# Introducción

En las entradas anteriores vimos cómo ordenar de manera eficiente. Si bien este es un problema muy interesante por sí mismo, otra de las grandes ventajas es que también es un sub-problema de muchos otros. Ahora veremos varios problemas algorítmicos en donde usaremos nuestros algoritmos de ordenar como sub-rutina.

# Elementos repetidos

En algunas ocasiones tenemos listas de elementos y es importante saber si hay repeticiones o no. En otras ocasiones es importante saber cuántas veces se repite un elemento. O cuántas veces se repite el que más se repite. Hay muchas variantes de este problema. Enfoquémonos de momento en la primera.

**Problema.** Se tiene una lista $L=[a_1,a_2,\ldots,a_n]$ y se quiere saber si hay algún elemento que se repita.

Una posible forma de resolver el problema es realizar las comparaciones entre todas las parejas posibles de elementos. Esto se puede hacer mediante el siguiente pseudocódigo:

```
para j de 1 a n:
  para i de 1 a n:
    si L[i]=L[j]:
      decir que hay dos elementos iguales
si nunca pasó, decir que todos son diferentes
```

Este es un algoritmo correcto pues verifica todas las posibles parejas de elementos. Toma tiempo $O(n^2)$. Podríamos quedarnos con la impresión de que esto es lo mejor posible: finalmente, necesitamos la garantía de que las $\binom{n}{2}=\Theta(n^2)$ parejas de números que nos dan sean distintas entre sí. Sin embargo, este argumento es incorrecto. Es posible que con menos operaciones podamos descartar la igualdad de más parejas. A continuación veremos cómo.

Lo primero que haremos es ordenar la lista $L$. Después, compararemos a cada elemento con el elemento que le sigue. Esto es lo que se hace en el siguiente pseudocódigo, en donde `ordenar` es una subrutina que ordene a $L$ en tiempo óptimo.

```
L=ordenar(L) # tiempo O(n logn)
para i desde 1 hasta n-1:
  si L[i]=L[i+1]:
      decir que hay elementos repetidos
si nunca paso, decir que todos son diferentes
```

El algoritmo es correcto pues tras ordenar, los elementos iguales (si es que existen) quedan juntos y por lo tanto los detectaríamos en alguna de las comparaciones de elementos adyacentes.

En términos de tiempo de ejecución, estamos mucho mejor. El haber pre-ordenado la lista nos permite ahorrarnos el ciclo dentro del ciclo. El análisis de tiempo queda como sigue. La subrutina `ordenar` toma tiempo $O(n\log n)$. El ciclo tiene $O(n)$ iteraciones, cada una con una cantidad constante de pasos. Así, en total tenemos tiempo $O(n\log n)+O(n)=O(n\log n)$, lo cual es más rápido que el tiempo $O(n^2)$ de arriba. $\square$

# Suma de 2 elementos: cuadrático vs. ordenar vs. tabla hash

Anteriormente habíamos visto el siguiente problema:

**Problema.** Problema de la 2 suma.

**Entrada.** Una lista $L=[a_1,a_2,\ldots,a_n]$ y un número $M$.

**Salida.** Determinar si existen dos números $a_i$ y $a_j$ (quizás el mismo) en $L$ tales que $a_i+a_j=M$.

Anteriormente habíamos visto una manera sencilla de resolver el problema: tomamos todas las posibles parejas y verificamos si suman $M$ o no. Vimos que esto es correcto (pues pasa por todas las posibilidades) y que toma tiempo $O(n^2)$.

Podemos aprovechar de nuevo a `ordenar` como subrutina para dar un algoritmo más eficiente. Procedemos como sigue:

- Como primer paso, vamos a ordenar la lista $L$ (tiempo $O(n\log n)$).
- Para cada valor de $i$ de $1$ a $n$ vamos a preguntarnos si $M-a_i$ está en la lista $L$. Para ello podemos usar búsqueda binaria pues $L$ ya está ordenada. Necesitamos $n$ búsquedas binarias, cada una de tiempo $O(\log n)$. En total, este paso toma tiempo $O(n\log n)$.
- Si sí, esto quiere decir que $M-a_i=a_j$ para algunos $i$ y $j$. Esto es exactamente lo mismo a decir que $a_i+a_j=M$.
- Si nunca pasa esto, no hay dos elementos que sumen $M$.

