Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

Unidad Profesional Interdisciplinaria de Ingeniería Campus Tlaxcala UPIIT

Algoritmos y Estructuras de Datos

Esaú Eliezer Escobar Juárez

Ingeniería en Inteligencia Artificial (IIA)

Definición Algoritmo

- Conjunto finito de instrucciones que si las seguimos resuelven una tarea particular.

1. Entrada/Salida
2. Definido: sin ambigüedades
3. Finito: Termina después de un número de pasos
4. Efectivo: Cada instrucción debe ser básica

Áreas de estudio

- Cómo diseñar algoritmos
- Cómo validar algoritmos
  - Mostrar que computa la respuesta correcta para todas las entradas legales.
- Cómo analizar algoritmos
  - Determinar cuanto tiempo computacional y almacenamiento requiere un algoritmo.
- Cómo probar un programa
  - Depurar: ejecutar el programa sobre datos de ejemplo y ver si los resultados son correctos.
  - Medida de desempeño: Medir el tiempo y el espacio requerido por el programa.

Pseudocódigo

- El siguiente algoritmo encuentra y devuelve el máximo de n números dados.

```
Algorithm Max(A, n)
// A is an array of size n.
{
    Result := A[1];
    for i := 2 to n do
        if A[i] > Result then Result := A[i];
    return Result;
}
```

Convertir un problema en algoritmo

Enunciado:

Diseñar un algoritmo que ordene una colección de n>=1 elementos.

De aquellos elementos que se encuentran en desorden, encontrar el más pequeño y colocarlo en la siguiente posición en la lista ordenada.

```
for i := 1 to n do
{
    Examine a[i] to a[n] and suppose
    the smallest element is at a[j];
    Interchange a[i] and a[j];
}
```

```
Algorithm SelectionSort(a, n)
// Sort the array a[1 : n] into nondecreasing order.
{
    for i := 1 to n do
    {
        j := i;
        for k := i + 1 to n do
            if (a[k] < a[j]) then j := k;
        t := a[i]; a[i] := a[j]; a[j] := t;
    }
}
```

Medidas

- ¿El algoritmo hace lo que queremos que haga?
- ¿Funciona correctamente?
- ¿Está documentado?
- ¿El código es legible?

Análisis de desempeño (Independiente de la máquina)

- Complejidad espacial
- Complejidad Temporal

Medida del desempeño (Dependiente de la máquina)

https://www.youtube.com/watch?v=UR2oDYZ-Sao

Complejidad espacial

Es la suma de 2 partes

- ✓ Una parte fija que es independiente de las características (número, tamaño) de las entradas y salidas
- ✓ Una parte variable que consiste del espacio necesario por las variables.

El requerimiento de espacio S(P) de un algoritmo P puede escribirse como:

$$S(P) = c + S_P(Entrada)$$

Complejidad espacial

Algoritmo 1: Computa a+b+b*c+(a+b-c)/(a+b)+4.0

Algorithm abc(a, b, c)
{
    return $a + b + b * c + (a + b - c)/(a + b) + 4.0$;
}

El espacio necesario no depende de la entrada pues sólo necesitamos a, b y c.

$$S_P(Entrada) = 0$$

Complejidad espacial

Algoritmo 2: Computa $\sum_{i=1}^{n} a[i]$

```
Algorithm Sum(a, n)
{
    s := 0.0;
    for i := 1 to n do
        s := s + a[i];
    return s;
}
```

El espacio necesario para a[ ] es de tamaño n, además es necesario el espacio de s, i y n mismo.

$$S_{Sum}(n) = (n + 3)$$

Complejidad espacial

Algoritmo 3: Computa $\sum_{i=1}^{n} a[i]$ recursivamente

```
Algorithm RSum(a, n)
{
    if (n ≤ 0) then return 0.0;
    else return RSum(a, n − 1) + a[n];
}
```

Cada llamada recursiva requiere de 3 variables: n, el apuntador de a[ ] y la dirección del "return". La profundidad de la recursión es n+1.

$$S_{RSum}(n) = 3(n + 1)$$

Complejidad Temporal

- El tiempo T(P) requerido por un programa P es la suma del tiempo de compilación y el tiempo de ejecución

- Si bien puede estar relacionado a el tiempo que toma hacer cada operación, en nuestro caso sólo contaremos el número de pasos de programa.

