Facultad de Ingeniería
Universidad Nacional de Jujuy

Desarrollo Sistemático de Programas

Unidad 2

Análisis de Algoritmos

1ra Parte

Ing. Carlos A. Afranllie

Introducción

Análisis de Algoritmos

- Cuando un programa se va a usar repetidamente resulta importante que los algoritmos implicados sean eficientes.

- Generalmente, asociaremos eficiencia con el tiempo de ejecución del programa, y más raramente con la utilización del espacio de memoria.

- Vamos a concentrarnos en la definición de eficiencia como el tiempo de ejecución de un algoritmo en función del tamaño de su entrada. A la eficiencia de un algoritmo también se le denomina costo, rendimiento o complejidad del algoritmo.

Costo de un Algoritmo

Para medir el coste de un algoritmo se pueden emplear dos enfoques:

- **Pruebas:** (también llamadas benchmarking en inglés) consiste en elaborar una muestra significativa o típica de los posibles datos de entrada del programa, y tomar medidas cuidadosas del tiempo de ejecución del programa para cada uno de los elementos de la muestra.

- **Análisis:** emplea procedimientos matemáticos para determinar el costo del algoritmo; sus conclusiones son fiables, pero a veces su realización es difícil o imposible a efectos prácticos.

Costo de un Algoritmo

El análisis de algoritmos mediante el uso de herramientas como por ejemplo la evaluación de costos intenta determinar que tan eficiente es un algoritmo para resolver un determinado problema. En general el aspecto más interesante a analizar de un algoritmo es su costo.

Análisis de Algoritmos

Costo de un Algoritmo

• **Costo de tiempo:** Suele ser el más importante, indica cuanto tiempo insume un determinado algoritmo para encontrar la solución a un problema, se mide en función de la cantidad o del tamaño de los datos de entrada.

• **Costo de espacio:** Mide cuanta memoria (espacio) necesita el algoritmo para funcionar correctamente.

Importante: *El análisis del costo de un algoritmo comprende el costo en tiempo y en espacio.*

Costo de un Algoritmo

1. El primer paso que debe llevarse a cabo para analizar un algoritmo es definir con precisión lo que entendemos por tamaño de los datos de entrada. Normalmente tal definición resulta natural y no es complicado dar con ella.

   Por ejemplo:

Costo de un Algoritmo

- En un algoritmo de ordenación, el tamaño de la entrada es el número de elementos a ordenar.

- En un algoritmo de búsqueda, el tamaño de la entrada es el número de elementos entre los que hay que buscar uno dado.

- En un algoritmo que actúa sobre un conjunto, el tamaño de la entrada es el número de elementos que pertenecen al conjunto.

Costo de un Algoritmo

2. Una vez definido el tamaño resulta conveniente usar una función **T(n)** para representar el número de unidades de tiempo (segundos, milisegundos,...) que un algoritmo tarda en ejecutarse cuando se le suministran unos datos de entrada de tamaño ***n***.

Costo de un Algoritmo

Como el tiempo de ejecución de un programa puede variar sustancialmente según el computador concreto en que se lleve a cabo la ejecución, resulta preferible que **T(n)** sea el número de instrucciones simples (asignaciones, comparaciones, operaciones aritméticas, etc.) que se ejecutan o equivalentemente, el tiempo de ejecución del algoritmo en un computador idealizado, donde cada asignación, comparación, lectura, etc. consume 1 unidad de tiempo.

Modelos computacionales

- **Maquina RAM de costo fijo:** La máquina RAM proviene de Random Access Memory. Y es una máquina ideal muy similar a una computadora actual aunque con algunas simplificaciones. Los programas de la máquina RAM se almacenan en memoria. Puede suponerse que todos los accesos a memoria tienen el mismo costo (en tiempo) y que todas las instrucciones tienen un costo constante e idéntico (en tiempo).

Modelos computacionales

- **Maquina RAM de costo Variable:** En ésta el concepto es similar a la anterior salvo que sus costos no son fijos.

Máquina RAM de costo fijo

Ejemplo 1

```
Algoritmo 1 - Este es un ejemplo
        y <- 1              /* Asignación
        A[2] <- 1           /* Asignación a un elemento de un vector
        si n < 1 entonces
                X <- w
        sino
                X <- y
        fin_si
        mientras n >= 0 hacer
                n  <- n -1
        fin_mientras
        para i desde 0 hasta 10 hacer
                A[i] <- 0
        fin_para
Fin_Algoritmo
```

Máquina RAM de costo fijo

Ejemplo 2

Algoritmo 2 - Ejemplo simple

```
        a  <- 3
        b  <-  a * 5
        si b = 5 entonces
                a  <- 2
        sino
                a  <- 1
        fin_si
Fin_Algoritmo
```

El **algoritmo 2** tiene 5 instrucciones de las cuales se ejecutan únicamente 4. Por lo tanto el costo del programa en tiempo es de C*4, siendo C una constante que indica cuanto tiempo tarda en ejecutarse una instrucción. El espacio que necesita el programa es el espacio ocupado por las variables A y B. Es decir 2*Ec siendo Ec el espacio que ocupa una variable.

