75.04 Algoritmos y Programación II

Práctica 6: árboles

Notas preliminares

- El objetivo de esta práctica es introducir distintas clases de estructuras de datos arbóreas y algoritmos para manipularlas.
- Los ejercicios marcados con el símbolo ♣ constituyen un subconjunto mínimo de ejercitación. No obstante, recomendamos fuertemente realizar todos los ejercicios.

---

1. Árboles binarios

Ejercicio 1

Dada una posible implementación dinámica de árboles binarios en C++,

```
template<typename T>
class bintree {
        T *t_;
        int n_;
        bintree *izq_;
        bintree *der_;
public:
        // ...
};
```

Hallar una relación matemática simple entre el número de punteros *null* y la cantidad de nodos del árbol, n. Mostrar, además, que esta relación no depende de la forma del árbol. Discutir, a partir de este resultado, cuánto espacio se "desperdicia" en punteros *null*.

Ejercicio 2 ♣

(a) Programar una función recursiva que, dado un árbol binario de n nodos, imprima la clave de cada nodo del árbol. Además, la complejidad debe ser una $O(n)$.

(b) Programar una función no recursiva que, dado un árbol binario de n nodos, imprima la clave de cada nodo del árbol. Usar una pila como estructura de datos auxiliar. Además, la complejidad debe ser una $O(n)$.

(c) Programar una función que, dado un árbol binario de n nodos, imprima la clave de cada nodo del árbol. Además, sólo se permite usar una cantidad de espacio de almacenamiento constante (es decir, la cantidad de memoria extra a usar -sin contar la memoria consumida por la representación del árbol- no debe depender de n). No se permite modificar el árbol, ni siquiera temporalmente, durante el proceso. La complejidad resultante debe ser una $O(n)$.

Nota: se puede suponer que la implementación provee un método `up()` que, dado un nodo o subárbol, devuelve el padre de ese nodo en tiempo constante.

Ejercicio 3

Dado un árbol binario t y dos nodos $u, v \in t$, la *distancia* $d(u, v)$ entre u y v es la cantidad de ejes que se deben atravesar en t para llegar desde u a v.

La *longitud de camino* L_t de t se define como la suma de las distancias de la raíz r_t de t a cada uno de sus nodos:

$$L_t = \sum_{v \in t} d(r_t, v)$$

Un *árbol binario extendido* es un árbol binario que se obtiene al reemplazar cada hijo nulo por un nuevo nodo especial en un árbol binario arbitrario. Los nodos originales se llaman *internos*, mientras que los agregados se llaman *externos*. De esta manera, definimos la *longitud de camino interna* (resp. *externa*) I_t (resp. E_t) de un árbol binario extendido t como la suma de las distancias de la raíz a cada uno de los nodos internos (resp. externos) de t.

(a) Probar que la longitud de camino de un árbol binario completo t de n nodos es

$$L_t = 2 + (n+1)\,l - 2^{l+1}$$

En donde $l = \lfloor \log_2(n+1) \rfloor$. Obtener, además, una expresión $\mathrm{O}(\cdot)$ de la profundidad promedio de nodo en un árbol binario completo.

(b) Escribir un algoritmo en C++ para calcular la I_t de un árbol extendido t. ¿Cúal es la complejidad resultante?

(c) Escribir un algoritmo en C++ para calcular la E_t de un árbol extendido t. ¿Cúal es la complejidad resultante?

(d) Encontrar una relación matemática entre la E_t y la I_t en un árbol n-ario t. Dar ejemplos y justificar.

Ejercicio 4

Siguiendo con las definiciones introducidas en el ejercicio anterior, un árbol binario extendido de m nodos externos determina una secuencia $\{l_1, \ldots, l_m\}$ que describe las m longitudes de camino desde la raíz hasta los nodos externos respectivos.

Contrariamente, dados $\{l_1, \ldots, l_m\}$, ¿es siempre posible construir un árbol binario extendido en el cual estos números son las longitudes de camino?

