Análisis de Algoritmos

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Maestría en Ciencias de la Computación
Análisis y Diseño de Algoritmos
MCOM 20300

Otoño 2020

Contenido

Motivación I

- Cuando encuentra que varios algoritmos distintos solucionan el mismo problema, tiene que decidir cúal es el más apropiado para la aplicación que desea desarrollar.

Motivación I

- Una herramienta esencial para este propósito es el **análisis de algoritmos**.

Motivación I

- Sólo después de que ha determinado la eficiencia de los distintos algoritmos será capaz de hacer una decisión bien informada.

Motivación II

- El problema es que no hay una fórmula mágica para analizar la eficiencia de los algoritmos.

Motivación II

- Principalmente el análisis es un asunto de juicio, intuición y experiencia.

Motivación II

- No obstante, existen algunas técnicas básicas que son útiles con frecuencia, tales como saber de que forma tratar con estructuras de control y ecuaciones de recurrencia.

Introducción

- El análisis de algoritmos usualmente se lleva a cabo desde adentro hacia afuera.

Introducción

- Primero determinamos el tiempo requerido por las instrucciones individuales (este tiempo frecuentemente está acotado por una constante).

Introducción

- Luego combinamos estos tiempos de acuerdo a las estructuras de control que componen las instrucciones en el programa.

Introducción

- Algunas estructuras de control tales como las secuencias (poner una instrucción después de otra) son fáciles de analizar, mientras que otras, como los ciclos **while**, son más difíciles.

Secuencia I

- Sean P_1 y P_2 dos fragmentos de un algoritmo. Estos pudieran ser instrucciones simples o subalgoritmos complejos.

Secuencia I

- Sean t_1 y t_2 los tiempos que consumen P_1 y P_2, respectivamente. Los tiempos pudieran depender de varios parámetros, tales como el tamaño del ejemplar.

Secuencia I

- Sean t_1 y t_2 los tiempos que consumen P_1 y P_2, respectivamente. Los tiempos pudieran depender de varios parámetros, tales como el tamaño del ejemplar.
- La **regla de secuenciación** dice que el tiempo requerido para calcular "$P_1; P_2$", esto es, primero P_1 y después P_2, simplemente es $t_1 + t_2$. Por la regla del máximo, este tiempo está en $\Theta(\max(t_1, t_2))$.

Secuencia II

- A pesar de su simplicidad, aplicar esta regla algunas veces es menos obvio de lo que pudiera parecer.

Secuencia II

- Por ejemplo, pudiera suceder que uno de los parámetros que controla a t_2 depende del resultado del cálculo efectuado por P_1.

Secuencia II

- Así, el análisis de la secuencia "$P_1; P_2$" no simpre puede ser realizado considerando que P_1 y P_2 son independientes.

Ciclos for I

Los ciclos **for** son los más fáciles de analizar. Considere el siguiente ciclo.

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

Ciclos for I

Los ciclos **for** son los más fáciles de analizar. Considere el siguiente ciclo.

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

- Aquí y en el resto de nuestra discusión, adoptaremos la convención de que $m = 0$ no es un error; simplemente significa que la instrucción controlada $P(i)$ nunca se ejecuta.

Ciclos for I

Los ciclos **for** son los más fáciles de analizar. Considere el siguiente ciclo.

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

- Suponga que este ciclo es parte de un algoritmo más extenso, el cual trabaja sobre un ejemplar de tamaño n (tenga cuidado en no confundir m y n).

Ciclos for I

Los ciclos **for** son los más fáciles de analizar. Considere el siguiente ciclo.

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

- El caso más fácil es cuando el tiempo consumido por $P(i)$ realmente no depende de i, no obstante pudiera depender del tamaño del ejemplar, más generalmente, del ejemplar en sí.

