
UDS
Mi Universidad

Ensayo

Nombre del alumno: Yahir Aguilar Sicalhua.

Nombre del tema: Unidad III. Árboles y Unidad IV. Ordenamiento.

Parcial: 1.

Nombre de la materia: Algoritmos y estructuras de datos.

Nombre del profesor: ISC. Emmanuel Eduardo Sánchez Pérez.

Nombre de la licenciatura: Ingeniería en Sistemas Computacionales.

Cuatrimestre: 5.

Un árbol se puede definir como una estructura jerárquica y en forma no lineal, aplicada sobre una colección de elementos u objetos llamados nodos. (Cairó & Guardati, 2006). Lineales y dinámicas de datos muy importantes del área de computación.

El ordenamiento de datos (es decir, colocar los datos en cierto orden específico, como ascendente o descendente) es una de las aplicaciones computacionales más importantes. Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia específica.

UNIDAD III. ÁRBOLES.

3.1. NOMENCLATURA BÁSICA DE ÁRBOLES.

Un árbol es una colección de elementos llamados “nodos”, uno de los cuales es la “raíz”. Existe una relación de parentesco por la cual cada nodo tiene un y sólo un “padre”, salvo la raíz que no lo tiene.

3.1.0.0.2. PROFUNDIDAD DE UN NODO. NIVEL.

La “profundidad” de un nodo es la longitud de único camino que va desde el nodo a la raíz. La profundidad del nodo g en el ejemplo es 2. Un “nivel” en el árbol es el conjunto de todos los nodos que están a una misma profundidad. El nivel de profundidad 2 en el ejemplo consta de los nodos e, f y g.

3.1.0.0.3. NODOS HERMANOS.

Se dice que los nodos que tienen un mismo padre son “hermanos” entre sí. Notar que no basta con que dos nodos estén en el mismo nivel para que sean hermanos. Los nodos f y g en el árbol de la figura 3.3 están en el mismo nivel, pero no son hermanos entre sí.

3.2. ORDEN DE LOS NODOS.

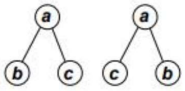

Figura 3.4: Arboles ordenados: el orden de los hijos es importante de manera que los árboles de la figura son diferentes.

3.2.1. PARTICIONAMIENTO DEL CONJUNTO DE NODOS.

Ahora bien, dados dos nodos cualquiera m y n consideremos sus caminos a la raíz. Si m es descendiente de n entonces el camino de n está incluido en el de m o viceversa. Por ejemplo, el camino de c, que es a, c, está incluido en el de h, a, c, g, h, ya que c es antecesor de h. Si entre m y n no hay relación de descendiente o antecesor, entonces los caminos se deben bifurcar necesariamente en un cierto nivel.

3.2.2.1. ORDEN PREVIO.

oExisten varias formas de recorrer un árbol listando los nodos del mismo, generando una lista de nodos. Dado un nodo n con hijos n1, n2, . . . nm, el “listado en orden previo” (“preorder”) del nodo n que denotaremos como oprev(n) se puede definir recursivamente como sigue.

3.2.2.2. ORDEN POSTERIOR.

Recorriendo el borde del árbol igual que antes (esto es en sentido contrario a las agujas del reloj), listando el nodo la última vez que el recorrido pasa por al lado del mismo. Por ejemplo, el nodo b sería listado al pasar por el punto 3. Recorriendo el borde en el sentido opuesto (es decir en el mismo sentido que las agujas del reloj), y listando los nodos la primera vez que el camino pasa cera de ellos. Una vez que la lista es obtenida, invertimos la lista. En el caso de la figura el recorrido en sentido contrario daría (a, d, c, g, h, b, f, e).

3.2.2.3. ORDEN POSTERIOR Y LA NOTACIÓN POLACA INVERTIDA.

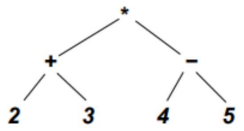

Figura 3.9: Arbol correspondiente a la expresión matemática $(2+3)*(4-5)$

3.3.1. ALGORITMOS PARA LISTAR NODOS.

Implementar un algoritmo para recorrer los nodos de un árbol es relativamente simple debido a su naturaleza intrínsecamente recursiva, expresada en (3.3). Un posible algoritmo puede observarse en el código 3.1. Si bien el algoritmo es genérico hemos usado ya algunos conceptos familiares de las STL, por ejemplo, las posiciones se representan con una clase iterator.

3.3.2. INSERCIÓN EN ÁRBOLES.

Para construir árboles necesitaremos rutinas de inserción supresión de nodos. Como en las listas, las operaciones de inserción toman un elemento y una posición e insertan el elemento en esa posición en el árbol.

3.3.2.1. ALGORITMO PARA COPIAR ÁRBOLES.

Con menores modificaciones la función puede copiar un árbol en forma espejada, es decir, de manera que todos los nodos hermanos queden en orden inverso entre sí. Para eso basta con no avanzar el iterator cq donde se copian los subárboles, es decir eliminar la línea 11 . Recordar que si en una lista se van insertando valores en una posición sin avanzarla, entonces los elementos quedan ordenados en forma inversa a como fueron ingresados.

3.5. IMPLEMENTACIÓN DE LA INTERFAZ BÁSICA POR PUNTEROS.

Así como la implementación de listas por punteros mantiene los datos en celdas enlazadas por un campo next, es natural considerar una implementación de árboles en la cual los datos son almacenados en celdas que contienen, además del dato elem, un puntero right a la celda que corresponde al hermano derecho y otro left_child al hijo más izquierdo.

3.5.1. EL TIPO ITERATOR.

hPor analogía con las listas podríamos definir el tipo iterator_t como un typedef a cell *. Esto bastaría para representar posiciones dereferenciables y posiciones no dereferenciables que provienen de haber aplicado right() al último hermano.

3.5.2. LAS CLASES CELL E ITERATOR_T.

El constructor por defecto iterator_t() devuelve un iterator con los tres punteros nulos. Este iterator no debería ser normalmente usado en ninguna operación, pero es invocado automáticamente por el compilador cuando declaramos iterators como en: iterator p.

UNIDAD IV. ORDENAMIENTO.

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 $k0 \leq k1 \leq ... \leq kn-1$, donde kj es la clave del j-ésimo elemento.

4.1.1 RELACIONES DE ORDEN DÉBILES.

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.

4.1.2 SIGNATURA DE LAS RELACIONES DE ORDEN. PREDICADOS BINARIOS.

Podemos entonces definir la función de comparación para orden lexicográfico independiente de mayúsculas/minúsculas. 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.

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.

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.

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 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).

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)).

4.4.1 TIEMPO DE EJECUCIÓN. CASOS EXTREMOS.

El tiempo de ejecución del algoritmo aplicado a un rango $[j_1, j_2)$ de longitud $n = j_2 - j_1$ es

$$T(n) = T_{\text{part}-\text{piv}}(n) + T(n_1) + T(n_2)$$

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.

En conclusión diríamos que el algoritmo es de carácter general y puede aplicarse a cualquier operación matemática o a cualquier problema.

La formulación de algoritmos fue uno de los más grandes adelantos dentro de la ciencia matemática ya que a partir de ellos se pudieron resolver infinidad de problemas.

Fuentes de información:

MWConsultores. (s/f). Antología UDS. Com.mx Recuperado el 15 de abril de 2023, de https://plataformaeducativauds.com.mx/libro.php?idLibro=168161402216