Nivel Superior

TÉCNICAS DE ANÁLISIS DE ALGORITMOS

Miguel Angel Norzagaray Cosío
Universidad Autónoma de Baja California Sur[1]

Resumen

Se presentan en las siguientes notas los aspectos teóricos y metodológicos que se utilizan para analizar una amplia variedad de algoritmos, tanto iterativos como recursivos. Estas notas son sólo una guía para el curso del mismo nombre y no consideran con la misma profundidad las técnicas de diseño de algoritmos, en aras de detallar mejor las técnicas de análisis en sí. Debido a que cada algoritmo posee una estructura de datos implícita (y viceversa), se presentarán algunas estructuras que son ilustrativas por el modo como son analizadas. Ejercicios y ejemplos se presentan durante las sesiones del curso.

1 Introducción

Los algoritmos existen desde épocas muy antiguas. Pero es hasta este siglo cuando son estudiados más detalladamente por el importante papel que juegan en el mundo de las computadoras.

La primera vez que alguien se enfrenta con el estudio de los algoritmos le "platican" que un algoritmo es una secuencia lógica de pasos encaminada a resolver un problema específico. Esta dista mucho de ser una definición formal y cualquiera con pretensiones formales diría que sobre esa escasa base no se puede construir mucho.

En lo que sí se está de acuerdo al momento de definir un algoritmo es en las características que debe reunir:

- **Correctitud**: Obvio es que lo primero es asegurar que el algoritmo realiza la tarea para la que se ha diseñado.
- **Finitud**: Siempre debe terminar en una cantidad finita de pasos, es decir, garantizar que no se ejecutará eternamente.
- **Definibilidad**: Los pasos que describen al algoritmo deben estar exentos de cualquier ambigüedad.
- **Entrada y salida**: Consistiendo en la información con que se trabajará desde el principio y el resultado que se desea obtener.
- **Efectividad**: Cada paso del algoritmo debe consistir en una operación básica de ejecución exacta y finita, es decir, que cada paso sea realizable por una persona y que siempre se obtengan los mismo resultados de manera exacta.

Estas son características igualmente empíricas aunque de lo más deseables. Presentar un modelo tan completo como el de una máquina de Turing para poder definir algoritmo de manera formal ocuparía mucho espacio, pero podemos establecer la siguiente definición más breve:

Llamaremos *método de cálculo* a una cuaterna (Q,I,O,f) donde I y O son subconjuntos de Q, $f:Q \rightarrow Q$ y $f(x)=x$ si x está en O. x define la *secuencia de cálculo* $x_0, x_1, \ldots, x_{k+1}$, como $x_0=x$,

[1] Email: manc@uabcs.mx

$x_{k+1}=f(x_k)$, $k\geq 0$. Esta secuencia termina en k pasos si k es el menor entero para el cual x_k está en O.

Definimos un *algoritmo* como un método de cálculo que genera secuencias de cálculo finitas para toda x en I.

Es posible poner cualquier algoritmo en términos de esta definición, misma que puede extenderse para que también cubra la característica de efectividad, que es la única que no se garantiza (ver [3], pág. 9). La correctitud es una característica que debe probarse independientemente del modelo de computación que se emplee (ver [5], capítulo 5).

2 Análisis y diseño de algoritmos

Es importante identificar bien la diferencia entre el diseño del algoritmo y su análisis. Hay una gran cantidad de técnicas o ideas que se pueden utilizar al momento de crear un algoritmo para realizar alguna tarea. Por ejemplo técnicas como divide y vencerás, algoritmos golosos, combinatorios, backtracking, branch and bound, programación dinámica, algoritmos de aproximación, aleatorizados, genéticos o meméticos. Estas a su vez no deben confundirse con recursividad, iteración, paralelismo o tablas look-up, que son técnicas de programación (para la construcción de algoritmos).

Para cubrir bien material como el mencionado se requiere de uno o dos cursos de un semestre, dependiendo del nivel de detalle que se desee. Se presenta aquí sólo lo referente a las técnicas con las que se estudian los algoritmos y que permiten medir su desempeño en diversas situaciones, antes de que sea implantado como parte de un sistema, es decir, independientemente del lenguaje o computadora donde se aplique.

Hay varias razones por las que es deseable analizar el comportamiento de un algoritmo.

