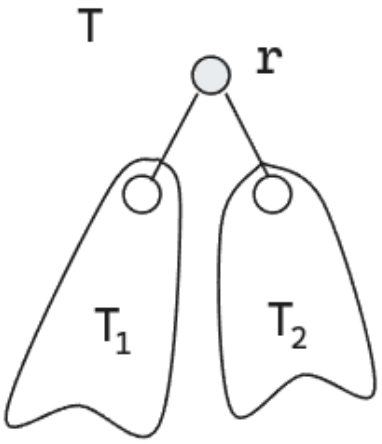
Lezione_12_DeA

4.2.2.1. Dimostrazioni

4.2.2.1.1. (1)

Proviamo la prima proprietà per induzione sull'altezza $h \geq 0$ di T.
Caso base: $h = 0 \implies T$ equivale a un singolo nodo $\implies n = 1$ e $m = 1 \implies \checkmark$.
Passo induttivo: Fisso $h \geq 0$ arbitrario.
Ipotesi induttiva: la proprietà (1) vale per tutti gli alberi binari propri di altezza $\leq h$.

Sia T un albero binario proprio di altezza $h + 1 \geq 1$, sia T_1 di altezza h_1 il sottoalbero con radice il figlio sinistro della radice di T, mentre T_2 di altezza h_2 quello del figlio destro.



Allora $h + 1 = 1 + \max\{h_1, h_2\} \implies h_1, h_2 \leq h \implies$ Per T_1 e T_2 vale l'ipotesi induttiva.

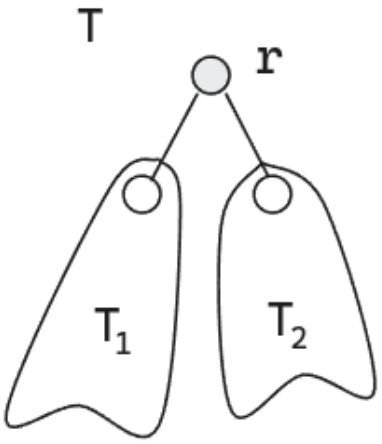
Sia m il numero di foglie di T, n il numero di nodi di T, m_i il numero di foglie di T_i, n_i il numero di nodi di T_i, $i = \{1, 2\}$.
$m = m_1 + m_2$, $n = n_1 + n_2 + 1$.
$m = m_1 + m_2 = (n_1 - m_1 + 1) + (n_2 + m_2 + 1) =$ per ipotesi induttiva:
$= (n_1 + n_2 + 1) - (m_1 + m_2) + 1 = n - m + 1$.
□

> **Osservazione**
>
> In un albero binario proprio T, dato che il numero di foglie è uguale al numero di nodi interni aumentato di uno, il numero totale di nodi è dispari.

4.2.2.1.2. (2)

Dimostriamo che $m \leq 2^h$ per induzione su $h \geq 0$.
Caso base: $h = 0 \implies T$ equivale a un singolo nodo $\implies m = 1 \implies \checkmark$.
Passo induttivo: Fisso $h \geq 0$ arbitrario.
Ipotesi induttiva: la proprietà vale per tutti gli alberi binari propri di altezza $\leq h$.

Sia T un albero binario proprio di altezza $h+1 \geq 1$, sia T_1 di altezza h_1 il sottoalbero con radice il figlio sinistro della radice di T, mentre T_2 di altezza h_2 quello del figlio destro.



Allora $h+1 = 1 + \max\{h_1, h_2\} \implies h_1, h_2 \leq h \implies$ Per T_1 e T_2 vale l'ipotesi induttiva.

Sia m il numero di foglie di T, m_i il numero di foglie di T_i, $i = \{1, 2\}$.
$m = m_1 + m_2$ e devo dimostrare che $m \leq 2^{h+1}$.
$m = m_1 + m_2 \leq 2^{h_1} + 2^{h_2} \leq$ per ipotesi induttiva:
$\leq 2^h + 2^h = 2^{h+1}$.
□

Dimostro che $m \geq h+1$ per induzione su $h \geq 0$.
Caso base: $h = 0 \implies T$ equivale a un singolo nodo $\implies m = 1 \implies \checkmark$.
Passo induttivo: Fisso $h \geq 0$ arbitrario.
Ipotesi induttiva: la proprietà vale per tutti gli alberi binari propri di altezza $\leq h$.

