TSP - PROGRAMACION II

COMPLEJIDAD ALGORÍTMICA

COMPLEJIDAD ALGORÍTMICA

- La **Complejidad Algorítmica** es una métrica teórica que se aplica a los algoritmos como una forma de comparar una solución algorítmica con otras y de esta forma conocer cómo se comportará al atacar problemas complejos.
- La complejidad no es un número: es una función.

COMPLEJIDAD ALGORÍTMICA

- **Complejidad Temporal:** representa la idea del **tiempo** que consume un algoritmo para resolver un problema.
- **Complejidad Espacial:** representa la cantidad de **memoria** que consume un algoritmo para resolver un problema.

Entonces lo que vamos a medir es cómo crece el número de instrucciones necesarias para resolver el problema con respecto al tamaño del problema.

COMPLEJIDAD ALGORÍTMICA - Ejemplo lineal

```
//pseudocódigo
funcion ejemplo(n: entero):entero
empieza
   variables: a, j, k enteros
   para j desde 1 hasta n hacer
      a=a+j
   fin para
   para k desde n hasta 1 hacer
      a=a-1
      a=a*2
   fin para
   devolver a
termina
```

1 instrucción -> 1 ejecución.

1 bucle -> n veces cada instrucción.

1 x n + 2 x n + 1 = **3n + 1**

Asignar 1 unidad de tiempo a **n**

Algoritmo responde de manera lineal

COMPLEJIDAD ALGORÍTMICA - Ejemplo cuadrática

```
//pseudocódigo
funcion ejemplo3(n:entero): entero
empieza
   variables i, j, k enteros
   para i=1 hasta n hacer
      para j=n hasta 1 hacer
         k=k+j
         k=k+i
      fin para
   fin para
    devolver k
termina
```

1 instrucción -> 1 ejecución.

1 bucle -> n veces cada instrucción.

2 x n x n + 1 = $\mathbf{2n^2 + 1}$

Asignar 1 unidad de tiempo a **n**

Algoritmo responde de manera cuadrática.

COMPLEJIDAD ALGORÍTMICA - Ejemplo de Peor Caso o Mejor Caso

```
//pseudocódigo
funcion EstaEn(v: array[1..n] de enteros, x: entero):booleano
empieza
   variables: i:entero, encontrado:booleano;
   i=1;
   encontrado=false;
   mientras(NO(encontrado) Y x<=n) hacer
      si v[i]==x entonces
        encontrado=true;
      fin si
      i=i+1
   fin mientras
   devolver encontrado
termina
```

1 instrucción -> 1 ejecución.

1 bucle -> n veces cada instrucción.

Peor caso -> se ejecuta hasta el final el bucle.

Mejor caso -> sólo se ingresa a la primera posición del array.

COMPLEJIDAD ALGORÍTMICA - Orden de Complejidad

- Debido a que analizar cada algoritmo de manera independiente para obtener la complejidad real sería muy complejo, lo que podemos hacer es agruparlos dependiendo de la complejidad que presenten, por ejemplo: lineal, cuadrática, cúbica, etc.
- A esta agrupación la vamos a llamar Orden de Complejidad.
- Veremos que realmente lo que influye en la complejidad de los algoritmos son los bucles y las sentencias de selección que tienen efecto en esos bucles.
- Los algoritmos recursivos tienen un análisis de complejidad distinto y complejo.

COMPLEJIDAD ALGORÍTMICA - Órdenes de Complejidad más comunes

Órden	Nombre	Comentario
O(1)	constante	Todos aquellos algoritmos que responden en un tiempo constante, sea cual sea la talla del problema. Son los que aplican alguna fórmula sencilla, por ejemplo, hallar el máximo de dos valores

COMPLEJIDAD ALGORÍTMICA - Órdenes de Complejidad más comunes

Órden	Nombre	Comentario
O(log n)	logarítmico	Los que el tiempo crece con un criterio logarítmico, independientemente de cuál sea la base mientras ésta sea mayor que 1. Por eso, normalmente, ni siquiera se indica la base. No suelen ser muchos, y normalmente están bien considerados, ya que implican que un bucle realiza **menos** iteraciones que la talla del problema, lo cual no suele ser muy común. Por ejemplo, la búsqueda dicotómica en un vector ordenado.

COMPLEJIDAD ALGORÍTMICA - Órdenes de Complejidad más comunes

Órden	Nombre	Comentario
O(n)	lineal	El tiempo crece linealmente con respecto a la talla. Por ejemplo, encontrar el máximo de un vector de talla n.

COMPLEJIDAD ALGORÍTMICA - Órdenes de Complejidad más comunes

Órden	Nombre	Comentario
O(n·log(n))	enelogarímico, loglineal, linearítmico o simplemente n por logaritmo de n	Éste orden tiene muchos nombres. Es un orden relativamente bueno, porque la mayor parte de los algoritmos tienen un orden superior. En éste orden está, por ejemplo, el algoritmo de ordenación Quicksort, o la transformada rápida de Fourier.

COMPLEJIDAD ALGORÍTMICA - Órdenes de Complejidad más comunes

Órden	Nombre	Comentario
$O(n^c)$, con $c>1$	polinómico	Aquí están muchos de los algoritmos más comunes. Cuando c es 2 se le llama **cuadrático**, cuando es 3 se le llama **cúbico**, y en general, polinómico. Intuitivamente podríamos decir que éste órden es el último de los aceptables (siempre y cuando c sea relativamente bajo). A partir del siguiente, los algoritmos son complicados de tratar en la práctica cuando n es muy grande.

COMPLEJIDAD ALGORÍTMICA - Órdenes de Complejidad más comunes

Órden	Nombre	Comentario
$O(c^n)$, con $c>1$	exponencial	Aunque pudiera no parecerlo, es mucho peor que el anterior. Crece muchísimo más rápidamente.
$O(n!)$	factorial	Es el típico de aquellos algoritmos que para un problema complejo prueban todas las combinaciones posibles.
$O(n^n)$	combinatorio	Tan intratable como el anterior. A menudo no se hace distinción entre ellos.|

COMPLEJIDAD ALGORÍTMICA - Listas enlazadas vs. vectores o matrices

	Vector	Lista Enlazada
Indexado	O(1)	O(n)
Inserción / Eliminación al final	O(1)	O(1) or O(n)^2
Inserción / Eliminación en la mitad	O(n)	O(1)
Persistencia	No	Simples sí
Localidad	Buena	Mala

Tabla extendida: http://bigocheatsheet.com/