PAULA RANGEL LÓPEZ
prangell@alumnos.unex.es

TAREA 3
Física Estadística Computacional

04-05-2024

Resumen de Capítulo Sorting and Selection de Numerical Recipes.

Los algoritmos de clasificación y selección son una rama fundamental de las recetas numéricas debido al importante papel que juegan en varias aplicaciones de computación científica. Su eficiencia y precisión pueden impactar significativamente el rendimiento de las simulaciones. Aquí se resumen y explican tres algoritmos esenciales de ordenación de listas: la Inserción directa y el método de Shell, Quicksort, y Heapsort. Los dos primeros son una respuesta intuitiva y directa de implementar ante el problema de ordenación de elementos. Quicksort, por otra parte, es un algoritmo recursivo que ofrece un gran rendimiento cuando se trata con conjunto de datos de considerable tamaño. Heapsort, basado en la comparación, es conocido por su alta estabilidad y eficiencia.

Inserción directa y Método de Shell

El tiempo requerido por el algoritmo de inserción directa es del orden N^2, siendo N el número de entradas al mismo: el tamaño de la lista a ordenar. Por ello se debe emplear con listas de pequeño tamaño, con $N < 20$.

Un pequeño ejemplo nos ayuda a entender mejor cómo funciona este algoritmo. Supongamos que disponemos de una baraja de cartas desordenadas en nuestra mano (nuestra lista `arr[]`). Escogemos una primera carta. Seleccionando una segunda, la comparamos con la que ya teníamos: si es de menos valor la situamos a al izquierda de la primera, o la situamos a la derecha de lo contrario. Escogiendo una tercera carta, la comparamos con las otras dos insertándola en la posición correcta. Repetimos este proceso hasta que acabemos con la baraja. Éstas estarán entonces ordenadas, representando nuestra lista objetivo.

El algoritmo es la implementación directa de este método. Dentro de un bucle que recorre todos los elementos de `arr[]`, almacenamos en una variable muda `a` el elemento correspondiente `arr[j]`, y comenzando desde su posición hacia el inicio de la lista vamos comprobando que el elemento anterior sea mayor que `a`. Si así es nos quedamos con dicha posición y la comparamos con su anterior, hasta llegar al elemento en que sea menor o igual que `a`. En dicho punto insertamos el valor de `a[j]`. Es por esto por lo que almacenamos en una variable muda el valor con el que estmaos comparando, para ahorrarnos el tiempo computacional que requeriríamos al buscar dentro de la lista cada vez que se vaya comparando con los valores sucesivos dentro del `while`. Otra opción en la que se podría estar interesado sería organizar una segunda lista en siguiendo el mismo orden de índices que determina la ordenación de la primera, en cuyo caso tan sólo debemos copiar las mismas instrucciones dentro del bucle asegurándonos de tan sólo comparar el array `arr[]`.

El Método de Shell es una modificación del algoritmo que acabamos de ver. Realiza una comparación *in situ*, ordenando pares de elementos muy lejanos entre sí, hasta acabar comparando elementos inmediatamente próximos. De esta forma, valores muy dispares en zonas más o menos uniformes pueden organizarse de forma rápida que con un intercambio simple entre vecinos.

Comenzando con un intervalo `k` dado entre las posiciones a comparar, denominado incremento, se establecen diferentes sublistas con todos los valores que diferieren en `k` posiciones. Se comparan y se intercambian en la lista original si así fuera necesario. Posteriormente se divide la lista en grupos de elementos con `k-1` posiciones entre ellos y se repite el mismo método anterior, hasta que se compare finalmente la lista completa con todos los números empleando el mismo método que en el algoritmo de Inserción directa. De esta forma, previo a esta última ordenación, los valores estarán cercanos a su posición final, y tan sólo tendrán que moverse unas pocas posiciones hasta quedar completamente colocados. En un conjunto de números aleatoriamente ordenados, $T(N) = \mathcal{O}(N^{1,25})$, cuando el tamaño de la entrada es inferior a 60000.

Quicksort

Este algoritmo se basa en el método de *divide y vencerás* de programación. Se selecciona un elemento que actúe como "pivote" dentro de la lista y se realiza una partición con los demás elementos en dos sublistas, según sean menores o iguales que el pivote, en el caso en el que se colocan en la sublista de la izquierda, o mayores, colocándose entonces en la de la derecha. Las sublistas se ordenan de manera recursivas. La manera en la que se selecciona este pivote `a` puede ser escogiendo el último valor de la lista, el primero o de forma aleatoria. Se aplica la partición escaneando un puntero comenzando por el primer elemento hasta que el valor sea mayor que a, y otro comenzando desde el final hasta encontrar uno menor. Se intercambian ambas

posiciones y se repite el proceso hasta que los punteros se crucen, lo que significa que dicha posición es la de `a` y se ha acabado la partición. En la implementación de Numerical Recipes no se hace uso de la recursividad, si no que se utiliza una lista auxiliar para realizar un seguimiento de las sublistas pendientes de organizar.

Se puede proporcionar como argumento opcional al algoritmo un entero `m` que indica el número de elementos que se ordenan dentro de la lista. `NSTACK` es la profundidad máxima de la pila de recursión. Si el tamaño de la sublista (originada por partición o no) es inferior a `7`, se emplea inserción directa en vez de Quicksort por rapidez. De lo contrario, se selecciona el pivote desde el medio de la sublista y la particiona alrededor de él, mediante el método de los punteros explicado previamente. El algoritmo ordena las sublistas empujándolas a una pila e iterando hasta que la pila quede vacía. Esta pila es una lista de enteros cuyos elementos representan los índicies de inicio y fin de la sublista a ordenar. Se puede implementar este algoritmo de forma que el arreglo de enteros quede ordenado en una nueva lista, pero el método es esencialmente el descrito previamente.

Heapsort

Este algoritmo, aunque más lento que Quicksort por un factor constante, se utiliza a menudo gracias a que no requiere de almacenamiento adicional, si no que ordena la lista verdaderamente *in situ*. Su tiempo de computación es del orden de $N \log_2 N$, e incluso en el peor de los casos tarda un mero 20 % más de su tiempo habitual.

Una *max heap* o pila ordenada por máximo es una manera eficiente de recuperar los elementos mayores de una lista. La raíz o base de la pila siempre es el elemento más grande ya que todos los valores hijos deben ser menores a éste, por lo que recuperar el elemento mayor es tan simple como empujar la base a la base de la pila. Si se repite este proceso hasta que todos los elementos han sido retirados logramos la lista correspondiente ordenada. El algoritmo de Heapsort consiste en transformar la lista en un árbol binario que se convierte a una pila de máximo, asegurándonos que todos los nodos padre sean mayores o iguales que los hijos. Se intercambia el nodo raíz por el último de la pila, promocionando su hijo de mayor valor al nivel `a_0`. La raíz inicial -ahora en la última posición- dejará de participar en las siguientes iteraciones ya que se encuentra en su posición final. Se repite el proceso con el resto de nodos hijos: se transforma la lista en un árbol, y ésta en una pila de máximo comprobando que se encuentren todos los valores con menor valor que el padre como hijos. Finalmente se alcanza la lista ordenada de menor a mayor.