In maniera analoga a prima, la proprietà vale per T_1 e T_2.
$m = m_1 + m_2 \geq (h_1 + 1) + (h_2 + 1) \geq$ per ipotesi induttiva:
$\geq \max\{h_1, h_2\} + 2 = (h+1) + 1$.
□

4.2.2.1.3. (3), (4), (5)

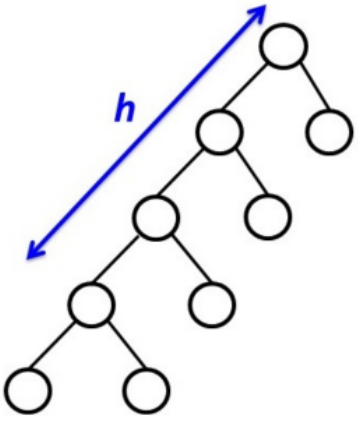
Per provare $(3): h \leq n-m \leq 2^h-1$, sostituisco m in (2) con $n-m+1$ (da (1)) e ottengo $h+1 \leq n-m+1 \leq 2^h \implies h \leq n-m \leq 2^h-1$.
□

Per provare $(4): 2h+1 \leq n \leq 2^{h+1}-1$ sommo (2) e (3).
□

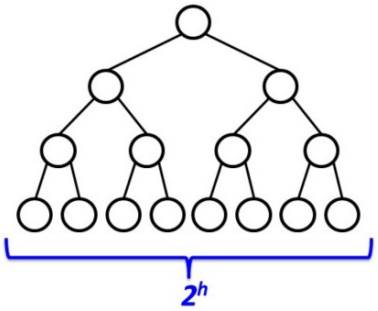
(5) deriva da (4), risolvendo rispetto a h:
$2h+1 \leq n \implies h \leq \frac{n-1}{2}$
$2^{h+1}-1 \geq n \implies 2^{h+1} \geq n+1 \implies h+1 \geq \log_2(n+1) \implies h \geq \log_2(n+1)-1$.
□

> **Osservazione**
>
> La quinta proprietà, (5), implica che in un albero binario proprio con n nodi, l'altezza è compresa tra $\Omega(\log n)$ e $O(n)$.

4.2.3. Alberi binari propri estremi



In questo esempio si ha $m = h+1$ e $n = 2h+1$. È importante notare come le parti sinistre di (2), (3) e (4) e la parte destra di (5) continuano a valere.
È un albero *molto sbilanciato* (*skewed*).



In questo esempio di hanno 2^i nodi al livello i, $0 \leq i \leq h$, quindi ci saranno $m = 2^h$ (equivalente al numero di nodi al livello h), quindi

$n = \sum_{i=0}^{h} 2^i = \frac{2^{h+1}-1}{2-1} = 2^{h+1} - 1$. È importante notare come le parti destre di (2), (3) e (4) e la parte sinistra di (5) continuano a valere.
È un albero *perfettamente bilanciato*.

> **⚠ Attenzione**
>
> In assenza di altre ipotesi, il migliore upper bound all'altezza di un albero binario con n nodi è $O(n)$, non $O(\log n)$.

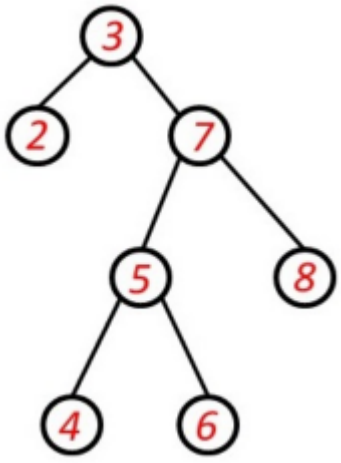
4.2.4. Visite di alberi binari

> Oltre alle visite in `preorder` e `postorder`, per gli alberi binari si definisce anche la *visita inorder*, che visita *prima (ricorsivamente) il sottoalbero sinistro, poi il padre, poi (ricorsivamente) il sottoalbero destro*.

Algoritmo `inorder(v)`
Input: $v \in T$.
Output: visita inorder di T_v

```
if(T.left(v) != null) then{
     inorder(T.left(v));
}
visita v;
if(T.right(v) != null) then{
     inorder(T.right(v));
}
```

La *chiamata iniziale* per visitare tutto T è `inorder(T.root())`.



> **ⓘ Osservazione**

È un esempio di albero binario di ricerca, che studieremo più avanti, dove la visita `inorder` tocca gli elementi presente in ordine crescente di valore.

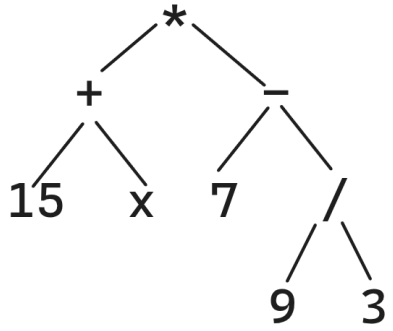
4.2.4.1. Complessità di `inorder(T.root())`

La complessità è $\Theta\left(n + \sum_{v \in T} t_v\right)$, dove $n = |T|$ e t_v è il costo della visita di v.
Si fa la stessa analisi di `preorder` per dimostrare la complessità.