Mostrar que esto es posible si y sólo si

$$1 = \sum_{k=1}^{m} 2^{-l_k}$$

Ejercicio 5 ♣

Probar que, conociendo los recorridos *preorder* e *inorder* de un árbol binario arbitrario sin elementos repetidos, se puede reconstruir la estructura original del mismo.

¿Puede afirmarse un resultado similar conociendo los recorridos en *preorder* y *postorder* (en vez de *inorder*)? ¿Y *postorder* e *inorder*?

Ejercicio 6

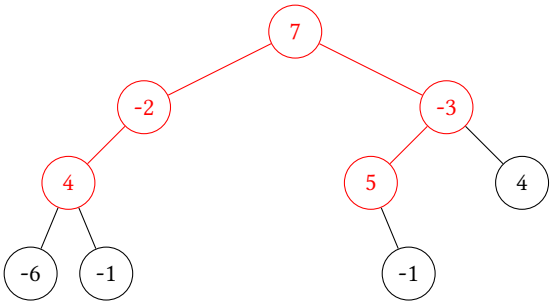
Dado un árbol binario de enteros t y dos nodos $u, v \in t$, llamamos *peso de camino* entre u y v, notado $\pi(u, v)$, a la suma del valor asociado a cada nodo en el camino que une a u con v, incluyendo a ellos mismos. Cuando es $u = v$, $\pi(u, v)$ es el valor asociado a u.

(a) Diseñar un algoritmo que, dado un árbol binario t de n enteros, determine el máximo peso de camino $\boldsymbol{\pi}_t$ entre dos nodos cualesquiera de t, y calcular luego su complejidad temporal. Puesto en términos formales, se desea calcular lo siguiente:

$$\boldsymbol{\pi}_t = \text{máx}\,\{\pi(u, v) \,/\, u, v \in t\}$$

Por ejemplo, para el árbol t mostrado abajo, el algoritmo debe devolver $\boldsymbol{\pi}_t = 11$, que corresponde al camino formado por los nodos marcados en rojo.

(b) Pensar cómo mejorar el algoritmo anterior para que corra en tiempo $\mathrm{O}(n)$ y no pase más de una vez por cada nodo de t.
Hint: considerar el uso de la técnica de generalización de funciones.



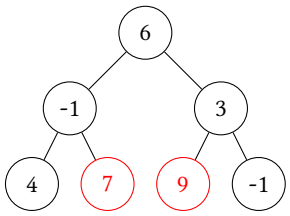
Ejercicio 7 ♣

Sea t un árbol binario completo de n enteros.

(a) Proponer un algoritmo iterativo que corra en tiempo $\Theta(n)$ para encontrar la mínima distancia $\boldsymbol{\delta}_t$ entre dos nodos adyacentes de t. En otras palabras, se debe calcular lo siguiente:

$$\boldsymbol{\delta}_t = \text{mín}\,\{|u - v| \,/\, u \text{ y } v \text{ son nodos adyacentes en } t\}$$

Por ejemplo, para el árbol t mostrado abajo, el algoritmo debe devolver $\boldsymbol{\delta}_t = 2$, que corresponde a los nodos marcados en rojo.



(b) Proponer un algoritmo que implemente la técnica de dividir y conquistar para calcular $\boldsymbol{\delta}_t$. Se debe mantener la misma complejidad temporal que en el caso anterior.

(c) Calcular la complejidad espacial de ambos algoritmos. ¿Cuál resulta más eficiente?

2. Árboles binarios de búsqueda y balanceados

Ejercicio 8 ♣

(a) Para cada secuencia de claves, dibujar el árbol binario de búsqueda que resulta de insertar las claves, una por una, en un árbol inicialmente vacío:

(i) $\{1, 2, 3, 4, 5, 6, 7\}$;
(ii) $\{4, 2, 1, 3, 6, 5, 7\}$;
(iii) $\{1, 6, 7, 2, 4, 3, 5\}$.

(b) Repetir el ejercicio anterior usando árboles AVL.

