Alberi binari di ricerca- Esercizi

Pietro Di Lena

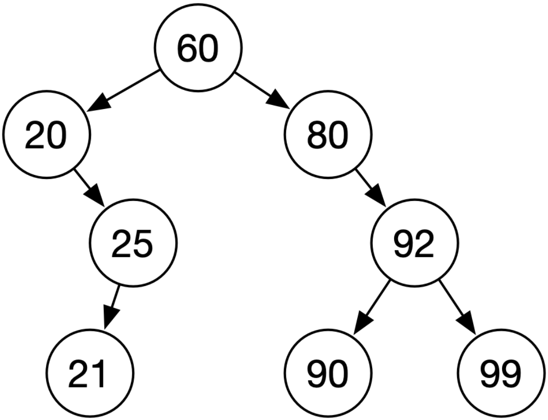
Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

Algoritmi e Strutture di Dati
Anno Accademico 2022/2023

Esercizio 1

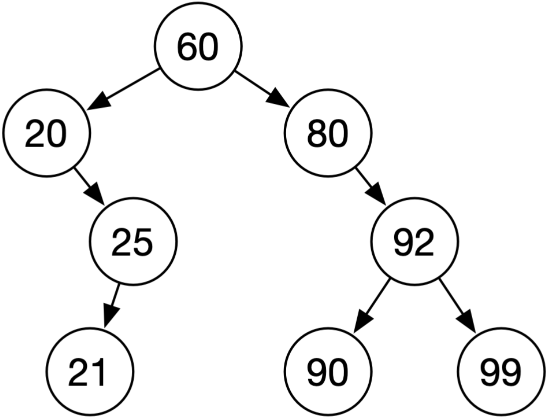
- Dato un Albero Binario di Ricerca con chiavi numeriche intere, inizialmente vuoto, disegnare l'albero ottenuto dopo l'inserimento in ordine dei seguenti valori: $60, 80, 20, 25, 92, 21, 99, 90$

Esercizio 1 - Soluzione

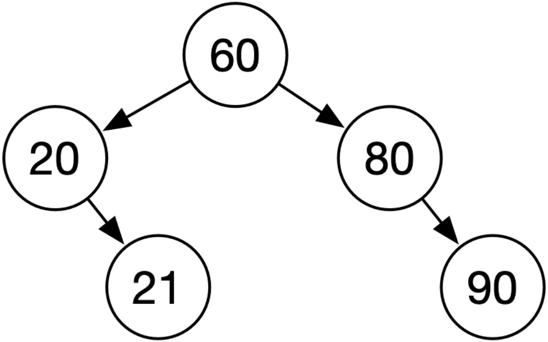


Esercizio 2

- Cancellare dall'albero ottenuto nell'esercizio 1 (mostrato sotto) i seguenti nodi in ordine: $92, 25, 99$

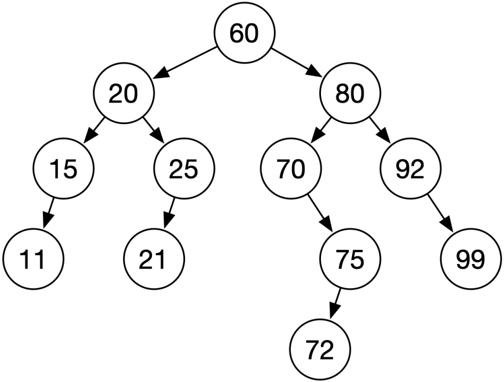


Esercizio 2 - Soluzione

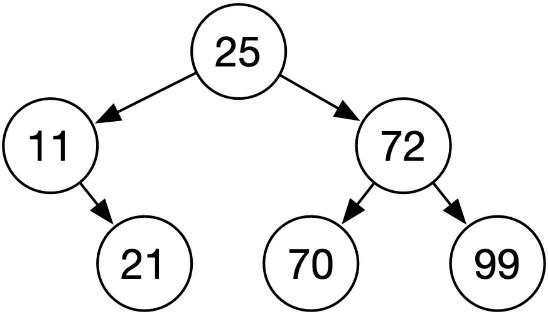


Esercizio 3

- Cancellare dall'albero mostrato sotto i seguenti nodi in ordine: $80, 15, 20, 75, 60, 92$



Esercizio 3 - Soluzione



Esercizio 4

- E' vero che se un nodo in un BST ha due figli, allora il suo successore non ha un figlio sinistro e il suo predecessore non ha un figlio destro?
- Giustificare la risposta

Esercizio 4 - Soluzione

- Assumiamo che u abbia due figli
- Il predecessore di u è il nodo v che viene visitato per ultimo in una visita in-ordine del sottoalbero sinistro di u ($u.left$)
  - Nel caso in cui le chiavi siano tutte distinte v coincide con il nodo con chiave massima nel sottoalbero sinistro di u
- Assumiamo che v sia il *nodo massimo* in $u.left$ e che abbia un figlio destro, allora in una visita in-ordine v non può essere l'ultimo nodo ad essere visitato e questo contraddice l'ipotesi che v sia il nodo massimo. Quindi v non può avere un figlio destro.