De esta manera, esto nos da una forma de resolver el problema con tiempo $O(n\log n)$.

Una tercer forma de resolver el problema es mediante el uso de tablas hash. Simplemente vamos leyendo los elementos en $L$ uno por uno y los vamos insertando en una tabla hash, con un indicador de que son «valores de la lista». Luego insertamos los elementos del estilo $M-a_i$, con un indicador de que son «valores modificados». Cada inserción toma tiempo $O(1)$ en promedio. Si en algún momento de la segunda etapa intentamos insertar en la tabla un elemento que ya habíamos puesto como «valor de la lista», esto nos da dos elementos con suma $M$. Así, en total tenemos $O(n)$ inserciones de tiempo promedio $O(1)$, así que en promedio este método tardará tiempo $O(n)$.

En tiempo promedio, usar tablas hash es mejor que nuestro método de usar `ordenar` como subrutina. Sin embargo, usando tablas hash corremos el riesgo (pequeño, pero existente) de que haya colisiones que hagan que las inserciones tomen tiempo $O(n)$, de modo que en el peor de los casos nos podría llevar a un tiempo de ejecución $O(n^2)$.

## Elementos cercanos

Otra muy buena pregunta por sí misma y que sirve como subrutina para otros problemas algorítmicos es, dada una lista de números, detectar cuáles dos de ellos son los más cercanos entre sí.

Por ejemplo, si tenemos la lista

```
L=[45,12,56,63,32,8,76,42,75,34,23],
```

¿cuáles son los dos elementos de $L$ que están más cercanos entre sí?

Una forma sencilla de encontrar a la pareja más cercana es tomar todas las $\binom{n}{2}=\Theta(n^2)$ diferencias $|a_i-a_j|$ y de ahí elegir la menor. Esto se puede hacer haciendo un bucle en $i$ y en $j$, lo cual tomaría tiempo cuadrático.

Sin embargo, una mejor forma de hacerlo es ordenando. Si primero ordenamos la lista $L$, basta con revisar las diferencias del estilo $a_{i+1}-a_{i}$ para $i=1,...,n-1$, y la que sea menor corresponderá a la pareja más cercana. El cuello de botella de este algoritmo es ordenar, pues toma tiempo $O(n \log n)$, después de lo cual las comparaciones ya sólo toman tiempo lineal.

## Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. En el texto se discute cómo encontrar los elementos más cercanos de una lista de números. Y, ¿los elementos más lejanos entre sí

1. En el texto se discute cómo encontrar los elementos más cercanos de una lista de números. Y, ¿los elementos más lejanos entre sí? qué tan rápido se pueden encontrar?
2. Diseña un algoritmo que, dada una lista de números, determine cuál de ellos es el que se repite más veces. Tu algoritmo debe correr en tiempo $O(n\log n)$.
3. Diseña un algoritmo que use tablas hash para determinar si en una lista hay elementos repetidos o no. Discute las ventajas y desventajas de este algoritmo frente a usar una subrutina de ordenamiento.
4. Se tienen dos listas $L_1$ y $L_2$ cada una con $n$ números. Queremos determinar si $L_1$ se puede obtener a partir de $L_2$ simplemente reordenando sus elementos. Da un algoritmo que haga esto en tiempo $O(n \log n)$.
5. Hace algún tiempo implementaste un algoritmo correcto para resolver el problema de las películas. Repasa tu código y observa si puedes utilizar una subrutina de ordenamiento para obtener una implementación más rápida.

‹ Anterior
Ordenar eficientemente

Siguiente
Heurísticas para la creación de algoritmos ›

Por Leonardo Ignacio Martínez Sandoval