Universidad Diego Portales

Apunte Clase 2: Algoritmos

Profesor Yerko Ortiz
2024

Tabla de contenidos

1 Introducción

> Computer science is no more about computers than astronomy is about telescopes.
>
> *Edsger Dijkstra*

Objetivos de la clase:

1. Comprender la metodología para estudiar algoritmos que se empleará durante el curso.
2. Revisar conceptos elementales aritmética y su relación con el pensamiento algorítmico.
3. Conceptos clave a revisar: algoritmo, problema computacional, entrada, salida, instancia, eficiencia, correctitud, estructura de datos, tipo de dato abstracto.

Los algoritmos son ideas abstractas que existen en nuestro día a día de manera silenciosa en muchas de nuestras actividades del diarias: escuchar música en algún dispositivo electrónico, búsquedas en algún buscador de internet, editar un archivo con un editor de texto, la ruta que toma el repartidor para que la pizza llegue caliente a su casa y hasta para jugar al Age of Empires.

Pero la noción de lo que un algoritmo no es para nada algo moderno sino que se remonta a mucho tiempo atrás, por ejemplo el algoritmo para encontrar el máximo común divisor entre dos números enteros es una idea que viene de la antigua Grecia de la mano de Euclides quien descubrió y documentó este método en su celebre libro Elementos.

Para el curso la pregunta primordial es cómo podemos representar esas ideas o aplicaciones en forma de problemas computacionales para luego resolverlas diseñando uno o varios algoritmos, pero el trabajo no queda ahí puesto que una vez tenemos un algoritmo que resuelve un problema debemos demostrar su correctitud y analizar su eficiencia.

2 Problema Computacional

Un problema computacional es un enunciado que denota la relación binaria entre un conjunto de datos de entrada y otro de salida. Dicho enunciado debe proveer una serie de propiedades para verificar que un determinado valor de salida está relacionado a una instancia de entrada, donde por nos referimos a un valor especifico de entrada perteneciente al conjunto definido en el problema. Otro punto importante el algoritmo debe ser capaz de resolver múltiples instancias de entradas de tamaño arbitrario.

2.1 Ejemplo: ordenar en una secuencia de enteros

Para comprender la idea de mejor manera se hará revisión de un problema bien conocido: ordenar una secuencia de enteros.

Si consideramos como enunciado la oración "ordenar una secuencia de enteros", a más de alguno se le ocurrirá almacenar la secuencia de enteros en un arreglo y luego aplicar el método sort de Java. Está bien pensar de esa forma para proponer una solución al problema, sobretodo admitiendo en que en un entorno de ingeniería dicha solución en muchos casos sería la más simple y efectiva, sin embargo antes de diseñar una solución hay comprender el problema y preguntarse una serie de sutilezas que no son explicitas en el mismo:

1. ¿Qué pasa si la maquina no tiene suficiente memoria RAM para ejecutar el algoritmo?
2. ¿Qué pasa si la maquina tiene más de un procesador?
3. ¿Cómo se puede validar si una secuencia de números está ordenada correctamente?

Este ejercicio de preguntas puede continuar de manera indeterminada; el punto que se busca ilustrar es que para comprender un problema computacional no basta con leer el enunciado, sino que tener claridad sobre la relación entrada y salida que el problema describe.

Durante el curso asumiremos que para la mayoría de los algoritmos a revisar:

1. La maquina tiene suficiente RAM para ejecutar un algoritmo de inicio a fin.
2. La maquina tiene uno y solo un procesador.
3. La maquina soporta a bajo nivel todos los tipos de datos y operaciones que Java especifica.

Ahora volviendo a las preguntas realizadas, ya podemos descartar dos preguntas, no nos preocuparemos de si la maquina tiene suficiente RAM, como tampoco de ejecutar los algoritmos de manera paralela con múltiples procesadores. Sin embargo, la pregunta de cómo es posible validar si una secuencia está ordenada es la clave para definir el problema.

