Aplicaciones básicas de análisis de algoritmos

Edson Huillca Alarcón

Tiempo de ejecución de un algoritmo

Para medir T(n) usamos el número de operaciones elementales. Una operación elemental puede ser:

- Operacion aritmetica
- Asignación a una variable
- Llamada a una función
- Retorno de una función
- Comparaciones lógicas
- Acceso a una estructura (arreglo, matriz, lista ligada, ….)

NOTA: se llama tiempo de ejecución, no al tiempo físico, sino al número de operaciones elementales que se llevan a cabo en el algoritmo

Ejemplo

Análisis de operaciones elementales en un algoritmo que busca un número dado dentro de un arreglo ordenado.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  int ArregloOrdenado[9]={1,3,7,15,19,24,31,38,40};
  int buscado, j=0;

  cout<<"¿Que numero entero quieres buscar?";
  cin>>buscado;

  while(ArregloOrdenado[j] < buscado && j<9)
      j=j+1;

  if(ArregloOrdenado[j] == buscado)
      cout<<"El numero buscado esta en el arreglo";
  else
      cout<<"Numero buscado no encontrado";

  system("PAUSE");

  return EXIT_SUCCESS;
}
```

1. Analicemos el número de operaciones elementales del algoritmo anterior tomando en cuenta el *mejor caso*
2. Analicemos el número de operaciones elementales del algoritmo anterior tomando en cuenta el *peor caso*

IMPORTANTE

- No vale mucho la pena complicar la metodología estimando las constantes.
- En lugar de calcular T(n) exactamente, queremos sólo cotas superiores (e inferiores) para T(n) ignorando factores constantes.

Cálculo de tiempo de ejecución en un ciclo *for*

Normalmente se considera el peor caso, cuando se calcula el número de OE.

```
for(int i=0; i<n; i++) {
  // instrucciones ...
}

// Forma equivalente del for:
int i=0;
while(i<n){
  // instrucciones ...
  i=i+1;
}
```

$$1 + 1 + \sum_{i=0}^{n-1} (\#OE\ de\ instrucciones + 2\ +\ 1)$$

Análisis de Complejidad

Ejercicios:
I. Indicar si las siguientes afirmaciones son ciertas:

1. $n^2 \in O(n^3)$
2. $n^3 \in O(n^2)$
3. $(n+1)! \in O(n!)$
4. $3^n \in O(2^n)$
5. $\log_2 n \in O(n^{1/2})$

II. Determina la complejidad O de las funciones:

1. $T(n) = 3n^3 + 2n$
2. $T(n) = 50n^2 + 5n + 1$
3. $T(n) = 5n^3 + 2020n^2 + 1$
4. $T(n) = \log(n) + 1010n + 1$
5. $T(n) = 100nlog(n)+n^2$
6. $T(n) = 6nlog(n)+n$

Clasificación de algoritmos con la notación O

- La notación O sirve para identificar si un algoritmo tiene un orden de complejidad mayor o menor que otro.
- Un algoritmo es mas eficiente mientras menor sea su orden de complejidad.

Ejercicio: Ordenar de mayor a menor

1. $T(n) = 3n^3 + 2n$
2. $T(n) = 50n^2 + 5n + 1$
3. $T(n) = 5n^3 + 2020n^2 + 1$
4. $T(n) = \log(n) + 1010n + 1$
5. $T(n) = 100nlog(n)+n^2$
6. $T(n) = 6nlog(n)+n$