Ejercicio 9

(a) Proponer un algoritmo que reciba un árbol AVL t y dos valores, a y b, $a \leq b$, y visite todas las claves $x \in t$ tales que $a \leq x \leq b$. El tiempo de corrida del algoritmo debe ser $O(k + \log n)$, donde k es el número de claves visitadas y n el número total de elementos en el árbol.

(b) Analizar cómo cambiaría la complejidad si t fuese un árbol binario de búsqueda arbitrario en lugar de un árbol AVL.

(c) Analizar cómo adaptar el algoritmo para soportar como entrada árboles binarios arbitrarios. ¿Qué pasa con la complejidad en este caso?

Ejercicio 10 ♣

(a) Proponer la interfaz de una clase C++ para representar árboles AVL.

(b) Escribir un método en dicha clase de complejidad sublineal que permita calcular la altura de un árbol AVL (i.e., la mayor de las distancias de los caminos que unen la raíz con cada una de las hojas).

Ejercicio 11

(a) Escribir una función C++ que reciba un arreglo $A[1 \dots n]$ de números enteros, ordenado ascendentemente, y construya, en tiempo lineal, un árbol binario de búsqueda conteniendo esa información. El árbol generado deberá ser balanceado, es decir, es necesario que la profundidad de todas las hojas sea l o $l - 1$ para algún $l \in \mathbb{N}$.

(b) Explicar qué cambios se necesitaría introducir en el código si se requiriese obtener un árbol completo (i.e., todas las hojas a profundidad l o $l - 1$ y estando las hojas del nivel l ocupando las posiciones consecutivas desde más a la izquierda, sin dejar huecos).

Ejercicio 12

Analizar si cada una de las siguientes aseveraciones es verdadera o falsa, justificando cuidadosamente las respuestas dadas:

(a) Existe un algoritmo que, dados n números, construye un árbol AVL en tiempo $\mathrm{O}(n)$.
(b) La cantidad de hojas de cualquier árbol binario de búsqueda de n nodos es $\Theta(n)$.
(c) La cantidad de hojas de cualquier árbol AVL de n nodos es $\Theta(n)$.
(d) Dado un árbol binario de búsqueda arbitrario t, es posible reconstruir a t a partir de un arreglo $P[1 \dots n]$ que representa su recorrido preorder.
(e) Dado un árbol AVL arbitrario, la diferencia de tamaño (i.e., cantidad de nodos) entre sus subárboles es $\mathrm{O}(1)$.
(f) Sea t un árbol binario completo de n nodos. Dado un nodo v en t, encontrar un camino desde la raíz de t a v utilizando el algoritmo BFS (i.e., el recorrido por niveles) lleva tiempo $\mathrm{O}(\log n)$.

Ejercicio 13

Un *árbol RN* es un árbol binario cuyos nodos pueden ser rojos o negros y que además verifica las siguientes condiciones:

- Todas sus hojas son de color negro,
- Los nodos rojos pueden tener únicamente hijos negros, y
- Cada camino desde la raíz hasta cualquiera de las hojas tiene la misma cantidad de nodos negros.

(a) Proponer un algortimo que utilice la técnica de dividir y conquistar para determinar si un árbol binario es RN. Asumir que se cuenta con una función `color()` para conocer el color de un nodo en tiempo constante.

(b) Calcular la complejidad temporal de peor caso del algoritmo anterior.

Ejercicio 14

Sea t un árbol AVL arbitrario, y sea $P[1 \dots n]$ un arreglo conteniendo el recorrido preorder de t. Diseñar un algoritmo que, dado e y el arreglo P, determine si $e \in t$ en tiempo estrictamente inferior que $\mathrm{O}(n)$. Indicar claramente la complejidad temporal del algoritmo desarrollado.

Ejercicio 15 ♣

(a) Proponer un algoritmo que, dado un árbol binario arbitrario t de n nodos, determine si t es un árbol de búsqueda en tiempo $O(n)$.

(b) Proponer un algoritmo que, dado un árbol binario arbitrario t de n nodos, determine si t es un árbol AVL en tiempo $O(n)$.

Hint: para ambos ítems, considerar el uso de la técnica de generalización de funciones.