Análisis y Diseño de Algoritmos.

Sesión 1. 19 de agosto de 2015.

Maestría en Sistemas Computacionales.

Por: Hugo Iván Piza Dávila.

Análisis de algoritmos

- Un problema → muchas soluciones
  - Si todas son igualmente **eficaces** o **precisas**, ¿cuál es la mejor?
  - La mejor es la más **eficiente**.
- Propone una Notación Matemática (análisis **a priori**):
  - Cuál algoritmo es más probable que tarde menos bajo circunstancias semejantes de HW, SW y cantidad de información.
  - De acuerdo a los recursos utilizados: número de instrucciones que se ejecutan, memoria ocupada, duración, cuál es el tamaño máximo del problema que permite la solución.

Análisis de algoritmos

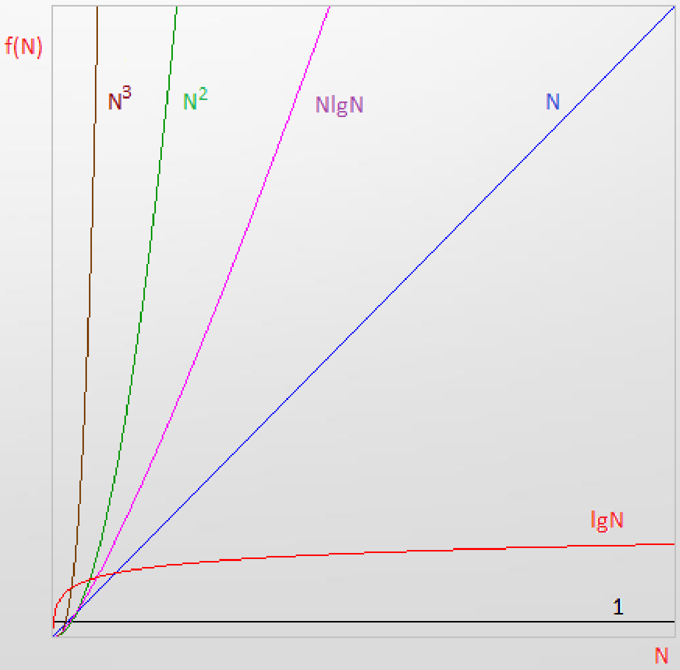
- Para el análisis, contemplaremos dos tipos de complejidad:
  - **Temporal**: número de instrucciones que se ejecutan
  - **Espacial**: memoria ocupada
- En muchos problemas estaremos interesados en analizar la *complejidad* de la solución en los casos: **mejor**, **peor** y **promedio**.
- Para muchos algoritmos ya se tiene identificado su complejidad en los tres casos.
  - Existen otros cuyo análisis es muy complicado y no hay un consenso.

Clasificación de algoritmos

- Los algoritmos de interés tienen un parámetro N
  - Representa el tamaño del problema
  - Afecta al tiempo de ejecución
- N puede representar
  - El número de filas de una matriz cuadrada
  - El tamaño de un archivo a ordenar
  - El número de nodos de un grafo
  - El grado de un polinomio …

Clasificación de algoritmos

- Tiempos de ejecución más conocidos:
  - Constante (K)
  - Logarítmico ($\log_b N$)
  - Lineal (N)
  - Quasi-lineal ($N \log_b N$)
  - Cuadrático (N^2)
  - Cúbico (N^3)
  - Exponencial (b^N)



Clasificación de algoritmos

- Constante (K)
  - Las instrucciones del algoritmo se ejecutan una cantidad fija de veces, sin importar el valor de N.
- Logarítmico ($\log_b N$)
  - Conforme N crece, el tiempo de ejecución aumenta en un factor cada vez menor.
  - Algoritmos que resuelven problemas grandes dividiéndolos en problemas más pequeños que se resuelven en tiempo constante.
  - Si b = 2, lo escribiremos **lg N**. En este caso, si se duplica N ¿qué tanto aumenta el tiempo de ejecución?