Al final lo importante es comprender la relación entre la entrada y salida del problema:

INPUT: Una secuencia S de N enteros $< s_0, s_1, ..., s_{N-1} >$

OUTPUT: Una permutación S' $< s'_0, s'_1, ..., s'_{N-1} >$ de la secuencia S tal que $s'_0 \leq s_1 \leq ... \leq s'_{n-2} \leq s'_{n-1}$

En esta descripción se aprecia la relación binaria entre la entrada y salida de un algoritmo, para toda instancia entrada debe existir uno o más valores de salida. También es posible observar que el algoritmo realiza una permutación eso implica que en lugar de retornar una nueva secuencia, este puede modificar la secuencia.

Para todo problema computacional que estudie hoy y en el futuro siempre es buena idea especificarlo en término de entrada y salida para comprender bien el mismo problema.

3 Algoritmo

Un algoritmo es una secuencia de pasos finita y bien definida que es capaz de transformar toda instancia de entrada en su respectivo valor de salida para un determinado problema computacional.

De esta definición se puede concluir que un algoritmo resuelve un problema, pero también se mencionan una serie de características:

1. Un algoritmo es finito: para toda instancia de entrada un algoritmo de producir el valor de salida en un tiempo finito, es decir, el algoritmo debe tener término.
2. Un algoritmo es bien definido: todos los pasos del algoritmos son definidos de manera rigurosa y sin ambigüedades.
3. Un algoritmo tiene una entrada: todo algoritmo debe recibir cero o más instancias de entrada.
4. Un algoritmo produce una salida: todo algoritmo debe producir una o más instancias de salida.

3.1 Correctitud

Un algoritmo se dice que es correcto si y solo si para toda instancia de entrada produce el valor de salida esperado.

En el curso se hará estudio de principalmente algoritmos deterministas, los cuales en su mayoría pueden ser demostrados por inducción o algún otro método a revisar cuando sea necesario.

3.2 Eficiencia

La eficiencia es el plato de fondo a la hora de analizar algoritmos. La eficiencia viene de la mano de los recursos computacionales que un algoritmo dispondrá en su ejecución: CPU y memoria RAM. Nunca olvidar que un algoritmo es para resolver un problema, el algoritmo más eficiente no es necesariamente el que mejor resuelve un problema.

En un principio para ver qué tanta RAM y CPU gasta un algoritmo es posible realizar algún benchmark que mida el uso de estos recursos, no obstante eso está sujeto a la maquina que ejecuta el algoritmo, es por eso que para analizar algoritmos sin sesgos se hará uso de cotas asintóticas.

3.3 Estructuras de Datos

Una estructura de datos es una forma de almacenar, organizar y especificar operaciones a un conjunto de datos. El curso se llama algoritmos y estructuras de datos por una justa razón estos siempre van de la mano, no se puede estudiar el uno sin el otro.

Como estructuras de datos usted ya conoce un par, como el arreglo, cola o pila. Durante el transcurso del curso se hará estudio a profundidad de las estructuras mencionadas y varias otras.

3.4 Abstracción

Así como algoritmo y estructura de datos van siempre de la mano, también se hará revisión de la estrecha relación de estos con la abstracción a la hora de programar. Piense que abstraer implica definir o pensar algo a alto nivel para utilizarlo en ideas más elaboradas o complejas. Para las relaciones entre esas ideas o componentes de alto nivel utilizaremos interfaces (tipos de datos abstractos).

A modo de ilustración piense en el sistema operativo de su computador personal, definir un sistema operativo como un solo algoritmo con un conjunto de entrada y de salida suena a una tarea cercana a lo imposible, pero si abstraemos el problema y definimos un sistema operativo como muchos algoritmos que resuelven problemas específicos y bien definidos (procesos, permisos, administración de memoria, manejo de drivers, entre otros), luego no es tan imposible diseñar un sistema operativo, lo único que resta es conectar esos distintos algoritmos entre sí con el fin de ofrecer a los usuarios un conjunto de funcionalidades.

4 Sistema posicional

El sistema o notación posicional es un sistema numérico en el que el valor actual de un número viene dado por las posiciones de los dígitos que lo conforman.

