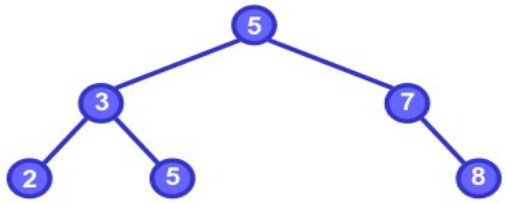
# ALBERI BINARI DI RICERCA

Sono strutture dati utilizzate per implementare dizionari (insieme di elementi che hanno una chiave che definiscono le relazioni d'ordine); ci consentono di ricercare un elemento in tempo minore. Si possono effettuare operazioni di consultazione (ricerca di un elemento) e modifica (inserimento e cancellazione). Sono realizzati mediante alberi: ogni nodo è un elemento caratterizzato dai campi parent (nodo genitore), left (figlio di sinistra), right (figlio di destra), key (dati satellite).

## Proprietà: verifica con un visita in preordine

Per ogni nodo (x) dell'albero, tutti i figli di sinistra sono elementi minori uguali a x, quelli de destra maggiori di x. Per verificare la proprietà occorre verificare che tutti i valori del sottoalbero di sinistra siano minori uguali a x.key, e che tutti i valori del sottoalbero di destra siamo maggiori o uguali a x.key.




## funzioni NO-MAGGIORE e NO-MINORE

```
NO-MAGGIORE(x,v)      ▷ x è la radice del sottoalbero
1. if x == NIL
2.        return TRUE
3. else   return (  (x.key <= v) and
4.                  NO-MAGGIORE(x.left,v) and
5.                  NO-MAGGIORE(x.right,v) )
```

```
NO-MINORE(x,v)        ▷ x è la radice del sottoalbero
1. if x == NIL
2.        return TRUE
3. else   return (  (x.key >= v) and
4.                  NO-MINORE(x.left,v) and
5.                  NO-MINORE(x.right,v) )
```

## Analisi della complessità

L'esplorazione di un albero con n nodi costa Θ anche se non sappiamo come siano distribuiti gli n nodi dei sottoalberi sinistro e destro.

Nel caso migliore, un albero binario di ricerca bilanciato costa: $T(n) = a * T(n/b) + p(n^k)$; per $a = b^k$ vale $T(n) = \Theta(n \log n)$.

Nel caso peggiore, un abr costa: $T(n) = T(n-1) \Theta(n)$; la complessità è: $T(n) = \Theta(n^2)$.

## Proprietà: verifica con una visita in postordine

Controllo (boolean) se è un abr attraverso il calcolo del massimo nel sottoalbero di sinistra e del minimo nel sottoalbero di destra. La ricorsione è un esempio di verifica in postordine con complessità Θ(n).

### visita in postordine ABR-POST

```
ABR-POST(x)   ▷ restituisce un oggetto (is_abr, min, max)
1.  if (x == NIL) return NIL
2.  l = ABR-POST(x.left)    ▷ l ha i campi is_abr, min, max
3.  r = ABR-POST(x.right)   ▷ r ha i campi is_abr, min, max
4.  if (l == NIL and r == NIL)     ▷ x è una foglia
5.      return (TRUE, x.key, x.key)
6.  if (l == NIL and r != NIL)     ▷ x ha il figlio destro
7.      out = r.is_abr and (x.key <= r.min)
8.      return (out, x.key, r.max)
9.  if (l != NIL and r == NIL)     ▷ x ha il figlio sinistro
10.     out = l.is_abr and (x.key >= l.max)
11.     return (out, l.min, x.key)
12. out = l.is_abr and r.is_abr    ▷ x ha entrambi i figli
13. out = out and (x.key <= r.min) and (x.key >= l.max)
14. return (out, l.min, r.max)
```

## Proprietà: verifica con una visita simmetrica

I valori vengono visitati in ordine crescente: figlio sinistro, padre e figlio destro. La strategia che si segue