Clasificación de algoritmos

- Lineal (N)
  - Cada dato de entrada se procesa una cantidad fija de veces.
  - Si N se duplica, el tiempo de ejecución también.
- Quasi-lineal ($N \log_b N$)
  - Algoritmos que resuelven problemas grandes dividiéndolos en problemas más pequeños que se resuelven en tiempo lineal.
  - Si b = 2, N = 1024, ¿cuál es el tiempo de ejecución? ¿Si N se duplica? ¿Qué sucede con el tiempo?

Clasificación de algoritmos

- Cuadrático (N^2)
  - Algoritmos que incluyen dos ciclos anidados.
  - Se procesan todos o casi todos los pares posibles de los datos de entrada en tiempo <u>constante</u>.
  - Si N se duplica, ¿qué sucede con el tiempo?
- Cúbico (N^3)
  - Algoritmos que incluyen tres ciclos anidados.
  - Se procesan ternas de los datos, o pares de datos en tiempo <u>lineal</u>.
  - Si N se duplica, ¿qué sucede con el tiempo?

Clasificación de algoritmos

- Exponencial (b^N)
  - Algoritmos que utilizan fuerza bruta para encontrar uno o más objetos de tamaño N que cumplan algún requisito. Cada elemento del objeto admite b valores posibles.
    - Optimizar una función de N dimensiones en un espacio de búsqueda de tamaño b por cada dimensión.
    - Problema de las N reinas
    - Encontrar un camino para salir de un laberinto
  - Si N se duplica, ¿qué sucede con el tiempo?

Tipos de análisis

- Para medir la eficiencia de un algoritmo podemos llevar a cabo dos tipos de análisis:
  1. A priori
     - Se aplica en la etapa de diseño del algoritmo.
     - Obtiene una expresión matemática que limita el tiempo de cálculo, mediante la **notación asintótica**.
  2. A posteriori
     - Se realizan muchas corridas del algoritmo ya implementado, usando diferentes valores de N.
     - Se reportan estadísticas de tiempo y espacio consumidos, y el número de operaciones (relevantes) efectuadas en cada caso.

Notación asintótica

- Propósito: identificar y especificar a qué **orden de complejidad temporal y espacial** pertenece un algoritmo.
- El tiempo de ejecución de un algoritmo no será exactamente igual a alguno de los vistos, pero sí será proporcional.
  - Un algoritmo que ejecute $2N^2$ instrucciones pertenece al orden de complejidad cuadrático.
  - Difiere a razón de una constante (2) del tiempo de referencia (N^2), y se escribe así: $2N^2 \in O(N^2)$.

Notación asintótica

- Una función g(N) está en el orden f(N), $g(N) \in O(f(N))$, si existen constantes positivas c_o y N_o tal que:

$$g\,(N) \leq c_o f\,(N) \text{ para todo } N > N_o.$$

- Demostrar: $2N^2 \in O(N^2)$

$g(N) = ?$ $f(N) = ?$ $c_o = ?$ $N_o = ?$

Notación asintótica

- Demostrar: $2N^2 \in O(N^2)$

  $g(N) = 2N^2$, $f(N) = N^2$

  $c_0 = 3$, $N_0 = 1$.

  $2N^2 < 3N^2$ se cumple para todo $N > 1$

  $[N = 2]\ 8 < 12$, $[N = 3]\ 18 < 27$, …

  $\therefore$ **$2N^2 \in O(N^2)$**

P vs NP

- Si el orden de complejidad de un algoritmo es **Exponencial**, decimos que el problema es resuelto en un tiempo **No Polinomial (NP)**.
- Para las otras seis clases, el tiempo es **Polinomial (P)**.
- Un tiempo es Polinomial si se puede expresar mediante una ecuación en donde N está involucrado sólo en: sumas, restas, multiplicaciones, divisiones y/o logaritmos.
  - Por ejemplo: $3N^2 - 2 \lg N + 5N$.

