Análisis y Diseño de Algoritmos

Introducción

Arturo Díaz Pérez

Sección de Computación
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. CP 07300

Tel. 5061 3800 Ext. 6562, 6660
e-mail: adiaz@cs.cinvestav.mx

"People who analyze algorithms have double happiness. First of all they experience the sheer beauty of elegant mathematical patterns that surround elegant mathematical computational procedures. Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically.

Mathematical models have been a crucial inspiration for all scientific activity, even though they are only approximate idealizations of real-world phenomena. Inside a computer, such models are more relevant than ever before, because computer programs create artificial worlds in which mathematical models often apply precisely. I think that´s why I got hooked on analysis of algorithms when I was a graduate student, and why the subject has been my main life´s work ever since".

D. E. Knuth

Revisión Histórica

Contenido

- Análisis de Algoritmos
  - Identificar la complejidad de diferentes algoritmos para resolver el mismo problema
- Diseño de Algoritmos
  - Reconocer diferentes estrategias para resolver un problema
- Computabilidad
  - Definición de los conceptos: computable, intratable
- Complejidad
  - Identificar las jerarquía de clases de problemas
    - Problemas polinomiales, no polinomiales

Revisión Histórica: Los Primeros Años

- Modelos formales de cómputo (1936)
  - Alan Turing: Máquina de Turing
  - Alonzo Church: Cálculo lambda
  - Claude Shannon: Aplicación del álgebra Booleana a la modelación de circuitos
    - Teoría matemática de la comunicación
- Primeras computadoras
  - Arquitectura Von Neumann
  - Z3 y ENIAC
- Primeros lenguajes de programación
  - FORTRAN, COBOL y LISP

Revisión Histórica: 50´s

- Máquinas de estados finitos
  - Rabin y Scott desarrollaron herramientas analíticas para el estudio de las capacidades y limitaciones de las máquinas de estados finitos y autómatas de pila
- Lenguajes Formales
  - Noam Chomsky: La jerarquía de lenguajes
    - Lenguajes regulares, lenguajes libres del contexto, lenguajes sensibles al contexto y lenguajes recursivamente enumerables

Revisión Histórica: 60´s

- Complejidad Computacional
  - Clasificación de funciones en tiempo y espacio: Complejidad computacional
  - Jerarquías de problemas: P y NP
  - Cook y Levine: problemas NP completos
  - Conjetura P = NP

Revisión Histórica: 70´s

- Tiempo de cómputo y complejidad de circuitos
  - Conexión entre máquinas de Turing y complejidad de circuitos
- Semántica de los lenguajes de programación
  - Métodos formales para el estudio de los programas y lenguajes
  - Notaciones y modelos para dar un significado a la frase "lenguaje de programación"
  - Métodos formales para asegurar la correctitud de programas
  - Modelo relacional: cálculo relacional para hacer consultas

Revisión Histórica: 70´s

- Balance espacio-tiempo
- Modelos basados en VLSI
- Algoritmos y estructuras de datos
  - Knuth
  - Aho, Hopcroft y Ullman

Revisión Histórica: 80´s

- Procesamiento paralelo y distribuido
  - Modelos para algoritmos paralelos. PRAM
  - Modelos para arquitecturas paralelas
    - Síncronas vs. Asíncronas
    - SIMD vs. MIMD
  - Modelos para evaluar la complejidad de entrada/salida de un algoritmo
    - El estudio del movimiento de datos entre unidades de memoria de una jerarquía
    - vecindad espacial y temporal

Revisión Histórica: 90´s

- Criptografía
  - ← Intercambio de información en forma segura
  - ← Llaves públicas y privadas
  - ← Autentificación

¿Qué hay acerca 200x?

- Métodos de aproximación
- Métodos de optimización combinatoria
- Computación evolutiva
- Computación cuántica
- No determinismo

Preliminares Matemáticos

Preliminares Matemáticos

- Teoría de conjuntos
  - cardinalidad, unión, intersección, diferencia
  - el conjunto potencia
  - producto Cartesiano
- Sistemas de numeración
  - $x_{k-1}b^{k-1} + x_{k-2}b^{k-2} + \ldots + x_1 b^1 + x_0 b^0$

Preliminares Matemáticos

- Lenguajes y cadenas de símbolos
  - Alfabetos, cadenas de símbolos, concatenación
  - Cadena vacía
  - Cerradura de Kleene y cerradura positiva
- Relaciones
  - Relación binaria
  - Propiedades: reflexividad, simetría, transitividad
  - Relación de equivalencia, clases de equivalencia

Preliminares Matemáticos

- Gráficas
  - Vértices y aristas
  - Dirigidas, no-dirigidas
  - Grado de un vértice
  - Camino, longitud de un camino. Ciclos
  - Gráficas dirigidas acíclicas

Preliminares Matemáticos

- Matrices
  - ← Determinante, inversa, suma, producto
- Funciones
  - ← Dominio, imagen
  - ← *n* variables de entrada, *m* variables de salida
  - ← Función Booleana

Algoritmos

Algoritmo

- Un algoritmo es una especificación de un proceso funcional. Es un conjunto finito de instrucciones que gobiernan el funcionamiento de un modelo de cómputo para resolver un problema.
  - Un algoritmo siempre termina
- Un algoritmo está relacionado a un problema
- Un algoritmo está "relacionado" a un modelo computacional

¿Qué es un buen algoritmo?

- Criterios
  - Simplicidad
    - Un algoritmo simple es más fácil de implementar correctamente que un algoritmo complejo
  - Claridad
    - Los programas deben ser escritos claramente y documentados cuidadosamente para que puedan mantenerse por otros

¿Qué es un buen algoritmo?

- Criterios
  - Eficientes
    - La eficiencia de su tiempo de ejecución
    - La cantidad de almacenamiento (memoria) que requiere
    - La cantidad de tráfico que genera en la red de computadoras
    - La cantidad de datos que mueve hacia disco
  - Problemas grandes: el tiempo de ejecución determina si un programa puede ser usado efectivamente
  - Algoritmos eficientes tienden a ser más complejos para escribir y entender

Otros Factores

- Si el programa se va a utilizar pocas veces, se debe elegir el algoritmo que es más fácil para implementar correctamente.
- Si el programa se va a ejecutar sólo en entradas de tamaño pequeño, el índice de crecimiento puede ser menos importante que el factor constante.
- Un algoritmo complicado pero eficiente puede no ser deseable cuando una persona diferente a la que escribe el programa va a mantenerlo más tarde.

Otros Factores

- Algunos algoritmos eficientes requieren del uso de memoria secundaria lo cual puede provocar lentitud en la ejecución.
- En algoritmos numéricos, exactitud como estabilidad son tan importantes como eficiencia.

Midiendo el Tiempo de Ejecución

- Benchmarking
  - *Benchmarks*: una colección de entradas típicas representativas de una carga de trabajo para un programa
- Profiling
  - Asociar a cada instrucción de un programa un número que representa la fracción del tiempo total tomada para ejecutar esa instrucción particular
  - Regla del 90-10
    - Regla informal que afirma que el 90% del tiempo de ejecución se invierte en el 10% del código

Midiendo el Tiempo de Ejecución

- Análisis
  - Agrupar las entradas de acuerdo a su tamaño, n, y estimar el tiempo de ejecución del programa en entradas de ese tamaño, $T(n)$