NOMNRE DEL ALUMNO: JOSE CARLOS TOLEDO PEREZ

NOMBRE DEL PROFESOR: Emmanuel Eduardo Sánchez Pérez

MARERIA: ALGORITMOS Y ESTRUCTURAS DE DATOS

TIPO DE TRABAJO: ensayo

LICENCIATURA: INGENIERIA EN SISTEMAS COMPUTACIONALES

CUATRIMESTRE: 5

26/03/2023

UNIDAD IV

El proceso de ordenar elementos (“sorting”) en base a alguna relación de orden (ver sección §2.4.4) es un problema tan frecuente y con tantos usos que merece un capítulo aparte. Nosotros nos concentraremos en el ordenamiento interna, es decir cuando todo el contenedor reside en la memoria principal de la computadora. El tema del ordenamiento externo, donde el volumen de datos a ordenar es de tal magnitud que requiere el uso de memoria auxiliar, tiene algunas características propias y por simplicidad no será considerado aquí.

4.1 INTRODUCCIÓN Asumiremos normalmente que los elementos a ordenar pertenecen a algún tipo key_t con una relación de orden < y que están almacenados en un contenedor lineal (vector o lista). El problema de ordenar un tal contenedor es realizar una serie de intercambios en el contenedor de manera de que los elementos queden ordenados, es decir k0 ≤ k1 ≤ ... ≤ kn−1, donde kj es la clave del j-ésimo elemento. Si el ordenamiento se hace sin requerir ningún tipo de memoria adicional (que crezca con n), decimos que es “in-place”. Si no, se debe tener en cuenta también como crece la cantidad de memoria requerida.

4.1.1 RELACIONES DE ORDEN DÉBILES Ya hemos discutido como se define una relación de orden en la sección §2.4.4. Igual que para definir conjuntos o correspondencias, la relación de ordenamiento puede a veces ser elegida por conveniencia. Así, podemos querer ordenar un conjunto de números por su valor absoluto. En ese caso al ordenar los números (1, 3, −2, −4) el contenedor ordenado resultaría en (1, −2, 3, −4). En este caso la relación de orden la podríamos denotar por y la definiríamos como La relación binaria definida así no resulta ser una relación de orden en el sentido fuerte, como definida en la sección §2.4.4, ya que, por ejemplo, para el par −2, 2 no son ciertos 2 −2, ni −2 2 ni −2 = 2. La segunda condición sobre las relaciones de orden puede relajarse un poco y obtener la definición de relaciones de orden débiles: Definición: “” es una relación de orden débil en el conjunto C si, Esta última condición se llama de antisimetría. Usamos aquí la notación de C++ para los operadores lógicos, por ejemplo && indica “and” y los valores booleanos true y false. Es decir, a b y b a no pueden ser verdaderas al mismo tiempo (son exclusivas).

4.1.2 SIGNATURA DE LAS RELACIONES DE ORDEN. PREDICADOS BINARIOS

donde la función tolower() convierte su argumento a minúsculas. La función tolower() está definida en la librería estándar de C (libc) para caracteres. Podemos entonces definir la función de comparación para orden lexicográfico independiente de mayúsculas/minúsculas como se muestra en el código 5.4. La función string_less_ci() es básicamente igual a la string_less_cs() sólo que antes de comparar caracteres los convierte con tolower() a minúsculas. De esta forma pepe, Pepe y PEPE resultan equivalentes.

4.2 MÉTODOS DE ORDENAMIENTO LENTOS Llamamos “rápidos” a los métodos de ordenamiento con tiempos de ejecución menores o iguales a O(n log n). Al resto lo llamaremos “lentos”. En esta sección estudiaremos tres algoritmos lentos, a saber, burbuja, selección e inserción.

4.2.1 EL MÉTODO DE LA BURBUJA Código 5.5: Algoritmo de ordenamiento de la burbuja. [Archivo: bubsort.h El método de la burbuja fue introducido en la sección §1.4.2. Nos limitaremos aquí a discutir la conversión al formato compatible con la STL. El código correspondiente puede observarse en el código 5.5. • Para cada algoritmo de ordenamiento presentamos las dos funciones correspondientes, con y sin función de comparación. • Ambas son templates sobre el tipo T de los elementos a ordenar.

La que no tiene operador de comparación suele ser un “wrapper” que llama a la primera pasándole como función de comparación less.

• Notar que como las funciones no reciben un contenedor, sino un rango de iteradores, no se puede referenciar directamente a los elementos en la forma v[j] sino a través del operador de dereferenciación *p. Así, donde normalmente pondríamos v[first+j] debemos usar *(first+j).