Máquina RAM de costo variable

Ejemplo

Algoritmo 3 - Ejemplo simple

```
        a  <- 3
        b  <-  a * 5
        si b = 5 entonces
                a  <- 2
        sino
                a  <- 1
        fin_si
Fin_Algoritmo
```

Este programa cuesta ahora 3*C1+1*C2+1*C3.
Donde C1 = costo de una asignación, C2 = costo de una multiplicación y C3 = costo de comparar dos números.

El costo del espacio en la máquina RAM de costo variable es igual al de la máquina RAM de costo fijo.

Algoritmos iterativos

Los algoritmos iterativos son aquellos que se basan en la ejecución de ciclos; que pueden ser de tipo for, while, repeat, etc.

La gran mayoría de los algoritmos tienen alguna parte iterativa y muchos son puramente iterativos.

Analizar el costo en tiempo de estos algoritmos implica entender cuantas veces se ejecuta cada una de las instrucciones del algoritmo y cual es el costo de cada una de las instrucciones.

Algoritmos iterativos

Ordenamiento Uno contra Todos

Uno de los temas importantes del curso es el análisis de algoritmos de ordenamiento, por lo que vamos a empezar con uno de los más simples: el de uno contra todos.

```
Algoritmo Uno_contra_todos (A,n). Ordena el vector A.
        para i desde 1 hasta (n -1) hacer
                para  j desde (i + 1) hasta n hacer
                        si A[j] < A[i] entonces
                                Swap(A[i], A[j])
                        fin_si
                fin_para
        fin_para
Fin_Algoritmo
```

Algoritmos iterativos

Ordenamiento Uno contra Todos

Ejemplo:			
3 4 1 5 2 7 6 4	**Compara A[1] con A[2]**	**1 2 4 5 3 7 6 4**	**Compara A[3] con A[5]**
3 4 1 5 2 7 6 4	**Compara A[1] con A[3]**	**1 2 3 5 4 7 6 4**	**Compara A[3] con A[6]**
1 4 3 5 2 7 6 4	**Compara A[1] con A[4]**	**1 2 3 5 4 7 6 4**	**Compara A[3] con A[7]**
1 4 3 5 2 7 6 4	**Compara A[1] con A[5]**	**1 2 3 5 4 7 6 4**	**Compara A[3] con A[8]**
1 4 3 5 2 7 6 4	**Compara A[1] con A[6]**	**1 2 3 5 4 7 6 4**	**Compara A[4] con A[5]**
1 4 3 5 2 7 6 4	**Compara A[1] con A[7]**	**1 2 3 4 5 7 6 4**	**Compara A[4] con A[6]**
1 4 3 5 2 7 6 4	**Compara A[1] con A[8]**	**1 2 3 4 5 7 6 4**	**Compara A[4] con A[7]**
1 4 3 5 2 7 6 4	**Compara A[2] con A[3]**	**1 2 3 4 5 7 6 4**	**Compara A[4] con A[8]**
1 3 4 5 2 7 6 4	**Compara A[2] con A[4]**	**1 2 3 4 5 7 6 4**	**Compara A[5] con A[6]**
1 3 4 5 2 7 6 4	**Compara A[2] con A[5]**	**1 2 3 4 5 7 6 4**	**Compara A[5] con A[7]**
1 2 4 5 3 7 6 4	**Compara A[2] con A[6]**	**1 2 3 4 5 7 6 4**	**Compara A[5] con A[8]**
1 2 4 5 3 7 6 4	**Compara A[2] con A[7]**	**1 2 3 4 4 7 6 5**	**Compara A[6] con A[7]**
1 2 4 5 3 7 6 4	**Compara A[2] con A[8]**	**1 2 3 4 4 6 7 5**	**Compara A[6] con A[8]**
1 2 4 5 3 7 6 4	**Compara A[3] con A[4]**	**1 2 3 4 4 5 7 6**	**Compara A[7] con A[8]**
		1 2 3 4 4 5 6 7	**FIN.**

Para un vector de 8 elementos el algoritmo insumió 28 comparaciones. Antes de analizar en forma genérica el costo en tiempo del algoritmo veamos ejemplos más simples.