Los números que usted usa a diario como 129 (ciento veintinueve) o 1000000 (un millón) pertenecen al sistema posicional base 10, conocido también como el sistema decimal. Estos números los usamos con justa razón puesto que trivializan ciertas operaciones aritméticas como suma, resta, multiplicación y división, por ejemplo el sistema numérico romano es un sistema no posicional donde las operaciones aritméticas son innecesariamente complejas, puesto que las posiciones de cada dígito no tienen un orden lógico a priori sencillo de manejar; para verificar esto de primera mano intente sumar MXIV(1014) con MCCXXXI(1231).

Si se está preguntando sobre la relación entre números y programación, la respuesta es que son parte de la misma cosa, la programación y las ciencias de computación tienen su origen en matemáticos que querían realizar cálculos más rápido (diseñar maquinas y algoritmos que realizan cálculos aritméticos). Esta necesidad dio origen a muchos inventos como la calculadora de Pascal en 1642, la maquina analítica diseñada por Ada Lovelace y Charles Babbage en 1837, hasta inventos modernos como la maquina de Turing(modelo de computación que utilizamos en el presente) y la arquitectura de Von Neumann(arquitectura de computador que utilizamos en el presente).

4.1 Sistema decimal

El sistema decimal hace uso de 10 dígitos que van desde el 0 hasta el 9, con sus respectivas posiciones que hacer mención a una unidad, decena, centena y así sucesivamente. El sistema decimal en el conjunto de los naturales (incluyendo el cero) se puede generalizar como:

$$\sum_{i \in \mathbb{N}} a_i 10^i \text{ para todo } a_i \text{ en el conjunto } \{0,1,2,3,4,5,6,7,8,9\}$$

Por ejemplo:

$$1245 = 10^3 \cdot 1 + 10^2 \cdot 2 + 10^1 \cdot 4 + 10^0 \cdot 5$$

4.2 Sistema binario

El sistema binario hace uso de 2 dígitos, 0 o 1, que con sus respectivas posiciones permiten representar el valor de un número. Haciendo el mismo ejercicio con el sistema decimal, el sistema binario se puede representar como:

$$\sum_{i \in \mathbb{N}} a_i 2^i \text{ para todo } a_i \text{ en el conjunto } \{0,1\}$$

Por ejemplo 11010:

$$11010 = 2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 0$$

4.2.1 Transformación de decimal a binario

Como caso de estudio considere el problema de transformar un número representado en sistema decimal a su homologo en sistema binario. Definamos el problema de forma apropiada para poder diseñar un algoritmo que lo solucione:
Input: Un número entero n
Output: Una secuencia de caracteres S $< s_0, s_1, ..., s_{n-1} >$ que representa el valor de n en sistema binario, tal que:

$$\sum_{i \in \mathbb{N}} s_i 2^i \text{ para todo } s_i \text{ en el conjunto } \{0, 1\}$$

Solución: para transformar un número representado en sistema decimal a binario, hay que hacer uso de la definición:

$$n = \sum_{i \in \mathbb{N}} s_i 2^i \text{ para todo } s_i \text{ en el conjunto } \{0, 1\}$$

En palabras sencillas, hay que calcular n como suma de potencias de 2. Por ejemplo consideremos n = 31:

$$31 = 16 + 8 + 4 + 2 + 1$$

$$31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

$$31 = 2^4 \cdot 1 + 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 1$$

$$\text{Por lo tanto } 31_{base10} = 11111_{base2}$$

Para diseñar un algoritmo que solucione el problema necesitamos saber cómo representar un número en sistema decimal en sumas de potencias de dos, para esto hay que encontrar la potencia de dos menor a n que es la más cercana:

1. Si n = 31, entonces la potencia de dos más cercana que no supera a n es 16, acto seguido hay que restarle a n la potencia encontrada. $31 - 16 = 15$ y repetir el proceso con el resultado de la resta hasta llegar a 0.
2. Si n = 15, entonces la potencia de dos más cercana es 8, 15 - 8 = 7.
3. Si n = 7, entonces la potencia de dos más cercana es 4, 7 - 4 = 3.
4. Si n = 3, entonces la potencia de dos más cercana es 2, 3 - 2 = 1.
5. Si n = 1, entonces la potencia de dos más cercana es 1, 1 - 1 = 0. Se termina el calculo.