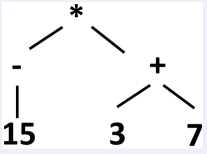
4.2.4.2. Applicazioni: espressioni aritmetiche

> Il *Parse Tree* T associato a un'espressione aritmetica E (con operatori solo binari) è un albero binario proprio i cui nodi foglia contengono le costanti/variabili di E e i nodi interni contengono gli operatori di E, in modo tale che:
>
> - Se $E = a$, con a una costante/variabile, allora T è costituito da un'unica foglia contenente a;
> - Se $E = (E_1 \operatorname{Op} E_2)$, la radice di T contiene Op e ha come sottoalbero sinistro il Parse Tree associato a E_1, mentre come sottoalbero destro il Parse Tree associato a E_2.

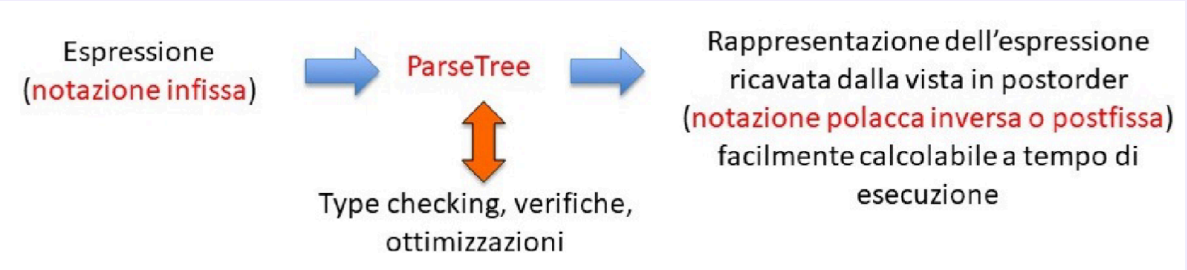
Ad esempio, posso scrivere un'espressione con la *notazione infissa*: $E = ((15 + x) * (7 - (9 \div 3)))$; altrimenti posso usare la *notazione postfissa* o *polacca inversa*: $E = 15x + 793 \div -*$. Il Parse Tree per quest'espressione E è



Osservazioni

- Le parentesi sono implicite nella struttura dell'albero;
- Se si ammettono operatori unari, l'albero non è più proprio, ad esempio: $-15 * (3 + 7)$ 

- Nei compilatori avviene il seguente processo:



Nelle sezioni successive vedremo come a partire dal Parse Tree T di un'espressione E si possono generare le rappresentazioni di E in notazione postfissa e infissa, usando le visite rispettivamente in postorder e inorder.

> Sia E_v l'espressione associata al sottoalbero T_v, con v nodo di T.

4.2.4.2.1. Generazione dell'espressione in notazione infissa

Si utilizza lo schema della visita inorder.

Algoritmo `infix(T,v,L)`
Input: Parse Tree T for E, $v \in T$, Lista L.
Output: Aggiunta a L di E_v in notazione infissa.

```
if(T.isExternal(v)) then{
     L.addLast(v.getElement());
}
else{
     L.addLast('(');
     infix(T, T.left(v), L);
     L.addLast(v.getElement());
     infix(T, T.right(v), L);
     L.addLast(')');
}
```

La *chiamata iniziale* per esprimere in notazione infissa tutta l'espressione E è `infix(T, T.root(), L)`, con $L = \emptyset$.

> **Osservazione**
>
> T e L sono variabili globali.

4.2.4.2.1.1. Complessità di `infix(T, T.root(), L)`

`infix(T, T.root(), L)` ha la stessa struttura della visita inorder, quindi la complessità è $\Theta\left(n + \sum_{v \in T} t_v\right)$, dove $n = |T|$ e t_v è il csto della visita di v, che in questo caso è $\Theta(1)$, quindi la complessità è $\Theta(n)$.

4.2.4.2.2. Generazione dell'espressione in notazione postfissa

Si utilizza lo schema della visita in postorder.

Algoritmo `ipostix(T,v,L)`
Input: Parse Tree T for E, $v \in T$, Lista L.
Output: Aggiunta a L di E_v in notazione postfissa.

```
if(T.isExternal(v)) then{
     L.addLast(v.getElement()); //visita di v ∈ Θ(1)
}
else{
     postfix(T, T.left(v), L);
     postfix(T, T.right(v), L);
     L.addLast(v.getElement()); //visita di v ∈ Θ(1)
}
```

La *chiamata iniziale* per esprimere in notazione postfissa tutta l'espressione E è `postfix(T, T.root(), L)`, con $L = \emptyset$.

4.2.4.2.2.1. Complessità di `postfix(T, T.root(), L)`

L'analisi è analoga a `infix`, quindi la complessità è $\Theta(n)$, con $n = |T|$.

Riepilogo alberi binari

- Definizioni: albero binario proprio;
- Visita inorder e le loro applicazioni;
- Algoritmi di visita come template generali.