Costo de una instrucción en ciclos simples

para i desde 1 hasta n **hacer**
    Instrucción
fin_para

El cálculo en costo de una instrucción en un ciclo simple puede hacerse utilizando una sumatoria.

$$\sum_{i=1}^{n} C1$$

Donde C1 es el costo de la instrucción que se ejecuta en el ciclo. El resultado de la sumatoria es n * C1, lo cual es evidente porque la instrucción dentro del ciclo se ejecuta n veces.

Algoritmos iterativos

Costo de una instrucción en ciclos anidados independientes

```
para i desde 1 hasta n hacer
        para j desde 3 hasata m hacer
                instrucción
        fin_para
fin_para
```

Podemos calcular el costo de la instrucción nuevamente usando sumatorias de la forma:

$$\sum_{i=1}^{n}\sum_{j=3}^{m} C1 = \sum_{i=1}^{n}(m-2)*C1 = n*(m-2)*C1$$

Algoritmos iterativos

Costo de una instrucción en ciclos anidados dependientes

para i desde 1 hasta n **hacer**
    para j desde i hasata n **hacer**
        instrucción
    fin_para
fin_para

$$\sum_{i=1}^{n}\sum_{j=i}^{n} C1 = \sum_{i=1}^{n}(n-i+1)*C1 = C1*\sum_{i=1}^{n}(n-i+1) =$$

$$= C1*(\sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1) = C1*(n*n - \frac{n*(n+1)}{2} + n) = C1*(n^2 - \frac{n^2+n}{2} + n) =$$

$$= C1*(\frac{2n^2 - n^2 - n + 2n}{2}) = C1*\frac{n^2+n}{2}$$

Algoritmos iterativos

Sumatorias Útiles

Antes de continuar vamos a recordar algunas sumatorias útiles.

$$\sum_{i=1}^{n} c = c * n$$

$$\sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$$

$$\sum_{i=1}^{n} \frac{1}{i} = \ln n + 1|$$

Algoritmos iterativos

Análisis del algoritmo Uno contra Todos

```
Algoritmo Uno_contra_todos (A,n). Ordena el vector A.
        para i desde 1 hasta (n -1) hacer
                para  j desde (i + 1) hasta n hacer
                        si A[j] < A[i] entonces
                                Swap(A[i], A[j])
                        fin_si
                fin_para
        fin_para
Fin_Algoritmo
```

El costo del algoritmo lo vamos a calcular suponiendo que C1 es el costo de efectuar la comparación y que C2 es el costo de efectuar el SWAP. Sin embargo el SWAP no se hace siempre sino que se hace únicamente cuando la comparación da A[j] > A[i], por ello debemos particionar el análisis en tres casos: el peor caso, el mejor caso y el caso medio.

Algoritmos iterativos

Peor caso

En el peor caso el algoritmo siempre hace el Swap. Esto ocurre por ejemplo cuando el vector viene ordenado en el orden inverso al que utilizamos.

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (C1 + C2)$$

$$T(n) = \sum_{i=1}^{n-1} (C1 + C2) * (n - (i+1) + 1)$$

$$T(n) = \sum_{i=1}^{n-1} (C1 + C2) * (n - i)$$

$$T(n) = (C1 + C2) * (\sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i)$$

Algoritmos iterativos

$$T(n) = (C1 + C2) * ((n-1) * n - \frac{(n-1) * n}{2})$$

$$T(n) = (C1 + C2) * (n^2 - n - \frac{n^2 - n}{2})$$

$$T(n) = (C1 + C2) * (\frac{2n^2 - 2n - n^2 + n}{2})$$

$$T(n) = (C1 + C2) * (\frac{n^2 - n}{2})$$

$$T(n) = (C1 + C2) * (\frac{1}{2}n^2 - \frac{1}{2}n)$$

Por ejemplo para N = 8 el peor caso nos da: (C1+C2)*(32–4)=(C1+C2)*28 un total de 28 comparaciones y 28 Swaps.

Algoritmos iterativos

Mejor caso

En el mejor caso el vector ya viene ordenado por lo que nunca efectúa ningún Swap, el costo total es entonces el costo de las comparaciones que es igual a lo calculado antes.

$$T(n) = C1 * (\frac{1}{2} n^2 - \frac{1}{2} n)$$

Caso medio

En el caso medio el algoritmo efectúa todas las comparaciones y solo la mitad de los Swaps.

$$T(n) = C1 * (\frac{1}{2} n^2 - \frac{1}{2} n) + C2 * \frac{1}{2} (\frac{1}{2} n^2 - \frac{1}{2} n)$$

Importante: *Al analizar el tiempo de un algoritmo se debe analizar el mejor caso, el peor caso y el caso medio. Habitualmente el que más interesa es el peor caso.*