Esto funciona para descomponer un número n cualquiera en potencias de dos, por ejemplo:

$$14 = 8 + 4 + 2$$

$$14 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 0$$

Por lo tanto $14_{base10} = 1110_{base2}$

Con esto ya tenemos un método algorítmico que podemos ejecutar a mano para obtener las potencias de dos que componen un número, para ahora debemos definir el algoritmo:

1. Crear una tabla T de tamaño k, que almacena una secuencia de potencias de dos: $T[0] = 2^0, T[1] = 2^1, T[k-1], 2^{k-1}$
2. Crear una lista L donde se guardarán los resultados.
3. Para todo n mayor a 0, encontrar en T, el valor T[i] que es más cercano a n, guardar en L el valor de i encontrado.
4. Asignar a n el valor de $n - T[i]$, repetir el paso anterior si n es distinto a 0.

Ahora tenemos una secuencia de instrucciones que más o menos nos dicen qué hacer para calcular a mano, pero esto todavía dista de resolver el problema original y segundo no es lo suficientemente riguroso como para considerarse un algoritmo, por ejemplo qué es una tabla, qué es una lista o cómo se hará búsqueda son preguntas que deben surgir cuando se lee esa secuencia de instrucciones, no obstante esa secuencia de instrucciones es suficiente para empezar a refinar y transformarla en un algoritmo que se pueda implementar en algún lenguaje de programación. Recuerde los algoritmos son ideas abstractas, la implementación en algún lenguaje de programación es el producto final.

Para dar solución al problema se iterará sobre la idea anterior para mejorarla. Para encontrar las potencias que componen a n utilizaremos otro método en lugar de tener una tabla. Para esto se utilizarán divisiones(división entera) por dos de manera iterativa, utilizando el módulo de cada n resultante como dígito: Si se hace uso del módulo de dos se puede observar lo siguiente:

$$31 \mod 2 = 1$$
$$31/2 = 15$$
$$15 \mod 2 = 1$$
$$15/2 = 7$$
$$7 \mod 2 = 1$$
$$7/2 = 3$$
$$3 \mod 2 = 1$$
$$3/2 = 1$$
$$1 \mod 2 = 1$$
$$1/2 = 0$$

$$31_{base10} = 11111_{base2}$$

Hagamos la prueba con n = 6:

$$\begin{aligned} 6 \mod 2 &= 0 \\ 6/2 &= 3 \\ 3 \mod 2 &= 1 \\ 3/2 &= 1 \\ 1 \mod 2 &= 1 \\ 1/2 &= 0 \end{aligned}$$

$$6_{base10} = 110_{base2}$$

Esta idea algorítmica puede ser implementada en Java:

```
public class IntToBin {
    public static String intToBin(int n) {
        String bin = "";
        while(n != 0) {
            bin = (n%2) + bin;
            n /= 2;
        }
        return bin;
    }

    public static void main(String[] args) {
        System.out.println(intToBin(25));
        System.out.println(intToBin(2505));
        System.out.println(intToBin(8));
        for(int i=0; i < 10000000; i++) {
            System.out.println(intToBin(i));
        }
    }
}
```

A modo de ejercicio:

1. Describa la secuencia de pasos que realiza la implementación en Java, tal como se hizo con la idea anterior(encontrar las potencias de dos).

2. Demuestre la correctitud del algoritmo para todo posible número natural.

4.3 Generalización a cualquier base

Para terminar y generalizar el sistema posicional, cada número puede ser representado con un sistema de base $k \in \mathbb{N}$:

$$\sum_{i \in \mathbb{N}} a_i k^i \text{ para todo } 0 \leq a_i < k \text{ y } k \geq 1$$

Esta definición se puede extender para números negativos (enteros) y reales (fraccionales e irracionales).

Referencias

[1] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. McGraw-Hill Higher Education, 2008. ISBN: 978-0-07-352340-8.

[2] R. Sedgewick and K. Wayne. *Algorithms, 4th Edition:* Addison-Wesley, 2011. ISBN: 9780321573513.

[3] S.S. Skiena and M.A. Revilla. *Programming challenges: The programming contest training manual*. Springer, 2003. ISBN: 978-0-387-00163-0.

[4] J.E. Thompson and M. Peters. *Arithmetic for the practical man*. Van Nostrand, 1962. ISBN: 978-0-442-28484-8.