1. Analizando se pueden descubrir características generales y particulares de un algoritmo y evaluar la facilidad de emplearlo en una aplicación, o compararlo con otras opciones de algoritmo para la misma aplicación.
2. El análisis de un algoritmo sirve para entender mejor sus propiedades y puede sugerir mejoras posteriores.
3. El análisis puede servir para identificar maneras de sacar las mayores ventajas del ambiente de programación (arquitectura de la computadora, sistema operativo, compilador), que puede tener un efecto significativo en el desempeño del algoritmo.
4. Muchos algoritmos poseen una estructura compleja que despierta interés matemático y permite el desarrollo de teoría útil para otras situaciones (como el análisis de otros algoritmos).

En general el objetivo es hacer un uso óptimo de los recursos disponibles: el espacio (memoria) y el tiempo (de procesamiento). Como resultado del análisis lo más deseable es obtener el desempeño del algoritmo en el peor y mejor caso, en el caso medio y su varianza, todo en términos de la cantidad de información por procesar.

3 Notación asintótica

Si se conoce el tiempo (o espacio) que utiliza un algoritmo para procesar 50 datos (nombres, números, imágenes), 70 datos, 100, 200, 300 y 500 datos, se puede construir una gráfica que dé una idea de cómo se comportará el algoritmo cuando la cantidad de datos siga creciendo. De hecho, es en términos de *n* datos como nos interesa su comportamiento. Estamos hablando aquí de su *comportamiento asintótico*.

Es necesario establecer cierta notación que facilite el estudio de tal comportamiento:

$O(g(n)) = \{f:\mathbf{N}\rightarrow\mathbf{N}$ para las que existen constantes positivas c y n_0 tales que $0 \leq f(n) \leq cg(n)$ para toda $n>n_0\}$

Este es el conjunto de funciones que acotan por arriba.

$\Omega(g(n)) = \{ f:\mathbf{N}\rightarrow\mathbf{N}$ para las que existen constantes positivas c y n_0 tales que $0 \leq cg(n) \leq f(n)$ para toda $n>n_0\}$

Este es el conjunto de funciones que acotan por abajo.

Con la unión de estos conjuntos se obtiene:

$\Theta(g(n)) = \{ f:\mathbf{N}\rightarrow\mathbf{N}$ para las que existen constantes positivas c_1, c_2 y n_0 tales que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para toda $n>n_0\}$

Estos conjuntos poseen muchas propiedades agradables en el momento de analizar un algoritmo (ver [1], sección 2.1).

3.1 Notación

Por conveniencia, se utilizan de manera indistinta expresiones como $T(n) = aT(n/b) + \Theta(n)$ denotando con $\Theta(n)$ alguna función del conjunto y no el conjunto en sí. De esta manera, $O(1)$ denota una constante.

4 Análisis de algoritmos

El método empleado en el análisis de un algoritmo depende por completo de la naturaleza del algoritmo que se trate. Por otra parte, un algoritmo puede ser analizado de distintas formas y más aún si consideramos que muchos se pueden programar iterativa o recursivamente. En algunas ocasiones se pueden establecer esquemas generales de análisis como los que a continuación se presentan.

4.1 Correctitud

Un error común es hacer una prueba de escritorio con algunos valores y concluir que el algoritmo funciona. Este método empírico puede no encontrar errores oscuros que no sean fáciles de detectar en pruebas experimentales.

Si el algoritmo es **iterativo** se procede de la siguiente manera:

1. Escribir una especificación para la salida que debe producirse en términos de los datos de entrada.
2. Para cada ciclo (comenzando de los más internos), diseñar un invariante que se mantenga verdadero en cada iteración y que capture el progreso del ciclo.
3. Probar que el invariante se cumple. Esto generalmente se hace por inducción sobre el número de iteraciones. Se puede comenzar generando una lista de los valores de las variables a partir de sus valores anteriores (esto se puede usar como hipótesis de inducción).
4. Utilizar el invariante de cada ciclo para verificar la finitud.
5. Utilizar el invariante de cada ciclo y las condiciones de terminación para probar que el algoritmo obtiene el resultado esperado.

