

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

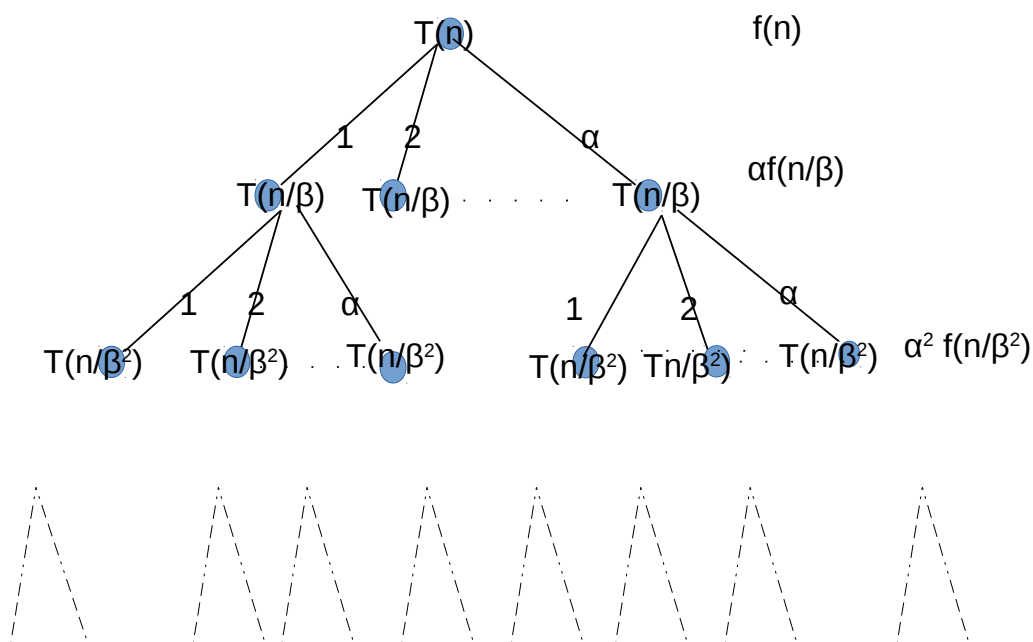
- Consideriamo la seguente relazione di ricorrenza:

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + f(n) & \text{altrimenti} \end{cases}$$

con $\alpha \geq 1$ e $\beta > 1$ costanti.

- Nel caso in cui $T(n)$ sia la funzione che scaturisce dall'analisi di un algoritmo basato sul paradigma del Divide et Impera, $f(n)$ è il tempo per il lavoro di suddivisione e di ricombinazione. In altre parole, $f(n) = d(n) + r(n)$.
- In realtà nella ricorrenza n/β dovrebbe essere $\lceil n/\beta \rceil$ oppure $\lfloor n/\beta \rfloor$. Per stimare $T(n)$, assumiamo per semplicità che n sia una potenza di β in modo da poter omettere le parti intere superiori o inferiori.

SOLUZIONE DELLE RELAZIONI DI RICORRENZA



Sia h l'altezza dell'albero ($h+1$ livelli). Per $i < h$, Il tempo di esecuzione per tutte le chiamate ricorsive a livello i è al più $\alpha^i f(n/\beta^i)$.

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

- L'algoritmo non effettua chiamate ricorsive quando l'input ha dimensione al più c . Quindi la ricorrenza non sarà più applicata quando si arriva al livello i per cui per la prima volta $n/\beta^i \leq c$, cioè $i = \lceil \log_\beta n/c \rceil$.
- Il numero di livelli dell'albero è quindi $\lceil \log_\beta n/c \rceil + 1$ (partiamo dal livello 0) e ciascun nodo sul livello $\lceil \log_\beta n/c \rceil$ corrisponde al tempo $T(n/\beta^{\lceil \log_\beta n/c \rceil}) \leq T(c) \leq c_0$. Il tempo totale per eseguire le $\alpha^{\lceil \log_\beta n/c \rceil}$ chiamate ricorsive in quest'ultimo livello è quindi $\leq \alpha^{\lceil \log_\beta n/c \rceil} c_0$.
- Abbiamo visto che per $i < \lceil \log_\beta n/c \rceil$, il tempo per eseguire tutte le chiamate sul livello i è $\alpha^i f(n/\beta^i)$.
- Sommando su tutti i livelli (compreso l'ultimo) si ha

$$T(n) \leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} \alpha^i f(n/\beta^i).$$