Ciclos for I

- Sea t el tiempo requerido para calcular $P(i)$. En este caso, el análisis obvio del ciclo es que $P(i)$ es ejecutado m veces, cada vez con costo t, así el tiempo total requerido por el ciclo simplemente es $l = tm$, ya que:

$$\sum_{i=1}^{m} t = t \sum_{i=1}^{m} 1 = tm.$$

Ciclos for I

- A pesar de que este enfoque usualmente es adecuado, existe un peligro potencial si no tomamos en consideración el tiempo necesario para **controlar el ciclo**.

Ciclos for I

- Después de todo, nuestro ciclo **for** es una abreviación para algo como el siguiente ciclo **while**.

Ciclos for II

$i \leftarrow 1$

while $i \leq m$ **do**

$P(i)$

$i \leftarrow i + 1$

Ciclos for II

$i \leftarrow 1$

while $i \leq m$ **do**

$P(i)$

$i \leftarrow i + 1$

- En la mayoría de las situaciones es razonable contar como de costo unitario la prueba $i \leq m$, la instrucción $i \leftarrow 1$ y $i \leftarrow i + 1$ y las operaciones de secuenciación (**go to**) implícitas en el ciclo **while**.

Ciclos for II

$i \leftarrow 1$

while $i \leq m$ **do**

$P(i)$

$i \leftarrow i + 1$

- En la mayoría de las situaciones es razonable contar como de costo unitario la prueba $i \leq m$, la instrucción $i \leftarrow 1$ y $i \leftarrow i + 1$ y las operaciones de secuenciación (**go to**) implícitas en el ciclo **while**.
- Sea c una cota superior sobre el tiempo requerido por cada una de estas operaciones.

Ciclos for III

$i \leftarrow 1$

while $i \leq m$ **do**

$P(i)$

$i \leftarrow i + 1$

El tiempo l consumido por el ciclo está acotado superiormente por

$$
\begin{array}{rcll}
l & \leq & c & \text{para } i \leftarrow 1 \\
 & + & (m+1)c & \text{para las pruebas } i \leq m \\
 & + & mt & \text{para las ejecuciones de } P(i) \\
 & + & mc & \text{para las ejecuciones de } i \leftarrow i+1 \\
 & + & mc & \text{para las operaciones de secuenciación} \\
 & \leq & (t+3c)m + 2c &
\end{array}
$$

Advertencias I

- Más aún, este tiempo está claramente acotado inferiormente por mt.

Advertencias I

- Si c es despreciable comparado con t, nuestro aproximado anterior de que l era aproximadamente igual a mt fue por tanto justificado, excepto por un caso crucial: $l \approx mt$ es completamente erroneo cuando $m = 0$.

Advertencias I

- Cuando estudiemos la técnica del barómetro, veremos que despreciar el tiempo requerido para el control del ciclo, puede guiarnos a errores serios en tales circunstancias.

Advertencias II

- Resista la tentación de decir que el tiempo consumido por el ciclo está en $\Theta(mt)$ con el pretexto de que a la notación Θ se le exige ser efectiva a partir de un umbral tal como $m \geq 1$.

Advertencias II

- El problema con este argumento, es que, si en efecto estamos analizando el algoritmo entero más que simplemente el ciclo **for**, el umbral implicado por la notación Θ involucra a n, el tamaño del ejemplar, en vez de m, el número de veces que pasamos por el ciclo, $m = 0$ pudiera ocurrir para valores de n arbitrariamente grandes.

Advertencias II

- Por otro lado, se puede demostrar (en los ejercicios se pide que lo haga) que si t está acotado inferiormente por una constante (lo cual simpre es el caso en la práctica) y si existe un umbral n_0 tal que $m \geq 1$ siempre que $n \geq n_0$, entonces l en verdad está en $\Theta(mt)$ cuando l, m y t son considerados como funciones de n.

Ciclos for IV

- El análisis del ciclo **for** es más interesante cuando el tiempo $t(i)$ requerido para $P(i)$ varía como una función de i (en general, el tiempo requerido para $P(i)$ pudiera depender no sólo de i sino también del tamaño n del ejemplar, incluso del ejemplar mismo).