Ejercicios

- Demuestre que: $g(N) = 4N \in O(N^2)$.
- Demuestre que: $g(N) = 2N^4 \notin O(N^3)$.
- Demuestre que: $g(N) = N\log_2 N + 5N \in O(N^2)$

Analizando un algoritmo

- Un algoritmo está compuesto por una o más instancias de lo siguiente:
  1. Declaraciones de variables, estructuras, funciones, ...
  2. Operaciones de asignación, aritméticas, relacionales, lógicas y a nivel de bits: =, +, -, *, /, %, +=, ++, >, ==, !, &&, ||, >>, &, |, ...
  3. Operaciones de E/S y acceso a arreglos: printf, cin, cout, a[i] ...
  4. Estructuras selectivas: if, switch, else if, case, ...
  5. Estructuras iterativas: for, while, do/while, repeat, ...
  6. Llamadas a funciones

Analizando un algoritmo

- Para el análisis de complejidad temporal, consideraremos que la duración de cualquiera de **los tres primeros** (declaraciones y operaciones) estará limitada por una constante K.
  - Su duración no dependerá del tamaño del problema (N)
  - Para simplificar el análisis, podemos asignar **1** a la duración, lo cual no afectará al resultado: *a qué* ***orden de complejidad temporal*** *pertenece la solución*.
  - El propósito no es medir cuánto tarda un algoritmo en efectuar una operación aritmética (dependiente del HW), sino cuántas operaciones aritméticas tuvo que efectuar para llegar a la solución.

Analizando un algoritmo

- La duración de las **estructuras selectivas** estará en función de cuál camino se haya elegido.
  - Por ello, contemplamos los casos: mejor, peor y promedio.
  - Si analizamos el peor caso, la duración será igual a una constante (expresión lógica que define el camino) más la duración del camino posible más tardado.
- La duración de las **llamadas a funciones** será igual a una constante más la duración de la función misma.

Analizando un algoritmo

- La duración de las **estructuras repetitivas** estará en función del tamaño del problema (N):
  - `for(i = 0; i < N; i ++) cout << i;`
    - Una operación de asignación: `i = 0`
    - N + 1 operaciones relaciones: `i < N`
    - N operaciones aritméticas: `i ++`
    - Como se ejecutan N iteraciones, son N operaciones E/S: `cout << i`
    - Duración: `1 + (N + 1) + N + N = 3N + 2`

Analizando un algoritmo

- ¿Cuál es la duración de un algoritmo que calcula la desviación estándar de un arreglo de N números dado?

$$\sigma^2 = {}^{1}\!/_{N} \sum_{i=1}^{N} (x_i - \hat{x})^2$$

```
double prom = 0, desv = 0;
for(int i = 0; i < N; i ++)
    prom += arr[i];
prom /= N;
for(int i = 0; i < N; i ++)
    desv += (arr[i] – prom) * (arr[i] - prom);
desv = sqrt(desv / N);
```

Analizando un algoritmo

- ¿Cuál es la duración de un algoritmo que calcula la desviación estándar de un arreglo de N números dado?

$$\sigma^2 = {}^1/_N \sum_{i=1}^{N} (x_i - \hat{x})^2$$

`double prom = 0, desv = 0;`	2
`for(int i = 0; i < N; i ++)`	1+(N+1)+N
`    prom += arr[i];`	N (2)
`prom /= N;`	1
`for(int i = 0; i < N; i ++)`	1+(N+1)+N
`    desv += (arr[i] – prom) * (arr[i] - prom);`	N (6)
`desv = sqrt(desv / N);`	2

Analizando un algoritmo

- En conclusión, se necesitan **12N + 9** pasos para calcular la desviación estándar de un arreglo de N elementos.
- Demostrar que el orden de complejidad temporal de la solución es **lineal**, es decir, **12N + 9** $\in$ **O(N)**.