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

- Vogliamo stimare la funzione

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + f(n) & \text{altrimenti} \end{cases}$$

quando la funzione $f(n)$ è limitata da $c' n^k$, dove c' e k sono due costanti tali che $k \geq 0$, $c' > 0$ ($f(n)$ polinomiale).

- Da quanto ottenuto nella slide precedente si ha che

$$\begin{aligned} T(n) &\leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} \alpha^i f(n/\beta^i) \\ &\leq \alpha^{\lceil \log_\beta n/c \rceil} c_0 + \sum_{i=0}^{\lceil \log_\beta n/c \rceil - 1} \alpha^i c' (n/\beta^i)^k \end{aligned}$$

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

- Abbiamo visto che se $f(n) \leq c' n^k$, dove c' e k sono due costanti tali che $k \geq 0$, $c' > 0$, allora

$$T(n) \leq \alpha^{\lceil \log_{\beta} n/c \rceil} c_0 + c' n^k \sum_{i=0}^{\lceil \log_{\beta} n/c \rceil - 1} (\alpha/\beta^k)^i.$$

- consideriamo i 2 seguenti casi:
- $\alpha = \beta^k$: In questo caso si ha

$$\alpha^{\lceil \log_{\beta} n/c \rceil} c_0 = (\beta^k)^{\lceil \log_{\beta} n/c \rceil} c_0 < (\beta^k)^{\log_{\beta}(n/c)+1} c_0 = \beta^k (n/c)^k c_0 = O(n^k)$$

e

$$c' n^k \sum_{i=0}^{\lceil \log_{\beta} n/c \rceil - 1} (\alpha/\beta^k)^i = c' n^k \sum_{i=0}^{\lceil \log_{\beta} n/c \rceil - 1} 1 = c' n^k \lceil \log_{\beta} n/c \rceil = O(n^k \log_{\beta} n).$$

$$\text{Quindi } T(n) = O(n^k) + O(n^k \log_{\beta} n) = O(n^k \log_{\beta} n) = O(n^k \log n).$$

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

- $\alpha \neq \beta^k$: In questo caso si ha

$$\begin{aligned} \alpha^{\lceil \log_{\beta} n/c \rceil} c_0 &< c_0 \alpha^{\log_{\beta} n/c + 1} = c_0 \alpha \cdot \alpha^{\log_{\beta} n/c} = c_0 \alpha \cdot \alpha^{\log_{\alpha}(n/c) \log_{\beta} \alpha} \\ &= c_0 \alpha (n/c)^{\log_{\beta} \alpha} = O(n^{\log_{\beta} \alpha}), \end{aligned} \quad (1)$$

e

$$c' n^k \sum_{i=0}^{\lceil \log_{\beta} n/c \rceil - 1} (\alpha/\beta^k)^i = c' n^k \cdot \frac{(\alpha/\beta^k)^{\lceil \log_{\beta} n/c \rceil} - 1}{(\alpha/\beta^k) - 1}. \quad (2)$$

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

Consideriamo i due sottocasi di $\alpha \neq \beta^k$: $\alpha < \beta^k$ e $\alpha > \beta^k$

- Caso $\alpha < \beta^k$:

$$\begin{aligned} \frac{(\alpha/\beta^k)^{\lceil \log_\beta(n/c) \rceil} - 1}{(\alpha/\beta^k) - 1} &= \frac{1 - (\alpha/\beta^k)^{\lceil \log_\beta n/c \rceil}}{1 - (\alpha/\beta^k)} \\ &< \frac{1}{1 - (\alpha/\beta^k)} = \frac{\beta^k}{\beta^k - \alpha} = O(1). \end{aligned}$$