Si el algoritmo es **recursivo** se prueba por inducción sobre el tamaño del problema:

6. Identificar lo que será utilizado como tamaño del problema.
7. Probar la base de la inducción (generalmente sólo involucra la base de la llamada recursiva).
8. Probar que cada llamada recursiva es un problema del mismo tipo pero más pequeño (ésta es la finitud).
9. Probar la hipótesis de inducción, asumiendo que cada llamada recursiva trabaja de manera correcta para demostrar que la llamada actual trabaja correctamente.

4.2 Complejidad algorítmica

Las ideas detrás del análisis son relativamente simples aunque generalmente muy difíciles de aplicar (ver [4]). En el caso de algoritmos iterativos, se carga O(1) a cada segmento de código que no incluya ciclos y se usan las reglas de sumas y productos de la notación asintótica para incluir ciclos.

En el caso de los algoritmos recursivos, la clave está en determinar la relación de recurrencia que describe la complejidad y resolverla. La solución de recurrencias generalmente se obtiene por iteración, sustitución o con el Teorema Maestro (ver [1] pág. 62).

Aún algoritmos sencillos pueden requerir un análisis para el que se necesite un buen conocimiento de matemáticas (ver [6], secciones 3.7 y 3.8, [2]). Para muchos algoritmos es más sencillo realizar un *análisis amortizado* (ver [1], cap. 18), tomando ideas de los economistas. En el análisis amortizado se obtiene como resultado el costo por operación, calculando el total para n operaciones y luego sacando un promedio, $T(n)/n$.

El análisis amortizado posee las siguientes características:

- Puede utilizarse para mostrar que el costo medio de una operación es pequeño.
- No interviene la probabilidad, a diferencia del análisis del caso medio.
- Garantiza el desempeño de cada operación en el caso medio.
- Es relativamente sencillo de aplicar en algoritmos sofisticados como los árboles de Fibonacci (ver [1], cap. 21).

Hay tres métodos posibles:

Agregación, considerando el mismo peso por operación.

Conteo, asignando pesos distintos a las operaciones (su peso amortizado).
Potencial, asociando energía potencial a la estructura como un todo. Si D_k es el estado de la estructura después de la k-ésima operación y $\Phi(D_k)$ su energía potencial y c_k el costo de la operación k-ésima, entonces el costo amortizado a por operación se calcula como

$$a_i = c_i + \Phi(D_k) - \Phi(D_{k-1}).$$

Un esquema general para aplicar análisis amortizado incluye tener que contestar las preguntas siguientes.

- ¿Cuánto se compra a crédito?
- ¿Cuál es la regla de uso del crédito?
- ¿Cuál es el costo asignado por operación?

Y entonces se prosigue a

1. mostrar que hay crédito suficiente para pagar una operación, y
2. mostrar que, después de una operación, hay suficiente crédito para satisfacer la regla de uso.

4.3 Técnicas experimentales

Hay varios esquemas pero los más utilizados son el de caja negra y el de caja blanca. Son complementos indispensables de las pruebas formales pues las constantes introducidas por la codificación de los algoritmos pueden mermar seriamente su desempeño en la práctica.

Caja negra: Consiste en tomar una muestra suficientemente representativa del conjunto de las posibles entradas y verificar que la salida es la correcta sin ver la forma como el algoritmo está trabajando.

Caja blanca: Consiste en preparar un conjunto de entradas que aseguren que cada línea de código será ejecutada alguna vez y que las condiciones serán evaluadas en sus límites.

Es posible utilizar estos dos métodos de manera complementaria.

5 Conclusiones

Se han presentado las ideas generales para poder hacer el análisis de un algoritmo. Como hay una gran variedad de algoritmos, es de esperarse que estos métodos se deban aplicar con las adaptaciones necesarias. El curso pretende mostrar algunas de las técnicas más ampliamente utilizadas como inicio de un estudio más detallado sobre análisis de algoritmos.

6 Bibliografía

[1] Cormen, Introduction to Algorithms, MIT Press, 1991.
[2] Knuth, Concrete Mathematics: Fundations for Computer Science, Addison Wesley, segunda edición, 1994.
[3] Knuth, El arte de programar ordenadores Vol I: Algoritmos Fundamentales, Reverté, 1995.
[4] Knuth, Selected Papers on Analysis of Algorithms, CSLI Publications, 2000.
[5] Parberry, Problems on Algorithms, Prentice Hall, 1995.
[6] Sedgewick, Analysis of Algorithms, Addison Wesley, 1996.