Esame I.A. 1 settembre 2020

Soluzioni

1 Esercizio 1

Data la famiglia di equazioni di ricorrenza indicata sotto, per valori di a interi positivi, individuare i limiti superiori e inferiori più stretti possibile:

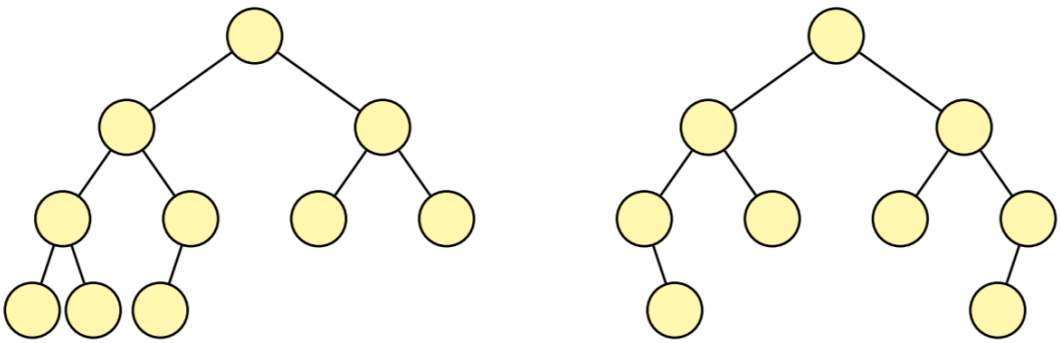
$$T(n) = \begin{cases} aT(\lfloor n/2 \rfloor) + n^{a-1} & n \geq 2 \\ 1 & n < 2 \end{cases}$$

Soluzione Data la forma della ricorrenza, è possibile utilizzare il Master Theorem:

- Per $a = 1$, abbiamo $\alpha = \log_2 1 = 0$, $\beta = 0$, da cui $T(n) = \Theta(\log n)$;
- Per $a = 2$, abbiamo $\alpha = \log_2 2 = 1$, $\beta = 1$, da cui $T(n) = \Theta(n \log n)$;
- Per $a = 3$, abbiamo $\alpha = \log_2 3 = 2$, $\beta = 2$, da cui $T(n) = \Theta(n^2)$;
- Proseguendo, si ottiene che in generale $\log_2 a < a - 1$ per tutti i valori interi $a \geq 3$, pertanto si ottiene $T(n) = \Theta(n^{a-1})$.

2 Esercizio 2

Un albero binario è simmetrico se il sottoalbero sinistro della radice è un'immagine speculare del sottoalbero destro della radice. Nella figura sotto, l'albero binario a sinistra non è simmetrico, mentre quello di destra lo è. Scrivere un programma in python IsSymmetric(T) che prenda in input un albero binario T non vuoto e restituisca `True` se T è simmetrico, `False` altrimenti.



Soluzione Un sottalbero è simmetrico se i suoi sottoalberi destro e sinistro sono speculari. Due sottoalberi $t1$, $t2$ sono speculari se il sottoalbero sinistro di $t1$ è speculare al sottoalbero destro di $t2$ e il sottoalbero destro di $t1$ è speculare al sottoalbero sinistro di $t1$. Il caso base è dato da due nodi nulli (si ritorna `True`) o da uno nodo nullo e un nodo non nullo (si ritorna `False`). Una possibile implementazione è la seguente:

```
class TreeNode:
    def __init__(self, tL, tR):
        self.left = tL
        self.right = tR


def isSymmetric(T):
    return isMirrored(T.left, T.right)


def isMirrored(tL, tR):
    if tL is None and tR is None:
        return True
    elif tL is not None and tR is not None:
        return isMirrored(tL.right, tR.left) and isMirrored(tL.left, tR.right)
    else:
        return False
```

Per testarla sugli alberi di esempio forniti nell'immagine sopra, è possibile utilizzare il seguente programma:

```
# Albero nella figura di esempio a sinistra
Tleft = TreeNode(TreeNode(TreeNode(TreeNode(None, None), TreeNode(None, None)),
                                        TreeNode(TreeNode(None, None), None)),
                                        TreeNode(TreeNode(None, None), TreeNode(
                                        None, None)))

print("Il primo albero e' simmetrico:", isSymmetric(Tleft))

# Albero nella figura di esempio a destra
Tright = TreeNode(TreeNode(TreeNode(None, TreeNode(None, None)), TreeNode(None,
                                        None)),TreeNode(TreeNode(None, None),
                                        TreeNode(TreeNode(None, None), None)))

print("Il secondo albero e' simmetrico:", isSymmetric(Tright))
```

La procedura effettua una visita su entrambi gli alberi, e quindi ha complessità $\Theta(n)$.

3 Quiz

Dato un vettore composto dai seguenti elementi: 15,20,10,18. Quali sono i passi intermedi di ordinamento quando questo viene ordinato utilizzando l'algoritmo Insertion Sort?

- ✓ 15,20,10,18 – 10,15,20,18 – 10,15,18,20 – 10,15,18,20.
- 15,18,10,20 – 10,18,15,20 – 10,15,18,20 – 10,15,18,20.
- 15,10,20,18 – 15,10,18,20 – 10,15,18,20.
- 10, 20,15,18 – 10,15,20,18 – 10,15,18,20.

Se tutti gli elementi di un vettore sono uguali, ad esempio [1,1,1,1,1,1], qual è la complessità computazionale dell'Insertion Sort?

- $O(2n)$
- $O(n^2)$
- ✓ $O(n)$
- Nessuna delle altre risposte

Affinché l'algoritmo di ricerca binaria possa funzionare correttamente, è necessario che il vettore di elementi all'interno del quale effettuare la ricerca sia:

- Inserito in una pila
- Inserito in un mucchio
- Non ordinato
- ✓ Ordinato

Quali dei seguenti algoritmi di ordinamento non sono stabili?

- Bubble sort
- Merge sort
- ✓ Selection sort
- Insertion sort

In quale caso una coda circolare può essere considerata piena?

- read = (read + 1) % size
- write = (write + 1) % size
- ✓ read = (write + 1) % size
- read = -1