Si ha quindi che la suddetta relazione insieme alla (1) e alla (2) della slide precedente implicano:

$$T(n) \leq O(n^{\log_\beta \alpha}) + c' n^k O(1) = O(n^{\log_\beta \alpha} + n^k).$$

Si noti che $\alpha < \beta^k$ implica $\log_\beta \alpha < k$ e di conseguenza si ha

$$T(n) = O(n^{\log_\beta \alpha} + n^k) = O(n^k).$$

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

- Caso $\alpha > \beta^k$:

$$\begin{aligned} \frac{(\alpha/\beta^k)^{\lceil \log_\beta(n/c) \rceil} - 1}{(\alpha/\beta^k) - 1} &< \frac{(\alpha/\beta^k)^{\log_\beta(n/c)+1} - 1}{(\alpha/\beta^k) - 1} = \frac{(\alpha/\beta^k)(\alpha/\beta^k)^{\log_\beta(n/c)} - 1}{(\alpha/\beta^k) - 1} \\ &= O((\alpha/\beta^k)^{\log_\beta(n/c)}) = O((\alpha/\beta^k)^{\log_{\alpha/\beta^k}(n/c) \log_\beta(\alpha/\beta^k)}) \\ &= O((n/c)^{\log_\beta(\alpha/\beta^k)}) = O((n/c)^{\log_\beta \alpha - \log_\beta \beta^k}) \\ &= O(n^{\log_\beta(\alpha) - k}) \end{aligned}$$

Si ha quindi che la suddetta relazione insieme alla (1) e alla (2) della slide precedente implicano:

$$T(n) \leq O(n^{\log_\beta \alpha}) + c' n^k O(n^{\log_\beta(\alpha) - k}) = O(n^{\log_\beta \alpha} + n^k n^{\log_\beta(\alpha) - k}) = O(n^{\log_\beta \alpha}).$$

SOLUZIONE DELLE RELAZIONI DI RICORRENZA

- Abbiamo stimato la funzione

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + c' n^k & \text{altrimenti,} \end{cases}$$

dove c' e k sono due costanti tali che $k \geq 0$, $c' > 0$.

- Abbiamo provato

$$T(n) = \begin{cases} O(n^k) & \text{se } \alpha < \beta^k \\ O(n^k \log n) & \text{se } \alpha = \beta^k \\ O(n^{\log_\beta \alpha}) & \text{se } \alpha > \beta^k \end{cases}$$

- **Esempi:** Nel caso di MergeSort $\alpha = 2$, $\beta = 2$ e $k = 1$. Si ha $\alpha = \beta^k$ e quindi $T(n) = O(n^k \log n) = O(n \log n)$.
Nel caso dell'algoritmo per la ricerca binaria $\alpha = 1$, $\beta = 2$ e $k = 0$. Si ha $\alpha = \beta^k$ e quindi $T(n) = O(n^k \log n) = O(\log n)$.

SOLUZIONE DELLE RELAZIONI DI RICORRENZA QUANDO n NON È POTENZA DI β

- Consideriamo la seguente relazione di ricorrenza:

$$T(n) = \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(n/\beta) + c' n^k & \text{altrimenti} \end{cases}$$

con $\alpha \geq 1$ e $\beta > 1$ costanti.

- Quando n non è una potenza di β la taglia di ciascun sottoproblema è $\lceil n/\beta \rceil$ oppure $\lfloor n/\beta \rfloor$.
- Siccome vogliamo stabilire un limite superiore per $T(n)$ mettiamoci nel caso peggiore in cui la taglia di ciascun sottoproblema è $\lceil n/\beta \rceil$.
Consideriamo quindi la seguente relazione di ricorrenza:

$$T(n) = \begin{cases} c_0 & \text{se } n \leq c \\ \alpha T(\lceil n/\beta \rceil) + c' n^k & \text{altrimenti} \end{cases}$$

con $\alpha \geq 1$, $\beta > 1$ e $k \geq 0$ costanti.