Ciclos for IV

- Si despreciamos el tiempo consumido por el control del ciclo, lo cual usualmente es adecuado cuando $m \geq 1$, el mismo ciclo **for**

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

consume un tiempo dado no por una multiplicación sino por una suma: esto es $\sum_{i=1}^{m} t(i)$.

Ciclos for V

Ilustraremos el análisis de ciclos **for** con un algoritmo iterativo simple para calcular la secuencia de Fibonacci. Recuerde que el algoritmo es

function FIBITER(n)

```
1  i ← 1
2  j ← 0
3  for k ← 1 to n
4  do j ← i + j
5       i ← j − i
6  return j
```

Ciclos for VI

- Si contamos todas las operaciones aritméticas con costo unitario, las instrucciones dentro del ciclo **for** consumen tiempo constante.

Ciclos for VI

- Suponga que el tiempo consumido por estas instrucciones está acotado superiormente por alguna constante c.

Ciclos for VI

- Sin tomar en cuenta el control del ciclo, el tiempo consumido por el ciclo **for** está acotado superiormente por n veces esta constante: nc.

Ciclos for VI

- Ya que las instrucciones antes y después del ciclo consumen un tiempo despreciable, concluimos que el algoritmo consume un tiempo en $O(n)$. Un razonamiento similar nos lleva a que este tiempo también está en $\Omega(n)$, así está en $\Theta(n)$.

Ciclos for VII

- No obstante, vimos que no es razonable contar como de costo unitario, las adiciones involucradas en el cálculo de la secuencia de Fibonacci, al menos que n sea muy pequeño.

Ciclos for VII

- Por lo tanto, debemos tomar en cuenta el hecho de que una instrucción tan simple como "$j \leftarrow i + j$" se encarece incrementalmente cada vez que pasamos por el ciclo.

Ciclos for VII

- Es fácil programar adiciones y sustracciones de enteros largos, tal que el tiempo necesario para sumar y sustraer dos enteros, esté en el orden exacto del número de dígitos del operando más grande.

Ciclos for VIII

- Para determinar el tiempo consumido por la $k-$ésima pasada por el ciclo, necesitamos conocer la longitud de los enteros involucrados.

Ciclos for VIII

- Se puede demostrar por inducción matemática que los valores de i y j al final de la $k-$ésima iteración son respectivamente f_{k-1} y f_k.

Ciclos for VIII

- Esto es precisamente por lo que el algoritmo funciona: regresa el valor de j al final de la $n-$ésima iteración, el cual es por lo tanto f_n, como se requiere.

Ciclos for VIII

- Más aún, a partir de la fórmula de de Moivre se puede demostrar que el tamaño de f_k está en $\Theta(k)$.

Ciclos for VIII

- Por lo tanto la $k-$ésima iteración consume un tiempo $\Theta(k-1)+\Theta(k)$, lo cual es lo mismo que $\Theta(k)$.

Ciclos for IX

- Sea c una constante tal que este tiempo está acotado superiormente por ck para todo $k \geq 1$.

Ciclos for IX

- Si despreciamos el tiempo requerido por el control del ciclo y por las instrucciones antes y después del ciclo, podemos concluir que el tiempo requerido por el algoritmo está acotado superiormente por

$$\sum_{k=1}^{n} ck = c\sum_{k=1}^{n} k = c\frac{n(n+1)}{2} \in O(n^2).$$

Ciclos for IX

- Un razonamiento similar nos conduce a que este tiempo está en $\Omega(n^2)$ y por lo tanto en $\Theta(n^2)$.

Ciclos for IX

- Así, en el análisis de **Fibiter**, contar como de costo unitario a las operaciones aritméticas, constituye una diferencia crucial con respecto a no contarlas como de costo unitario.

Ciclos for X

- El análisis de ciclos **for** que inician en valores distintos a 1, o que se ejecutan a pasos más grandes, debe ser obvio en este punto.

