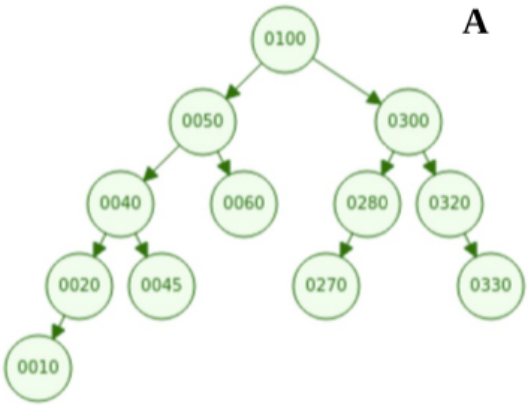
Soluzione BST

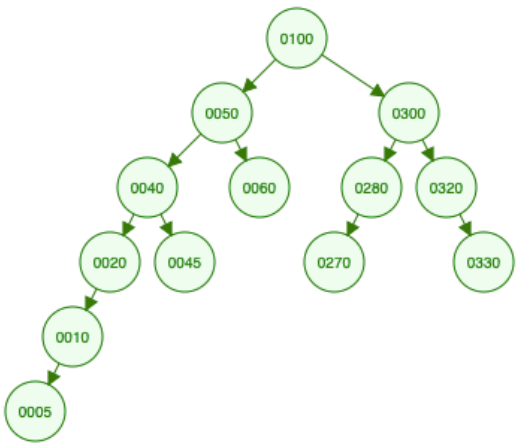
InsertElem

> Si consideri il seguente BST che indicheremo con **A**.
>
> 
>
>
> **[1/3 del punteggio]** Assumendo che le chiavi siano numeri interi, si disegni come viene modificato il BST **A** dopo la seguente chiamata (senza fornire alcuna spiegazione dei passaggi: disegnate solo il risultato): `insertElem(5, “elem”, A);`
>
> La chiamata insertElem(5, “elem”, A) rappresenta il caso peggiore della operazione insertElem sui BST, rispetto alla complessità temporale? Motivare la risposta.

Il nodo formato dall'elemento 5, viene **confrontato** dapprima con la radice, poi a seconda se il valore è Maggiore o Minore, si considera il figlio destro o sinistro. Nel nostro caso vengono effettuati i seguenti controlli:

- 5 < 100 ? --> controllo Left Tree
- 5 < 50 ? --> controllo Left Tree
- 5 < 40 ? --> controllo Left Tree
- 5 < 20 ? --> controllo Left Tree
- 5 <10 ? si --> il nodo 10 non ha figli quindi **aggiungo il nodo 5** come figlio del nodo 10

Ed otteniamo il seguente BST:



La funzione di `insert` nel caso **peggiore** ha complessità: $O(n)$ (n numero elementi). Ricadiamo in questo caso solamente se l'albero è molto sbilanciato (ha solo figli dx o solo figli sx). Nel nostro caso, pur dovendo scorrere tutto un lato del BST, ricadiamo nel caso **average** che vale $O(log\ n)$

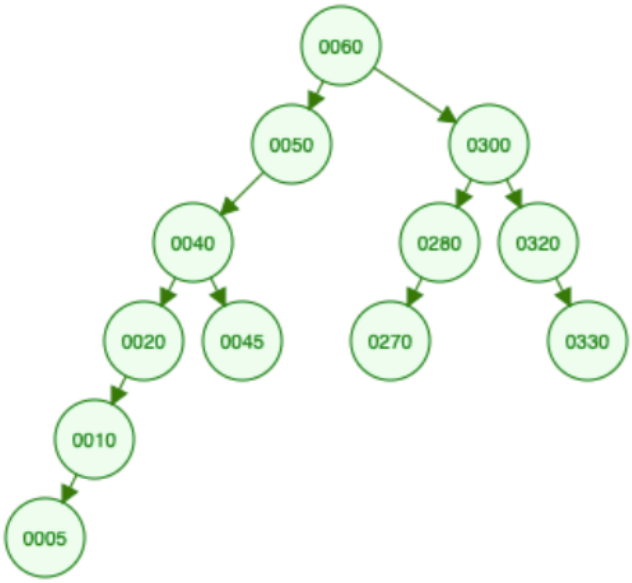
DeleteElem

> **[2/3 del punteggio]** Si spieghino dettagliatamente, possibilmente mediante disegni chiari e autoesplicativi, i passaggi principali della chiamata `deleteElem(100, A);` effettuata sull'albero **A** modificato a seguito dell'inserimento dell'elemento con chiave 5.
>
> La chiamata deleteElem(100, A) rappresenta il caso peggiore della operazione deleteElem sui BST, rispetto alla complessità temporale? Motivare la risposta.

Dato che il nodo da cancellare (100) possiede **entrambi i figli**, eseguo i seguenti passi:

1. la sostituzione del valore presente nel nodo da cancellare, con il valore minimo del sottoalbero destro, o con il massimo nel sottoalbero sinistro ;
2. la cancellazione del valore minimo dal sottoalbero destro (o del max dal sottoalbero sx).

In questo caso sceglieremo il **valore massimo nel sottoalbero sinistro**, ossia **60**. Si copia il contenuto del nodo al posto dell'elemento da cancellare (copio 60 al posto di 100), quindi 60 diventa la **nuova radice** dell'albero, e si elimina il vecchio nodo (elimino old 60).
Otteniamo quindi il seguente albero:



La funzione di `deleteElem` nel caso **peggiore** ha complessità: $O(n)$ (n numero elementi). Nel nostro caso, però, l'albero non è sbilanciato e anzi dobbiamo scorrere solo 2 elementi. Ricadiamo quindi nell'avg time che vale $O(log\ n)$

!! A pagina 18-19 degli appunti ci sono altri esempi