- Usando questa nuova relazione di ricorrenza potremmo usare un procedimento simile a quello usato per il caso in cui n è potenza di β per provare le stesse limitazioni superiori viste per quel caso. Nel seguito invece useremo un argomento molto semplice per dedurre che quelle limitazioni valgono anche quando n non è potenza di β .

SOLUZIONE DELLE RELAZIONI DI RICORRENZA QUANDO n NON È POTENZA DI β

- Sia p il più piccolo intero positivo per cui $n \leq \beta^p$, cioè p è l'intero per cui $\beta^{p-1} < n \leq \beta^p$.
- Osserviamo che siccome $T(n)$ è una funzione non decrescente allora $T(n) \leq T(\beta^p)$.
- Applicando a $T(\beta^p)$ la limitazione asintotica dimostrata per le potenze di β si ha

$$T(\beta^p) = \begin{cases} O((\beta^p)^k) & \text{se } \alpha < \beta^k \\ O((\beta^p)^k \log(\beta^p)) & \text{se } \alpha = \beta^k \\ O((\beta^p)^{\log_\beta \alpha}) & \text{se } \alpha > \beta^k \end{cases} \quad (3)$$

- Osserviamo che $\beta^p = \beta\beta^{p-1} < \beta n$, dal momento che $\beta^{p-1} < n$. Si ha quindi che

$$\begin{aligned} O((\beta^p)^k) &= O((\beta n)^k) = O(\beta^k n^k) = O(n^k), \\ O((\beta^p)^k \log(\beta^p)) &= O((\beta n)^k \log(\beta n)) = O(n^k (\log(\beta) + \log n)) = O(n^k \log n), \\ O((\beta^p)^{\log_\beta \alpha}) &= O((\beta n)^{\log_\beta \alpha}) = O(\beta^{\log_\beta \alpha} n^{\log_\beta \alpha}) = O(n^{\log_\beta \alpha}). \end{aligned}$$

La (3) può essere quindi scritta come segue

$$T(\beta^p) = \begin{cases} O(n^k) & \text{se } \alpha < \beta^k \\ O(n^k \log n) & \text{se } \alpha = \beta^k \\ O(n^{\log_\beta \alpha}) & \text{se } \alpha > \beta^k \end{cases}$$

- Poichè $T(n) \leq T(\beta^p)$ allora le limitazioni appena provate valgono anche per $T(n)$.

ESEMPI DI RELAZIONI DI RICORRENZA DELLA FORMA

$$T(n) \leq \alpha T(n/\beta) + cn^k$$

- Ricerca binaria

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \text{ oppure } k \text{ è l'elemento centrale} \\ T(n/2) + c & \text{altrimenti} \end{cases}$$

Si ha $\alpha = 1, \beta = 2, k = 0$.

Siccome $\alpha = \beta^k$, siamo nel secondo caso e si ha

$$T(n) = O(n^k \log n) = O(\log n).$$

ESEMPI DI RELAZIONI DI RICORRENZA DELLA FORMA

$$T(n) \leq \alpha T(n/\beta) + n^k$$

Nell'ordinamento per fusione,

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 2T(n/2) + cn & \text{altrimenti} \end{cases}$$

Quindi,

- $\alpha = 2$, $\beta = 2$ e $k = 1$
- siamo nel caso $\alpha = \beta^k$ e quindi $T(n) = O(n^k \log n) = O(n \log n)$.