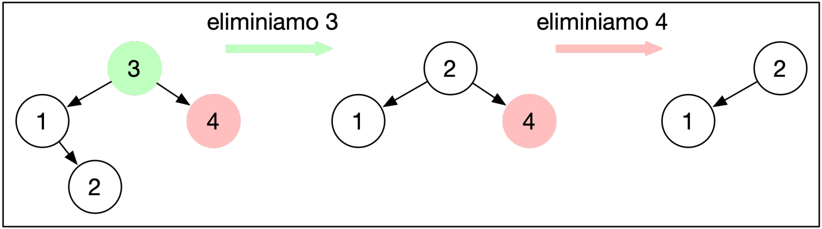
- Questa dimostrazione si applica simmetricamente al caso successore

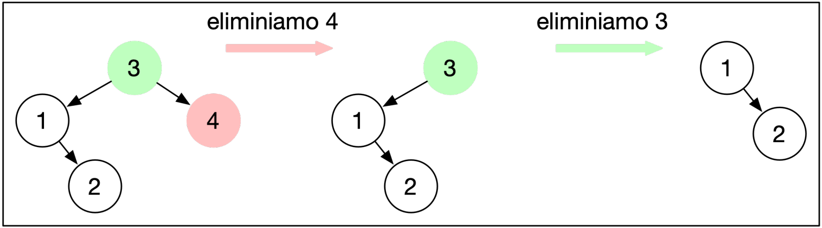
Esercizio 5

- L'operazione DELETE su BST è commutativa?
- Per esempio, dato un qualsiasi BST, eliminare prima un nodo u e poi un nodo v produce sempre lo stesso BST che otterremmo eliminando prima v e poi u?
- Giustificare la risposta

Esercizio 5 - Soluzione

- Mostriamo con un contro-esempio che l'operazione DELETE su un BST non è commutativa





Esercizio 6

- Dato un albero binario di ricerca, scrivere un algoritmo ricorsivo che stampi i valori delle chiavi in ordine decrescente

Esercizio 6 - Soluzione

```
function REV-INORDER(NODE T)
    if T ≠ NIL then
        REV-INORDER(T.right)
        PRINT(T.key)
        REV-INORDER(T.left)
```

Costo: $\Theta(n)$, $n =$ numero di nodi in T

Esercizio 7

- Scrivere un algoritmo possibilmente efficiente che dato in input l'albero binario di ricerca T e due valori interi a e b, con $a < b$, ritorni il numero di nodi la cui chiave appartiene all'intervallo $[a, b]$ (estremi inclusi)

Esercizio 7 - Soluzione

```
1: function COUNT(NODE T, INT a, INT b) → INT
2:     if T == NIL then
3:         return 0
4:     else if T.key < a then
5:         return COUNT(T.right, a, b)
6:     else if T.key > b then
7:         return COUNT(T.left, a, b)
8:     else
9:         return 1 + COUNT(T.left, a, T.key) + COUNT(T.right, T.key, b)
```

- Costo nel caso ottimo (nessuna chiave in $[a, b]$): $O(h)$, $h =$ altezza di T
- Costo nel caso pessimo (tutte le chiavi in $[a, b]$): $\Theta(n)$, $n =$ nodi in T
- Per quanto nel caso pessimo visitiamo comunque tutto l'albero, le condizioni a riga 4 e 6 ci evitano di visitare tutto l'albero

Esercizio 8

- Consideriamo la funzione

$$\text{BSTBuild}(\text{Array } A[1, \cdots, n]) \rightarrow \text{BST}$$

  che costruisce un Albero Binario di Ricerca da un array di interi

- Dimostrare che qualsiasi implementazione di BSTBuild basata sul confronto ha costo pessimo $\Omega(n \log n)$
  - Non riusciamo a costruire l'albero con un costo migliore di $n \log n$

- Con *implementazione basata sul confronto* intendiamo che l'albero viene costruito con una qualche procedura che utilizza il confronto dei valori delle chiavi per generare l'albero

Esercizio 8 - Soluzione

- Possiamo inventare un nuovo algoritmo di ordinamento BSTSort nel seguente modo
  1. Costruiamo un BST con BSTBuild
  2. Otteniamo l'array ordinato con una in-visita su tale array
- Il costo nel caso pessimo di BSTSort è $\Omega(n + f(n))$ dove
  - n è il costo di una visita inorder
  - $f(n)$ è il costo di BSTBuild (basato sul confronto, per ipotesi)
- BSTSort è un algoritmo di ordinamento basato sul confronto
- Questo implica che il suo costo nel caso pessimo è $\Omega(n \log n)$
- Deve quindi essere $\Omega(n + f(n)) = \Omega(n \log n)$ che necessariamente implica $f(n) = \Omega(n \log n)$