Analizando un algoritmo

- Ahora, ¿cuánta memoria ocupa el algoritmo que calcula la desviación estándar de un arreglo de N números dado?

```
double prom = 0, desv = 0;
for(int i = 0; i < N; i ++)
    prom += arr[i];
prom /= N;
for(int i = 0; i < N; i ++)
    desv += (arr[i] – prom) * (arr[i] - prom);
desv = sqrt(desv / N);
```

Analizando un algoritmo

- ¿Ahora, ¿cuánta memoria ocupa el mismo algoritmo?
  - Aquí consideraremos el espacio máximo ocupado a la vez: por eso, contabilizamos a la variable **i** sólo una vez.

```
double prom = 0, desv = 0;
for(int i = 0; i < N; i ++)
    prom += arr[i];
prom /= N;
for(int i = 0; i < N; i ++)
    desv += (arr[i] – prom) * (arr[i] - prom);
desv = sqrt(desv / N);
```

2: prom, desv
1: i
N: arr

Analizando un algoritmo

- En conclusión, se necesitan **N + 3** casillas de memoria para calcular la desviación estándar de un arreglo de N elementos.
  - N para guardar la lista
  - 3 para la guardar la media, la desviación estándar y el contador i.
- Demostrar que el orden de complejidad espacial de la solución es **lineal**, es decir, **N + 3** ∈ **O(N)**.

Ejercicios

- ¿A qué orden de complejidad temporal y espacial pertenecen los algoritmos conocidos que resuelven los siguientes problemas?
  1. Resta de matrices de N x N.
  2. Sumatoria de los primeros N números naturales.
  3. Determinar si el elemento situado a la mitad de un arreglo de N números es divisible entre 3.
  4. Cálculo del producto punto de dos vectores de tamaño N.
  5. Almacenar todos las combinaciones posibles <Camisa, Pantalón, Zapatos> existiendo N_1 camisas, N_2 pantalones, N_3 zapatos ($N_1 \approx N_2 \approx N_3$).
  6. Dado un cajón con N artículos de joyería, ¿de cuántas maneras diferentes puedo llenar un estuche al que le caben sólo 2 artículos?
  7. Almacenar cada una de las maneras diferentes a que refiere el problema 6.

Tarea (parte 1)

- ¿A qué orden de complejidad temporal y espacial pertenecen los algoritmos conocidos que resuelven los siguientes problemas? Justificar su respuesta.
  1. Determinación de igualdad de dos cadenas de texto.
  2. Cálculo de la mediana en una lista desordenada de números enteros.
  3. Multiplicación de matrices.
  4. Conteo de los números primos en el rango [a … b].
  5. Encontrar el número de veces en que se tiene que dividir un número entero entre 3 hasta llegar a la unidad.
  6. Encontrar el número de agrupaciones de N dígitos (iguales o diferentes) que sumados no sean mayores a M. Considerar por ejemplo: $\{0, 1, 2, 3\} \neq \{1, 0, 2, 3\}$
  7. Registrar todas los pares (a, b) de números enteros (de 1 a N) que satisfagan la desigualdad: $\cos(a) \cdot \sin(b) \leq b / 2a$

Más de notación asintótica

- Recordando las ecuaciones resultantes de los análisis de complejidad espacial y temporal del algoritmo de la desviación estándar.
- ¿Se cumplen: **$12N + 9$** $\in$ **$O(N^2)$**, **$N + 3$** $\in$ **$O(N \log N)$**?
- Demostrando el primero (el segundo queda para el alumno):

$g(N) = 12N + 9$, $f(N) = N^2$

$c_0 = \mathbf{12}$, $N_0 = \mathbf{1}$.