Complejidad Temporal

- Las operaciones sólo cuentan un paso

  return a+b*+(a+b –c)/(a+b)+4.0

- Los comentarios no cuentan
- Las asignaciones cuentan un paso
- Las iteraciones cuentan el número de veces que se ejecutan multiplicado por el número de instrucciones que contienen.

Complejidad Temporal

- Una forma de medir la complejidad temporal es añadir instrucciones de cuenta dentro del algoritmo.

Complejidad temporal

Algoritmo 2: Computa $\sum_{i=1}^{n} a[i]$

```
Algorithm Sum(a, n)
{
    s := 0.0;
    count := count + 1; // count is global; it is initially zero.
    for i := 1 to n do
    {
        count := count + 1; // For for
        s := s + a[i]; count := count + 1; // For assignment
    }
    count := count + 1; // For last time of for
    count := count + 1; // For the return
    return s;
}
```

Podemos ver que dentro del ciclo for el valor de count se incrementa en 2n, por lo que el tiempo total es de:

$$t_{Sum}(n) = 2n + 3$$

Complejidad temporal

Algoritmo 3: Computa $\sum_{i=1}^{n} a[i]$ recursivamente

```
Algorithm RSum(a, n)
{
    count := count + 1; // For the if conditional
    if (n ≤ 0) then
    {
        count := count + 1; // For the return
        return 0.0;
    }
    else
    {
        count := count + 1;  // For the addition, function
                             // invocation and return
        return RSum(a, n − 1) + a[n];
    }
}
```

- Vemos que cuando n=0, tenemos: $t_{RSum}(0)=2$
- Cuando n>0, cuenta se incrementa en 2 mas el resultado de invocar a la función nuevamente: $2+t_{RSum}(n-1)$

$$t_{RSum}(n) = \begin{cases} 2, & Si \quad n = 0 \\ 2 + t_{RSum}(n-1), & Si \quad n > 0 \end{cases}$$

Complejidad temporal

- Vemos que cuando n=0, tenemos: $t_{RSum}(0)=2$
- Cuando n>0, cuenta se incrementa en 2 mas el resultado de invocar a la función nuevamente: $2+t_{RSum}(n-1)$

$$t_{RSum}(n) = \begin{cases} 2, & Si \quad n = 0 \\ 2 + t_{RSum}(n-1), & Si \quad n > 0 \end{cases}$$

- Esta relación de recurrencia puede resolverse de la siguiente manera:

$$\begin{aligned} t_{\mathsf{RSum}}(n) &= 2 + t_{\mathsf{RSum}}(n-1) \\ &= 2 + 2 + t_{\mathsf{RSum}}(n-2) \\ &= 2(2) + t_{\mathsf{RSum}}(n-2) \\ &\vdots \\ &= n(2) + t_{\mathsf{RSum}}(0) \\ &= 2n + 2, \qquad\qquad n \geq 0 \end{aligned}$$

