Alicia Magaly Azúa Saucedo

1743217

Reporte de algoritmo de ordenación

31 agosto de 2017

En este documento hablaremos de 4 de los tipos de algoritmos de ordenamiento como los son bubble, insertion, selection, y Quicksort y veremos cómo funcionan y sus seudocódigos entre otras cosas.

Bubble:

es un algoritmo de clasificación simple que repite varias veces la lista para ordenar, compara cada par de elementos adyacentes y los intercambia si están en el orden incorrecto. El paso a través de la lista se repite hasta que no se necesitan swaps, lo que indica que la lista está ordenada. El algoritmo, que es un tipo de comparación, tiene el nombre de la forma en que los elementos más pequeños o más grandes "burbuja" a la parte superior de la lista. Aunque el algoritmo es simple, es demasiado lento e impráctico para la mayoría de los problemas

El tipo de burbuja tiene el peor de los casos y la complejidad media tanto O (n ^2), donde n es el número de elementos que se están ordenando.

Ejemplo:

Tomemos la matriz de números "5 1 4 2 8", y ordenamos la matriz del número más bajo al número más grande usando el tipo de burbuja. En cada paso, los elementos escritos en **negrita** están siendo comparados. Se necesitarán tres pasos.

Primer paso

(5 **1** 4 2 8) (**1 5** 4 2 8), Aquí, el algoritmo compara los dos primeros elementos, y los intercambios desde 5> 1.

(1 **5 4** 2 8) (1 **4 5** 2 8), Swap desde 5> 4

(1 4 **5 2** 8) (1 4 **2 5** 8), Swap desde 5> 2

(1 4 2 **5 8**) (1 4 2 **5 8**), Ahora, puesto que estos elementos ya están en orden (8> 5), el algoritmo no los intercambia.

Segundo paso

(**1 4** 2 5 8) (**1 4** 2 5 8)

(1 **4 2** 5 8) (1 **2 4** 5 8), Swap desde 4> 2

(1 2 **4 5** 8) (1 2 **4 5** 8)

(1 2 4 **5 8**) (1 2 4 **5 8**)

Ahora, la matriz ya está ordenada, pero el algoritmo no sabe si se ha completado. El algoritmo necesita un pase **entero** sin **ningún** intercambio para saber que está ordenado.

Tercer paso

(**1 2** 4 5 8) (**1 2** 4 5 8)

(1 **2 4** 5 8) (1 **2 4** 5 8)

(1 2 **4 5** 8) (1 2 **4 5** 8)

(1 2 4 **5 8**) (1 2 4 **5 8**)

Seudocódigo:

```
Procedure bubbleSort (A: lista de elementos clasificables)
    N = longitud (A)
    repetir
        Newn = 0
        Para i = 1 a n-1 inclusive
            Si A [i-1]> A [i] entonces
                Swap (A [i - 1], A [i])
                Newn = i
            terminará si
        Final para
        N = nuevo
    Hasta n = 0
Procedimiento final
```

Inserción:

El tipo de inserción itera, consumiendo un elemento de entrada cada repetición y creando una lista de resultados ordenada. En cada iteración, el tipo de inserción elimina un elemento de los datos de entrada, encuentra la ubicación que pertenece dentro de la lista ordenada e inserta allí. Se repite hasta que no queden elementos de entrada.

La clasificación se realiza normalmente en el lugar, iterando la matriz, creciendo la lista ordenada detrás de ella. En cada posición de la matriz, comprueba el valor allí contra el valor más grande en la lista ordenada (que pasa a estar al lado de ella, en la posición de matriz anterior marcada). Si es más grande, deja el elemento en su lugar y se mueve a la siguiente. Si es menor, encuentra la posición correcta dentro de la lista ordenada, desplaza todos los valores mayores hasta hacer un espacio e inserta en esa posición correcta.

La matriz resultante después de *k* iteraciones tiene la propiedad donde se ordenan las primeras entradas *k* + 1 ("+1" porque se omite la primera entrada). En cada iteración se retira la primera entrada restante de la entrada y se inserta en el resultado en la posición correcta

La entrada de peor caso más simple es una matriz ordenada en orden inverso. El conjunto de todas las entradas de peor caso consiste en todas las matrices donde cada elemento es el más pequeño o el segundo más pequeño de los elementos antes de él. En estos casos, cada iteración del bucle interno explorará y desplazará toda la subsección clasificada de la matriz antes de insertar el siguiente elemento. Esto le da a la inserción un orden cuadrático del tiempo de ejecución (es decir, O (n^ 2)).

Ejemplo: La siguiente tabla muestra los pasos para ordenar la secuencia {3, 7, 4, 9, 5, 2, 6, 1}. En cada paso, se subraya la clave en cuestión. La clave que se movió (o se dejó en su lugar porque era la más grande aún considerada) en el paso anterior se muestra en negrita.

