; i++) {
    // constant time code here
}
for (int i = 1; i <= n + 457737; i++) {
    // constant time code here
}
```

Ejemplos

Podemos hallar la complejidad de múltiples ciclos multiplicando las complejidades de cada ciclo. Este ejemplo es `O(n * m)`.

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        // constant time code here
    }
}
```

De forma más general cada ciclo tiene complejidad `O(n)` multiplicado por la complejidad de lo que ejecuta.

Ejemplos

Si un algoritmo contiene múltiples bloques, su complejidad es la peor complejidad de cualquiera de los bloques. Por ejemplo, la complejidad del siguiente código es $O(n^2)$.

```
for (int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        // constant time code here
    }
}
for (int i = 1; i <= n + 58834; i++) {
    // more constant time code here
}
```

Ejemplos

Calcular la complejidad del siguiente código:

```
while (n != 0) {
    for (int i = 1; i <= m; i++) {
        // constant time code here
    }

    n = n / 2;
}
```

Ejemplos

Calcular la complejidad del siguiente código:

```
for (int i = 1; i * i <= m; i++) {
    // constant time code here
}
```

Orden de funciones

Para determinar cuál es el término con mayor complejidad se puede usar la siguiente lista:

$$n! > 2^n > n^3 > n^2 > nlog_2 n > n > \sqrt{n} > log_2 n > cte$$

Complejidades comunes

n	Peor complejidad	Ejemplo
$n \leq 11$	$O(n!)$	enumerar permutaciones
$n \leq 18$	$O(2^n n^2)$	TSP con DP y máscaras de bits
$n \leq 22$	$O(2^n)$	Enumerar subconjuntos
$n \leq 400$	$O(n^3)$	Floyd-Warshall
$n \leq 2000$	$O(n^2 logn)$	2 Loops anidados + Búsqueda binaria
$n \leq 10^4$	$O(n^2)$	Recorrer una matriz cuadrada
$n \leq 10^6$	$O(nlogn)$	Construir una árbol de segmentos
$n \leq 10^8$	$O(n), O(logn), O(1)$	Recorrer un arreglo
$n \leq 10^{18}$	$O(logn), O(1)$	Búsqueda binaria o fórmula

A practicar!

El cálculo de complejidades puede no parecer tan importante ahora, pero es muy importante al resolver problemas más difíciles.

Trata de calcular la complejidad del código que escribes!