OIFem 2021: cuarta semana

Índice

Algoritmos de ordenación

Dada una lista de elementos de un conjunto ordenado, es decir, que si **a** y **b** son del conjunto entonces **a** ≤ **b** o **a** ≥ **b**, podemos ordenar los elementos de manera que si **a** aparece antes que **b** en la lista entonces **a** ≤ **b**.

Para conseguir ordenar los elementos existen distintos algoritmos, cada uno de ellos tiene unas características a tener en cuenta:

- La complejidad en el mejor de los casos
- La complejidad en el peor de los casos
- La complejidad media
- La memoria extra usada
- La dificultad de implementarlo

Con estas características y en función de nustro objetivo tenemos diferentes opciones. Podéis encontrar una lista de los algoritmos de ordenación más usados en Wikipedia. Aquí vamos a explicar solo unos cuantos.

Ordenación por inserción (insertion sort)

Dada una lista de *n* elementos, la ordenación por inserción funciona de la siguiente manera:

1. Suponemos que los *k* elementos que originalmente estaban al principio de la lista ya están ordenados en su orden relativo.

2. Miramos el elemento que está en la posición *k+1*, es decir el primero que no está ordenado, lo llamaremos *x*.
3. Comparamos *x* con el elemento que aparece justo a la izquierda en la lista actual, llamaremos a este elemento *y*. Si $y > x$, intercambiamos sus posiciones en la lista. Repetimos el paso (3) hasta que *x* sea el primer elemento de la lista o encontremos un *y* con $y \leq x$.
4. Ahora ya tenemos los *k+1* elementos que originalmente estaban al principio de la lista ordenados y podemos volver al paso (1).

Inicialmente empezamos con *k=1* ya que una lista de un solo elemento siempre está ordenada, repetimos hasta que *k=n*.

Podemos ver que este algoritmo tiene como en el mejor de los casos una complejidad O(n) y en el peor de los casos $O(n^2)$. En general, la complejidad media es $O(n^2)$.

Ordenamiento por selección (selection sort)

Dada una lista de *n* elementos, la ordenación por inserción funciona de la siguiente manera:

1. Empezamos con una lista desordenada de *n* elementos
2. Suponemos que tenemos los *k* elementos menores en su posición final.
3. Buscamos el elemento más pequeño entre los *n-k* restantes.
4. Colocamos el elemento encontrado en la posición *k+1* que es su posición final.
5. Si no hemos acabado volvemos a (2).

Podemos ver que este algoritmo tiene como en el mejor de los casos una complejidad $O(n^2)$ y en el peor de los casos $O(n^2)$. En general, la complejidad media es $O(n^2)$.

Ordenamiento “burbuja” (bubble sort)

1. Empezamos con una lista desordenada de *n* elementos
2. Vamos iterando sobre todos los elementos de la lista en orden, si en algún momento encontramos dos valores consecutivos que no están ordenados los intercambiamos.
3. Si al final de la iterción no hemos hecho ningún cambio es que la lista ya estava ordenada, si no volvemos a (2).

Podemos ver que este algoritmo tiene como en el mejor de los casos una complejidad O(n) y en el peor de los casos $O(n^2)$. En general, la complejidad media es $O(n^2)$.

Ordenamiento por mezcla (merge sort)

Este algoritmo funciona de menra recursiva. Dada una lista de *n* elementos, la divide en dos listas de *n/2* elementos cada una. Las ordena por separado y luego las mezcla.

1. Empezamos con una lista desordenada de *n* elementos
2. Dividimos la lista por la mitad en dos listas L_1 y L_2
3. Recursivamente ordenamos L_1 y L_2
4. Mezclamos el resultado para obtener la lista original ordenada

La parte más importante de este algoritmo es la mezcla, se van metiendo los elementos de V1 y V2 de nuevo dentro de V en el orden correcto. Al estar V1 y V2 ordenados solo tenemos que comparar el primer elemento sin colocar de cada lista.

1. Empezamos con dos listas L1 y L2 con n_1 y n_2 elementos
2. Vamos a ir llenando la lista L, inicialmente vacia
3. Los elementos x1 y x2 son los primeros elementos de L1 y L2
4. Comparamos x1 con x2, si x1 < x2 metemos x1 como siguiente elemento de L y x1 pasa a ser el siguiente elemento de L1
5. Una vez hemos usado toda una lista metemos lo que nos queda de la otra al final de L

La complejidad en todos los casos de este algoritmo es O(n*log(*n*)*)

Ordenamiento rápido (quick sort)

Este algoritmo también funciona de forma recursiva, en este caso elejimos un elemento x y dividimos la lista en dos listas también. Una con los elementos menores de x y una con los mayores que x.

1. Empezamos con una lista desordenada de *n* elementos

2. Elegimos un elemento de la lista aleatorio *x*
3. Separamos la lista en tres partes, la lista L1 con los elementos (< x), el elemento x, y la lista L2 con los elementos (> x).
4. Ordenamos recursivamente las listas L1 y L2.

La complejidad en tel mejor de los casos y en media es O(n*log(*n*)*), en el peor de los casos puede llegar a ser $O(n^2)$.

Ordenar en C++

Para ordenar con C++ existe ya una función optimizada que nos permite ordenar sin tener que implementar los algoritmos anteriores. Podéis encontrar información sobre su funcionamiento aquí.

Búsqueda binaria

Para poder encontrar un número en un vector ordenado no hace falta que miremos todas las posiciones, existe un algoritmo mejor. Este algoritmo se llama búsqueda binaria y aprovecha el hecho de que el vector está ordenado para lo siguiente:

Mantenemos el intervalo de posiciones factibles en las que puede estar el entero que buscamos.

Miramos el elemento que está en la mitad del intervalo, si es mayor que el número qu buscamos podemos descartar la mitad superior de nuestro intervalo, si es menor la mitad inferior, si es el número ya hemos terminado.

Vamos dividiendo entre 2 la medida del intervalo de posiciones factibles hasta que solo quede encontremos el número o no quede ninguna, en el segundo caso indica que el número no está en nuestro vector.

Podéis encontrar más información aquí.