

Hoja 3: Ejercicio 4

Ejercicios de montículos

Diego Rodríguez Cubero

UCM

19 de octubre de 2025

Contenidos

- 1 Enunciado del problema
- 2 Planteamiento
- 3 Desarrollo
- 4 Ejemplo
- 5 Complejidad temporal
- 6 Conclusión

Ejercicio 4

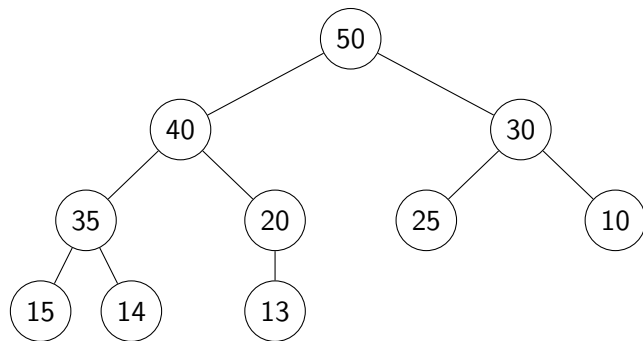
Diseñar un algoritmo que compruebe si un vector $V [1..n]$ es un montículo de máximos y determinar su complejidad temporal.

Definición

Un montículo de máximos es un árbol binario semicompleto (todos los niveles llenos excepto posiblemente el último, que se llena de izquierda a derecha), que además cumple que su raíz es mayor que las de sus hijos, los cuales a su vez son montículos de máximos.

Esto último es equivalente a decir que cada nodo es mayor que sus descendientes.

Ejemplo de montículo de máximos



$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

Esta representación como vector se obtiene recorriendo el árbol por niveles, por lo que el primer elemento del nivel h se encuentra en la posición $2^h - 1$ del vector, y esta seguido por el resto de elementos del nivel.

Idea del ejercicio

$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

Primero es importante notar que si un nodo esta en la posición i del vector, sus hijos estarán en las posiciones $2i + 1$ y $2i + 2$

Por ejemplo el elemento 40 que esta en la posición 1 tiene como hijos a los elementos 35 y 20 que están en las posiciones 3 y 4 respectivamente.

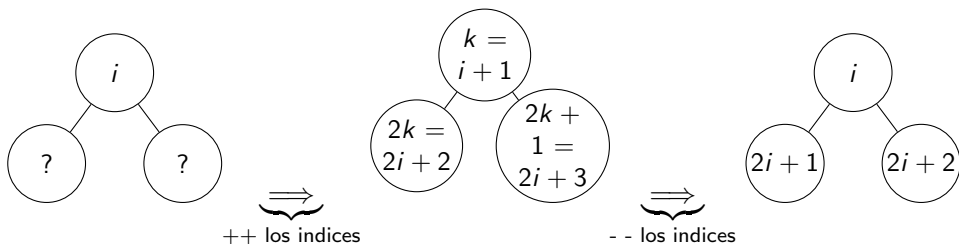
Asumimos que el arbol es semicompleto, y por tanto el vector no tiene ningún "hueco".

Usaremos esto para recorrer todos los nodos que tienen hijos y comprobar que sean mayores que ellos.

Notación importante

Por comodidad en el algoritmo, llamaremos $i = 0$ a la raíz del árbol, y los hijos de cada nodo i estarán en las posiciones $2i + 1$ y $2i + 2$.

Esto se deduce de la notación empleada en clase donde los hijos del nodo k están en las posiciones $2k$ y $2k + 1$, pero aquí el vector empieza en la posición 0 en lugar de la 1.



Listing 1: Función para comprobar si un vector es un montículo de máximos

```
bool esMonticuloMaximos(int V[], int n) {  
    for (int i = 0; i <= (n - 2) / 2; i++) { // Recorremos nodos con  
        hijos  
        int izq = 2 * i + 1; // Hijo izquierdo  
        int der = 2 * i + 2; // Hijo derecho  
        if (izq < n && V[i] < V[izq]) {  
            return false; // Violación de la propiedad del montículo  
        }  
        if (der < n && V[i] < V[der]) {  
            return false; // Violación de la propiedad del montículo  
        }  
    }  
    return true; // Es un montículo de máximos  
}
```


Ejemplo de ejecución

- Sea $V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$ y $n = 10$.
- Iteramos desde $i = 0$ hasta $(n - 2)/2 = 4$ (incluido).
- Comprobamos cada nodo con sus hijos.

Paso $i = 0$

$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

- Como $i = 0 \implies$ hijos en $2 * 0 + 1 = 1$ y $2 * 0 + 2 = 2$
- $V[0] = 50$
- Hijos: $V[1] = 40$, $V[2] = 30$
- $50 > 40$ y $50 > 30$ (cumple)

Paso $i = 1$

$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

- Como $i = 1 \implies$ hijos en $2 * 1 + 1 = 3$ y $2 * 1 + 2 = 4$
- $V[1] = 40$
- Hijos: $V[3] = 35$, $V[4] = 20$
- $40 > 35$ y $40 > 20$ (cumple)

Paso $i = 2$

$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

- Como $i = 2 \implies$ hijos en $2 * 2 + 1 = 5$ y $2 * 2 + 2 = 6$
- $V[2] = 30$
- Hijos: $V[5] = 25$, $V[6] = 10$
- $30 > 25$ y $30 > 10$ (cumple)

$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

- Como $i = 3 \implies$ hijos en $2 * 3 + 1 = 7$ y $2 * 3 + 2 = 8$
- $V[3] = 35$
- Hijos: $V[7] = 15$, $V[8] = 14$
- $35 > 15$ y $35 > 14$ (cumple)

Paso $i = 4$

$$V = [50, 40, 30, 35, 20, 25, 10, 15, 14, 13]$$

- Como $i = 4 \implies$ hijos en $2 * 4 + 1 = 9$ y $2 * 4 + 2 = 10$
- $V[4] = 20$
- Hijos: $V[9] = 13$, $V[10]$ no existe ($10 \geq n = 10$)
- $20 > 13$ (cumple)

- El bucle for itera desde $i = 0$ hasta $(n - 2)/2$, lo que implica $\frac{n}{2} - 1$ iteraciones, es decir, $O(n)$ iteraciones.
- En cada iteración, se realizan un número constante de operaciones (comparaciones y asignaciones), todas ellas de tiempo $O(1)$, al tener vectores que nos permiten acceso directo a sus elementos.
- Por lo tanto, el tiempo total del algoritmo es la suma del tiempo de todas las iteraciones:

$$T(n) = \sum_{i=0}^{(n-2)/2} O(1) = O(n)$$

- Por lo tanto, la complejidad temporal total del algoritmo es $O(n)$.

- Hemos diseñado un algoritmo eficiente para comprobar si un vector representa un montículo de máximos.
- El algoritmo recorre todos los nodos con hijos y verifica la propiedad del montículo en tiempo lineal $O(n)$.