$12N + 9 < \mathbf{12}N^2$ se cumple para todo $N > \mathbf{1}$

$[N = 2]\ 33 < 48$, $[N = 3]\ 45 < 108$, $[N = 4]\ 57 < 192$, …

$\therefore 12N + 9 \in O(N^2)$ … ¿Entonces $12N + 9$ es cuadrático?

Más de notación asintótica

- Lo que la notación O dice que la complejidad de un algoritmo nunca será mayor que una función de referencia dada.
  - Si $f(N) \in O(g(N)) \rightarrow f(N)$ nunca será mayor que $g(N)$
- Para ciertos algoritmos, el número de pasos que se necesitan para resolver un problema varía en función de los datos de entrada.
  - Esta variación puede ocasionar que la complejidad sea diferente.
- Los algoritmos más conocidos susceptibles a esta variación son los de **ordenamiento** y **búsqueda**.

Más de notación asintótica

- Existen notaciones complementarias para atender la variación en complejidad de los algoritmos:
  - La notación Ω define un *límite inferior*: **la complejidad no será menor que** (o mejor que)
  - La notación Θ define *límites inferior y superior*: **la complejidad no será menor ni mayor que** (no hay pierde)
- Definición formal:
  - $g(N) \in \Omega(f(N))$ si existen constantes positivas c_0 y N_0 tal que $\boldsymbol{g(N) \geq c_0 f(N)}$ para todo $N > N_0$
  - $g(N) \in \Theta(f(N))$ si existen constantes positivas c_0, c_1 y N_0 tal que $\boldsymbol{c_0 f(N) \leq g(N) \leq c_1 f(N)}$ para todo $N > N_0$

Ejercicios

- Nótese que se cumplen:
  1. $g(N) \in \Omega(f(N)) \Leftrightarrow f(N) \in O(g(N))$
  2. $g(N) \in \Omega(f(N)) \wedge g(N) \in O(f(N)) \Leftrightarrow g(N) \in \Theta(f(N))$
- Demuestre que: $\frac{1}{2}$ N lg N $\in \Omega(N)$.
- Demuestre que: $2N^3 + 4N^2 \in \Theta(N^3)$.

Tarea (parte 2)

- **Algoritmo de Euclides**: calcula el máximo común divisor de dos números enteros A, B
  - Primer algoritmo *interesante* de la historia.
  - complejidad temporal y espacial en el peor caso: O(lg n).
    - A y B son dos números consecutivos de la serie de Fibonacci.
- Comprobar de forma práctica (*a posteriori*) tal complejidad:

8. Implementarlo en su lenguaje de programación favorito (no más de 4 líneas de código). Suponer A > B.

9. Contar el número de divisiones que toma el cálculo GCD(A, B), donde A = Fibonacci(n), B = Fibonacci(n – 1), para n = 2 hasta 16, y reportarlo en una tabla.

Tarea (parte 2)

10. Apoyado de Excel, crear una gráfica de dispersión (ó XY) tomando A como las abscisas y el conteo de divisiones de GCD(A, B) como las ordenadas.
11. Sobre los datos de la gráfica, agrega una línea de tendencia (*trendline*). El tipo de tendencia debe ser *logarítmica*. Seleccionar la opción *Presentar ecuación en el gráfico*.
12. Con la ecuación mostrada, demuestre: $g(N) \in O(\lg N)$.

    [PILÓN]

13. Identifique el mejor caso y su complejidad g(N).
14. ¿Se cumple $g(N) \in \Omega(\lg N)$?

Documento de la tarea

- Subir al buzón un solo archivo en formato PDF que incluye:
  - Portada:
    - Fecha, integrantes del equipo. *Tarea 1: Análisis de algoritmos*
  - Por cada pregunta (1, 14):
    - Número y descripción de la pregunta
    - La respuesta, que puede ser:
      - Orden de complejidad (espacial y temporal)
      - Explicaciones breves y sin ambigüedades
      - Código fuente (correctamente indentado)
      - Gráficas, tablas, demostraciones
  - Referencias bibliográficas o electrónicas