• Recordar que las operaciones aritméticas con iteradores son válidas ya que los contenedores son de acceso aleatorio. En particular, las funciones presentadas son sólo válidas para vector<>, aunque se podrían modificar para incluir a deque<> en forma relativamente fácil

• Notar la comparación en la línea 8 usando el predicado binario comp() en vez de operator

• Para la discusión de los diferentes algoritmos haremos abstracción de la posición de comienzo first, como si fuera 0. Es decir consideraremos que se está ordenando el rango [0,n) donde n=last-firs

4.2.2 EL MÉTODO DE INSERCIÓN

En este método (ver código 5.6) también hay un doble lazo. En el lazo sobre j el rango [0, j) está ordenado e insertamos el elemento j en el rango [0, j), haciendo los desplazamientos necesarios. El lazo sobre k va recorriendo las posiciones desde j hasta 0 para ver donde debe insertarse el elemento que en ese momento está en la posición j.

4.2.3 EL MÉTODO DE SELECCIÓN

En este método (ver código 5.7) también hay un doble lazo (esta es una característica de todos los algoritmos lentos). En el lazo j se elige el menor del rango [j, N) y se intercambia con el elemento en la posición j.

4.3 ORDENAMIENTO INDIRECTO

Si el intercambio de elementos es muy costoso (pensemos en largas listas, por ejemplo) podemos reducir notablemente el número de intercambios usando "ordenamiento indirecto", el cual se basa en ordenar un vector de cursores o punteros a los objetos reales. De esta forma el costo del intercambio es en realidad el de intercambio de los cursores o punteros, lo cual es mucho más bajo que el intercambio de los objetos mismos.

4.3.1 MINIMIZAR LA LLAMADA A FUNCIONES

Si la función de comparación se obtiene por composición, y la función de mapeo es muy costosa. Entonces tal vez convenga generar primero un vector auxiliar con los valores de la función de mapeo, ordenarlo y después aplicar la permutación resultante a los elementos reales. De esta forma el número de llamadas a la función de mapeo pasa de ser $O(n^2)$ a $O(n)$. Por ejemplo, supongamos que tenemos un conjunto de árboles y queremos ordenarlos por la suma de sus valores nodales. Podemos escribir una función Si aplicamos por ejemplo bubble_sort, entonces el número de llamadas a la función será $O(n^2)$. Si los árboles tienen muchos nodos, entonces puede ser preferible generar un vector de enteros con los valores de las sumas y ordenarlo. Luego se aplicaría la permutación correspondiente al vector de árboles.

4.4. EL MÉTODO DE ORDENAMIENTO RÁPIDO, QUICK-SORT

Este es probablemente uno de los algoritmos de ordenamiento más usados y se basa en la estrategia de "dividir para vencer". Se escoge un elemento del vector v llamado "pivote" y se "particiona" el vector de manera de dejar los elementos ≥ v a la derecha (rango [l, n), donde l es la posición del primer elemento de la partición derecha) y los < v a la izquierda (rango [0, l)). Está claro que a partir de entonces, los elementos en cada una de las particiones quedarán en sus respectivos rangos, ya que todos los elementos en la partición derecha son estrictamente mayores que los de la izquierda. Podemos entonces aplicar quick-sort recursivamente a cada una de las particiones.

4.4.1 TIEMPO DE EJECUCIÓN.

CASOS EXTREMOS donde n1 = l − j1 y n2 = j2 − l son las longitudes de cada una de las particiones. Tpart−piv(n) es el costo de particionar la secuencia y elegir el pivote. El mejor caso es cuando podemos elegir el pivote de tal manera que las particiones resultan ser bien balanceadas, es decir n1 ≈ n2 ≈ n/2. Si además asumimos que el algoritmo de particionamiento y elección del pivote son O(n).

El peor caso ocurre, por ejemplo, si el vector está inicialmente ordenado y usando como estrategia para el pivote el mayor de los dos primeros distintos. Si tenemos en v, por ejemplo los enteros 1 a 100 ordenados, que podemos denotar como un rango [1, 100], entonces el pivote sería inicialmente 2.

4.4.2 ELECCIÓN DEL PIVOTE

En el caso promedio, el balance de la partición dependerá de la estrategia de elección del pivote y de la distribución estadística de los elementos del vector. Compararemos a continuación las estrategias que corresponden a tomar la "mediana" de los k primeros distintos. Recordemos que para k impar la mediana de k elementos es el elemento que queda en la posición del medio del vector (la posición (k − 1)/2 en base 0) después de haberlos ordenado. Para k par tomamos el elemento en la posición k/2 (base 0) de los primeros k distintos, después de ordenarlos.