Ciclos for X

- Por ejemplo considere el siguiente ciclo.

$$\textbf{for } i \leftarrow 5 \textbf{ to } m \textbf{ step } 2 \textbf{ do } P(i)$$

Ciclos for X

- Por ejemplo considere el siguiente ciclo.

$$\textbf{for } i \leftarrow 5 \textbf{ to } m \textbf{ step } 2 \textbf{ do } P(i)$$

- Aquí, $P(i)$ se ejecuta $((m-5) \div 2) + 1$ veces siempre que $m \geq 3$. Para que un ciclo **for** tenga sentido (se ejecute cero o más veces), el punto final debe ser siempre al menos tan grande como el punto inicial **menos** el paso.

Ejercicios I

1. ¿Cuánto tiempo requiere el siguiente "algoritmo" como función de n?

procedure DUMMY(n)
1 $l \leftarrow 0$
2 **for** $i \leftarrow 1$ **to** n
3 **do for** $j \leftarrow 1$ **to** n^2
4 **do for** $k \leftarrow 1$ **to** n^3
5 **do** $l \leftarrow l + 1$

Expresa tu respuesta en notación Θ, de la manera más simple posible. Puedes considerar que cada instrucción individual (incluyendo el control de ciclos) es elemental.

2. ¿Cuánto tiempo requiere el siguiente "algoritmo" como función de n?

procedure DUMMY'(n)
1 $l \leftarrow 0$
2 **for** $i \leftarrow 1$ **to** n
3 **do for** $j \leftarrow 1$ **to** i
4 **do for** $k \leftarrow j$ **to** n
5 **do** $l \leftarrow l + 1$

Ejercicios II

Expresa tu respuesta en notación Θ, de la manera más simple posible. Puedes considerar que cada instrucción individual (incluyendo el control de ciclos) es elemental.

3. Considera el ciclo

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P$$

que es parte de un algorimo más grande que trabaja en una instancia de tamaño n. Sea t el tiempo necesario para cada ejecución de P, que para este problema supondremos independiente de i (pero t sí podría depender de n). Demustra que este ciclo toma tiempo en $\Theta(mt)$, siempre que t está inferiormente acotado por una constante y que existe un umbral n_0 tal que $m \geq 1$ siempre que $n \geq n_0$. Recuerda que en Advertencias II se estableció que la conclusión deseada no se obtiene sin esas restricciones.

Ejercicios III

4. Demuestra por inducción matemática que los valores de i y j al final de la k-ésima iteración de FIBITER son f_{k-1} y f_k, respectivamente, en donde f_n denota el n-ésimo número de Fibonacci.

Para O tenemos: sea c la constante que acota superiormente a las operaciones de las líneas 1 y 5 del algoritmo y al control de los ciclos.

$$t(n) \leq \sum_{i=1}^{n} \sum_{j=1}^{n^2} \sum_{k=1}^{n^3} c =$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n^2} c \sum_{k=1}^{n^3} 1 = \sum_{i=1}^{n} \sum_{j=1}^{n^2} cn^3 = \sum_{i=1}^{n} cn^3 \sum_{j=1}^{n^2} 1 =$$

$$\sum_{i=1}^{n} cn^3 n^2 = cn^3 n^2 \sum_{i=1}^{n} 1 = cn^3 n^2 n = cn^6.$$

Por lo tanto, $t(n) \in O(n^6)$.

Para Ω tenemos: sea d la constante que acota inferiormente a las operaciones de las líneas 1 y 5 del algoritmo y al control de los ciclos.

$$t(n) \geq \sum_{i=1}^{n} \sum_{j=1}^{n^2} \sum_{k=1}^{n^3} d =$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n^2} d \sum_{k=1}^{n^3} 1 = \sum_{i=1}^{n} \sum_{j=1}^{n^2} dn^3 = \sum_{i=1}^{n} dn^3 \sum_{j=1}^{n^2} 1 =$$

$$\sum_{i=1}^{n} dn^3 n^2 = dn^3 n^2 \sum_{i=1}^{n} 1 = dn^3 n^2 n = dn^6.$$

Por lo tanto, $t(n) \in \Omega(n^6)$, así $t(n) \in \Theta(n^6)$.