- Entonces la cuenta para RSum es $t_{RSum}(n)=2n+2$

Notación Asintótica

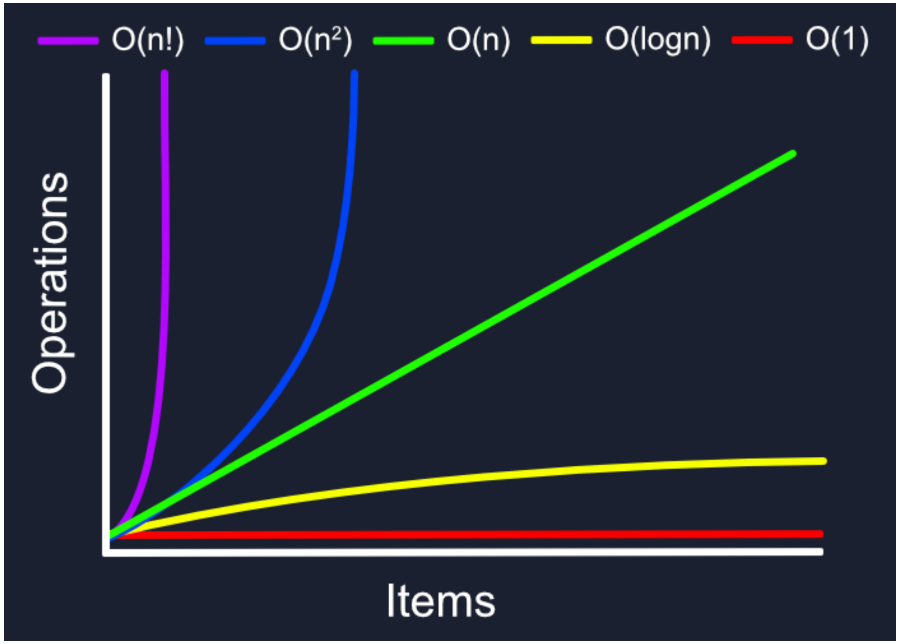
- La función $f(n) = O\big(g(n)\big)$ si y solo si existe las constantes positivas c y n_0 tal que $f(n) \leq c * g(n)$ para todo n, $n \geq n_0$

- Ejemplos:

$$3n + 2 = O(n) \quad \rightarrow \quad 3n + 2 \leq 4n, \qquad para\ todo\ n \geq 2$$
$$3n + 3 = O(n) \quad \rightarrow \quad 3n + 3 \leq 4n, \qquad para\ todo\ n \geq 3$$
$$10n^2 + 4n + 2 = O(n^2) \quad \rightarrow 10n^2 + 4n + 2 = 11n^2,$$
$$para\ todo\ n \geq 5$$

De acuerdo a la notación O

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4,096	65,536
5	32	160	1,024	32,768	4,294,967,296



En tiempo…

En una computadora de 1 billón de instrucciones por segundo

	Time for $f(n)$ instructions on a 10^9 instr/sec computer						
n	$f(n) = n$	$f(n) = n \log_2 n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = n^4$	$f(n) = n^{10}$	$f(n) = 2^n$
10	.01 μs	.03 μs	.1 μs	1 μs	10 μs	10 s	1 μs
20	.02 μs	.09 μs	.4 μs	8 μs	160 μs	2.84 hr	1 ms
30	.03 μs	.15 μs	.9 μs	27 μs	810 μs	6.83 d	1 s
40	.04 μs	.21 μs	1.6 μs	64 μs	2.56 ms	121.36 d	18.3 min
50	.05 μs	.28 μs	2.5 μs	125 μs	6.25 ms	3.1 yr	13 d
100	.1 μs	.66 μs	10 μs	1 ms	100 ms	3171 yr	$4*10^{13}$ yr
1,000	1 μs	9.96 μs	1 ms	1 s	16.67 min	$3.17*10^{13}$ yr	$32*10^{283}$ yr
10,000	10 μs	130 μs	100 ms	16.67 min	115.7 d	$3.17*10^{23}$ yr	
100,000	100 μs	1.66 ms	10 s	11.57 d	3171 yr	$3.17*10^{33}$ yr	
1,000,000	1 ms	19.92 ms	16.67 min	31.71 yr	$3.17*10^{7}$ yr	$3.17*10^{43}$ yr	

```
Algorithm D(x, n)
{
    i := 1;
    repeat
    {
        x[i] := x[i] + 2; i := i + 2;
    } until (i > n);
    i := 1;
    while (i ≤ ⌊n/2⌋) do
    {
        x[i] := x[i] + x[i + 1]; i := i + 1;
    }
}
```