Estructuras de Datos y Algoritmos: Clase 1

Yerko Ortiz

13 de marzo de 2024

Objetivo: La primera clase revisa los contenidos a estudiar durante el semestre, denotando las ideas centrales del curso: problema, algoritmo, eficiencia, correctitud y estructura de datos.

Índice

1. Introducción

El objetivo del curso es formar y fomentar el pensamiento algorítmico. Esto implica a priori lo siguiente:

- Resolver problemas computacionales mediante el diseño de algoritmos, además del uso e implementación de estructuras de datos.

- Argumentar la eficiencia de un algoritmo
- Probar la correctitud de un algoritmo

El detalle está en que el verdadero objetivo del curso es la abstracción de ideas algorítmicas y la capacidad de comunicar esas ideas. Si bien un algoritmo lo expresamos a una maquina mediante código, en la vida real también es importante expresar nuestro algoritmos e ideas a otras personas mediante palabras. Entonces el verdadero fin, es poder comunicar ideas y soluciones algorítmicas de manera clara y elocuente.

La presencia de algoritmos hoy en día es algo que puede pasar desapercibido para el día a día de los usuarios de estos mismos; bases de datos(estructuras de datos), sistemas de pago electrónico(algoritmos de cifrado), videojuegos(algoritmos gráficos), transmisión de señales (transformada de Fourier), blockchain (árbol de Merkle), transmisión de información(códigos correctores de errores), sistemas operativos (scheduling), logística (optimización). No obstante su tenue existencia sirve para recalcar una característica fundamental de estos, son ideas abstractas que se pueden expresar mediante código o palabras, existen en silencio en la mente de alguien o materializados como proceso computacional. La importancia de estos queda fuera de dudas, si los algoritmos de cifrado fueran fáciles de romper todos guardarían el dinero bajo el colchón, si no existiese la transformada de Fourier y los códigos correctores de errores, posiblemente no existiría internet. Es por esto que los algoritmos también pueden ser considerados tecnología al igual que un dispositivo de hardware.

2. Problema Computacional

Un problema computacional se puede describir como un enunciado que describe la relación entre un conjunto de datos de entrada y otro de salida. Un problema contiene propiedades y restricciones que permiten verificar si el output para una instancia de entrada es correcta. Así mismo es posible afirmar la existencia de problemas para los que una misma instancia de entrada es posible tener más de una salida correcta; es el enunciado del problema el que describe y denota todos estos puntos:

- Relación binaria entre conjunto entrada y de salida (input sería el dominio y output el codominio).
- Propiedades para verificar la correctitud de la salida respecto una instancia de entrada.
- La entrada es un conjunto de cardinalidad (tamaño) arbitraria.

Ejemplos de problemas computacionales:

- Encontrar el máximo en un arreglo

- Ordenar un arreglo
- Encontrar la ocurrencia de una palabra en un texto
- Multiplicar matrices
- Comprimir un archivo wav en formato mp3

3. Algoritmo

Un algoritmo es un conjunto de instrucciones finitas y bien definidas que transforman cada instancia de entrada de un problema en un valor de salida especifico(determinista); un algoritmo determinista es aquel que para el mismo valor de entrada siempre retorna el mismo valor de salida, independientemente si el problema admite múltiples soluciones. Se dice que un algoritmo es correcto si y solo si para cada instancia de entrada, retorna el valor de salida esperado por el problema en un tiempo de ejecución finito, en este aspecto el valor de salida se puede validar con las propiedades y restricciones que el enunciado del problema establece.

4. Correctitud

Un algoritmo es correcto cuando entrega el valor de salida esperado para cada instancia de entrada. Para probar la correctitud es posible utilizar inducción identificando alguna condición invariante dentro del algoritmo que se propone.

1. Identificar la invariante y proponer una hipótesis en base a esta.
2. Validar un caso base.
3. Asumir que la hipótesis es cierta para un conjunto de entrada de tamaño arbitrario.
4. Probar que la hipótesis se cumple para un conjunto de entrada de mayor tamaño, agregando uno o más elemento al conjunto de entrada propuesto en el paso anterior.

5. Eficiencia

La eficiencia de un algoritmo describe el ratio de crecimiento entre los recursos computacionales y el tamaño de entrada de un problema. Teniendo la noción de qué tan eficiente es un algoritmo es posible comparar con otros algoritmos, para así escoger el que haga más sentido para resolver el problema(a veces el algoritmo menos eficiente, puede ser mejor solución que otro más eficiente). Si

bien es posible medir la eficiencia de un algoritmo realizando benchmarks(midiendo el tiempo que demora o asignaciones en memoria), esto tiene una gran desventaja porque depende completamente de la maquina en la cual es ejecutado. Puesto que los algoritmos son ideas abstractas que habitan en la mente, es importante tener una herramienta que permita compararlos de manera objetiva y sin dependencias externas, es por esto que se hace uso de notación asintótica para describir el tiempo de ejecución o espacio de memoria que un algoritmo requiere.

A priori la notación asintótica puede describirse como contar la cantidad de operaciones que un algoritmo realiza, luego definir ese calculo como una función en términos de la entrada y para finalizar ajustar esa función utilizando una cota asintótica:

- Cota superior, Big oh $\mathcal{O}(f(n))$ para el peor caso
- Cota inferior: Omega $\Omega(f(n))$ para el mejor caso
- Cota ajustada: Theta $\Theta(f(n))$ para el caso promedio

6. Estructura de datos

Una estructura de datos es una forma de almacenar y consultar datos. Las estructuras de datos suelen implementar un conjunto de operaciones para acceder, almacenar y consultar datos de forma eficiente.
Ejemplos:

- Arreglo estático
- Arreglo dinámico
- Lista Enlazada
- Árbol de búsqueda binario
- Árbol rojo negro
- Tabla de Hash
- Heap
- Record
- Lista de adyacencia

- Trie
- Bit Vector
- Union Find Disjoint Set

7. Tipo de dato abstracto

Otro concepto importante es el de tipo de dato abstracto, un tipo de datos abstracto es un tipo de datos que es definido por su comportamiento(operaciones), en lugar de sus atributos. Esto quiere decir que un tipo de datos abstracto es agnóstico a su implementación siempre y cuando esta contenga todas las operaciones que definen al tipo de dato. Los tipos de datos abstracto suelen ser implementados con estructuras de datos conocidas o diseñadas a priori.
Ejemplos:

- List, a menudo implementado usando listas enlazadas o arreglos.
- Stack, a menudo implementado usando listas enlazadas o arreglos.
- Queue, a menudo implementado usando listas enlazadas o arreglos.
- Priority Queue, a menudo implementado usando heap.
- Set, a menudo implementando algún árbol balanceado(rojo negro, AVL, B-tree) o tablas de hash.
- Map, a menudo implementando algún árbol balanceado(rojo negro, AVL, B-tree) o tablas de hash.