3 7 4 9 5 2 6 1

3 7 4 9 5 2 6 1

3 **7** 4 9 5 2 6 1

3 **4** 7 9 5 2 6 1

3 4 7 **9** 5 2 6 1

3 4 **5** 7 9 2 6 1

2 3 4 5 7 9 6 1

2 3 4 5 **6** 7 9 1

1 2 3 4 5 6 7 9

Pseudocode:

Function insertionSortR (matriz A, int n)

si n> 0

InserciónSortR (A, n-1)

X ← A [n]

J ← n-1

Mientras que j> = 0 **y** A [j]> x

A [j + 1] ← A [j]

J ← j-1

Terminar mientras

A [j + 1] ← x

Fin si la

función final

Selection:

Es un algoritmo de clasificación, específicamente un tipo de comparación en el lugar. Tiene complejidad de tiempo O (n^ 2), haciéndola ineficaz en listas grandes, y generalmente realiza peor que el tipo de inserción similar. El tipo de selección se destaca por su simplicidad y tiene ventajas de rendimiento sobre algoritmos más complicados en ciertas situaciones, particularmente cuando la memoria auxiliar es limitada.

El algoritmo divide la lista de entrada en dos partes: la sublista de ítems ya ordenados, que se construye de izquierda a derecha en la parte delantera (izquierda) de la lista y la sublista de ítems que quedan por ordenar que ocupan el resto de la lista lista. Inicialmente, la sublista ordenada está vacía y la subred no ordenada es toda la lista de entradas. El algoritmo procede encontrando el elemento más pequeño (o mayor, dependiendo del orden de clasificación) en la sublista sin clasificar, intercambiándola (intercambiándola) con el elemento no clasificado más a la izquierda (poniéndolo en orden ordenado) y moviendo los límites de sublista un elemento a la derecha.

Pseudocode:

```
int minimo=0, i, j;

  int swap;

  for (i=0; i<n-1; i++)

  {

     minimo=i;

     for (j=i+1; j<n; j++)

        if (x[minimo] > x[j])

           minimo=j;

     swap=x[minimo];

     x[minimo]=x[i];

     x[i]=swap;

  }}
```

Quicksort:

Es un algoritmo de clasificación eficiente, que sirve como un método sistemático para colocar los elementos de una matriz en orden. Desarrollado por Tony Hoare en 1959 [1] y publicado en 1961, [2] sigue siendo un algoritmo comúnmente utilizado para la clasificación. Cuando se implementa bien, puede ser alrededor de dos o tres veces más rápido que sus principales competidores.

Quicksort es un tipo de comparación, lo que significa que puede ordenar elementos de cualquier tipo para los que se define una relación "menos que" (formalmente, un orden total). En implementaciones eficientes no es un tipo estable, lo que significa que el orden relativo de los elementos de igualdad no se conserva. Quicksort puede funcionar in situ en una matriz, requiriendo pequeñas cantidades adicionales de memoria para realizar la clasificación.

El análisis matemático de Quicksort muestra que, en promedio, el algoritmo toma O (n log n) comparaciones para ordenar n elementos. En el peor de los casos, hace comparaciones O (n 2), aunque este comportamiento es raro.

Quicksort es un algoritmo de división y conquista. Quicksort primero divide un arreglo grande en dos sub-arrays más pequeños: los elementos bajos y los elementos altos. Quicksort puede ordenar de forma recursiva los sub-arrays.

Los pasos son:

Elija un elemento, denominado pivote, de la matriz.

Partición: reordenar la matriz de modo que todos los elementos con valores menores que el pivote vienen antes del pivote, mientras que todos los elementos con valores mayores que el pivote vienen después de él (valores iguales pueden ir de cualquier manera). Después de esta división, el pivote está en su posición final. Esto se llama operación de partición.

Aplicar recursivamente los pasos anteriores a la sub-matriz de elementos con valores más pequeños y por separado a la sub-matriz de elementos con valores mayores.

El caso base de la recursión son matrices de tamaño cero o uno, que nunca necesitan ser ordenadas.

La selección del pivote y los pasos de partición se pueden hacer de varias maneras diferentes; La elección de los planes de implementación específicos afecta en gran medida el rendimiento del algoritmo.

Pseudocode:

```
Algoritmo Quicksort (A, lo, hi) es
   si lo <hi entonces
      P: = partición (A, lo, hi)
      Quicksort (A, lo, p - 1)
      Quicksort (A, p + 1, hi)

Algoritmo de partición (A, lo, hi) es
   Pivote: = A [hi]
   I: = lo - 1
   Para j: = lo a hi - 1 hacer
      si A [j] <pívot entonces
         I = i + 1
         Intercambiar A [i] con A [j]
   Si A [hi] <A [i + 1] entonces
      Intercambiar A [i + 1] con A [hi]
   Devuelve i + 1
```