

Algoritmos y Estructuras de Datos III

Tomás Agustín Hernández



Complejidad Computacional (repaso)

Problema

Descripción de los datos de entrada y la respuesta a proporcionar para cada uno de los datos de entrada.

Instancia de un Problema

Es un juego válido de datos de entrada.

Máquina RAM

Supongamos una Máquina RAM.

- La memoria está dada por una sucesión de celdas numeradas. Cada **celda** puede almacenar un valor de **b bits**.
- Supondremos habitualmente que esos **b bits** de cada celda están fijos, y suponemos que todos los datos que maneja el algoritmo se pueden almacenar con **b bits**. **Ej.:** Lo que quiere decir esto es que suponemos que todas las celdas son de 8 bits, y los datos que maneja el algoritmo también son de 8 bits.
- Se tiene un programa imperativo que NO está almacenado en memoria que está compuesto por asignaciones y las estructuras de control habituales.
- Las asignaciones acceden a las celdas de memoria y realizan las operaciones estándar sobre los tipos de datos primitivos habituales.

Cada una de las instrucciones que se ejecuten tienen un tiempo de ejecución asociado

- El acceso a cualquier celda de memoria, tanto lectura como escritura es $O(1)$.
- Las asignaciones y el manejo de las estructuras de control se realiza en $O(1)$.
- Las operaciones entre valores lógicos son $O(1)$.

Las operaciones entre enteros/reales dependen de b

- Las sumas y restas son $O(b)$.
- Las multiplicaciones y divisiones son $O(b \log b)$

Nota: Si b está fijo, entonces las operaciones entre enteros/reales es $O(1)$.

Tiempo de Ejecución de un Algoritmo

Sea A un algoritmo, su tiempo de ejecución es: $T_A(I)$ donde esto indica que es la suma de los tiempos de ejecución del algoritmo en una instancia dada I.

$|I|$: Cantidad de bits necesarios para almacenar los datos de entrada de I.

Nota: Si **b está fijo** y la entrada ocupa n celdas de memoria entonces $|I| = bn = O(n)$

Complejidad de un Algoritmo

Sea A un algoritmo, su complejidad es: $f_A(n) = \max_{I: |I|=n} T_A(I)$ donde esto indica que la complejidad de un algoritmo A dado un n cualquiera, es el que de mayor tiempo de ejecución tiene en una instancia dada I.

Cotas

Cota Superior (O): $f(n) \in O(g(n)) \iff \exists c \in \mathbb{R} > 0, n_0 \in \mathbb{N} \text{ tal que } \forall n \geq n_0 : f(n) \leq c * g(n)$

Cota Inferior (Ω): $f(n) \in \Omega(g(n)) \iff \exists c \in \mathbb{R} > 0, n_0 \in \mathbb{N} \text{ tal que } \forall n \geq n_0 : f(n) \geq c * g(n)$

Cota Ajustada (θ): $f(n) \in \theta(g(n)) \iff f(n) \in O(g(n)) \text{ y } f(n) \in \Omega(g(n))$. Es decir, $\theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Tipos de Funciones

- $O(n)$: lineal
- $O(n^2)$: cuadrático
- $O(n^3)$: cúbico
- $O(n^k)$ $k \in \mathbb{N}$: polinomial. Ej.: $O(n^4)$, $O(n^5)$
- $O(\log n)$: logarítmico.
- $O(d^n)$ $d \in \mathbb{R}_{>1}$: exponencial. Ej.: $O(2^n)$, $O(4^n)$

Algoritmos Satisfactorios y No Satisfactorios

Un Algoritmo Satisfactorio es un algoritmo que tiene un costo menor a otro.
Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor).
Los algoritmos supra-polinomiales se consideran no satisfactorios.

Problema de Optimización

Sea $x \in S$, un problema de optimización consiste en encontrar la mejor solución dentro de un conjunto:

- $z^* = \max f(x)$
- $z^* = \min f(x)$

Función Objetivo: Es una función de la forma $f : S \Rightarrow \mathbb{R}$

- El conjunto S es la **región factible**.
- Los elementos $x \in S$ se llaman **soluciones factibles**.
- El valor $z^x \in \mathbb{R}$ es el **valor óptimo** del problema, y cualquier solución factible $x^* \in S / f(x^*) = z^x$ se llama un **óptimo** del problema

Algoritmos de Fuerza Bruta

También llamado búsqueda exhaustiva o generate and test. Genera todas las soluciones factibles y se queda con la mejor.
Suele ser fácil de implementar y es un algoritmo exacto: si hay solución, siempre la encuentra.
Lo malo es su complejidad (suele ser exponencial)

Backtracking

Es una técnica que consiste en una exploración ordenada del espacio de soluciones por medio de la extensión de soluciones parciales.
Cada posible extensión de la solución parcial se explora haciendo recursión sobre la nueva solución extendida, esto se puede ver como un árbol y que podemos podarlo (descartar configuraciones antes de explorarlas) al árbol para reducir el espacio de búsqueda.

Todas las soluciones:

```
1 | algoritmo BI(a,k)
2 |   si a es solución entonces
3 |     procesar(a)
4 |     retornar
5 |   sino
6 |     para cada a' en Sucesores(a,k)
7 | ----- BI(a', k + 1)
8 |     fin para
9 |   fin si
10 |  retornar
```

Una solución:

```
1 | algoritmo BI(a,k)
2 |   si a es solución entonces
3 |     sol <- a
4 |     encontro <- true
5 |   sino
6 |     para cada a' en Sucesores(a,k)
7 | ----- BI(a', k + 1)
8 |       si encontro entonces
9 |         retornar
10 |      fin si
11 |    fin para
12 |   fin si
13 |  retornar
```

Ej.: Resolver un sudoku se resuelve en forma muy eficiente con un algoritmo de backtracking.

Fuerza Bruta

Un algoritmo de fuerza bruta (también llamado búsqueda exhaustiva) analiza todas las posibles configuraciones.

Ej.: Imaginemos que tenemos un tablero de ajedrez y tenemos que buscar las soluciones en las cuales ninguna dama amenace a otra. Una solución por fuerza bruta sería buscar todas las soluciones que existen, y de ahí agarrar las que me sirvan.