ESEMPI DI RELAZIONI DI RICORRENZA DELLA FORMA

$$T(n) \leq \alpha T(n/\beta) + cn^k$$

- Moltiplicazione veloce di interi: primo algoritmo

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 4T(n/2) + cn & \text{altrimenti} \end{cases}$$

Applicazione del risultato provato:

- si ha che $\alpha = 4$, $\beta = 2$ e $k = 1$
- $\alpha > \beta^k$, quindi si applica il terzo caso e si ha $T(n) = O(n^{\log_2 4}) = O(n^2)$
- Moltiplicazione veloce di interi: secondo algoritmo

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 3T(n/2) + cn & \text{altrimenti} \end{cases}$$

Applicando il risultato dimostrato,

- si ha che $\alpha = 3$, $\beta = 2$ e $k = 1$
- $\alpha > \beta^k$, quindi si applica il terzo caso e si ha $T(n) = O(n^{\log_2 3}) = O(n^{1.585})$

SOMMATORIE UTILI

•

$$\sum_{i=1}^n i = n(n+1)/2$$

•

$$\sum_{i=1}^n i^2 = n(n+1)(2n+1)/6$$

•

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1} \text{ per } a \neq 1$$

•

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1 - a} \text{ per } 0 < a < 1.$$

DIVIDE ET IMPERA SU ALBERI

- **Caso base:** per $u = \text{null}$ o una foglia
- **Decomposizione:** riformula il problema per i sottoalberi radicati nei figli di u .
- **Ricombinazione:** ottieni il risultato con Ricombina

```
1 Decomponibile(u):
2   IF (u == null) {
3     RETURN valore base;
4   } ELSE {
5     i=0;
6     FOR( ciascun figlio f di u ){
7
8       risultatiFigli[i] = Decomponibile(f);
9       i=i+1 }
10    RETURN Ricombina(risultatiFigli);
11  }
```

La ricombinazione dei risultati delle chiamate ricorsive sui figli potrebbe essere effettuata anche nel for man mano che vengono ottenuti i risultati delle chiamate sui figli.

DIVIDE ET IMPERA SU ALBERI BINARI

- **Caso base:** per $u = \text{null}$ o una foglia
- **Decomposizione:** riformula il problema per i sottoalberi radicati nei figli $u.sx$ e $u.dx$
- **Ricombinazione:** ottieni il risultato con Ricombina

```
1 Decomponibile(u):
2   IF (u == null) {
3     RETURN valore base;
4   } ELSE {
5     risultatoSX = Decomponibile(u.sx);
6     risultatoDx = Decomponibile(u.dx);
7     RETURN Ricombina(risultatoSX, risultatoDx);
8   }
```

Analisi dell'algoritmo Decomponibile

- Assumiamo che il tempo per la decomposizione e la ricombinazione sia costante
- Se escludiamo il tempo impiegato per le chiamate ricorsive, l'algoritmo impiega tempo $O(1 + c_v)$, dove c_v è il numero di figli di v
- Se cominciamo la visita dal nodo w , l'algoritmo viene invocato su tutti i discendenti di w

→ **Tempo totale** =
$$\sum_{v \in T_w} O(c_v + 1) = O(|T_w|)$$

- La visita di tutto l'albero richiede tempo $O(|T|)$
- Se l'albero ha n nodi la visita richiede tempo $T(n) = O(n)$

ANALISI DELL'ALGORITMO DECOMPONIBILE

- Nell'analisi precedente abbiamo usato il fatto che $\sum_{v \in T_w} c_v = |T_w| - 1$.
- È facile vedere che vale questa uguaglianza in quanto ogni nodo di T_w , eccezion fatta per la radice w , è figlio di un unico nodo v dell'albero T_w e quindi viene contato esattamente una volta in quella sommatoria.

ANALISI DELL'ALGORITMO DECOMPONIBILE PER UN ALBERO BINARI MEDIANTE RELAZIONE DI RICORRENZA

La funzione $T(n)$ che esprime il tempo di esecuzione dell'algoritmo Decomponibile su un albero binario con n nodi può essere descritta dalla seguente relazione di ricorrenza, dove $r - 1 \geq 0$ è il numero di nodi del sottoalbero sinistro.

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ T(r-1) + T(n-r) + c & \text{altrimenti} \end{cases}$$

Dimostriamo per induzione che $T(n) \leq c'n$ per $n \geq 1$ e per una costante $c' > 0$. In altre parole $T(n) = O(n)$.

- Base: $T(1) \leq c_0$ implica $T(1) \leq c'$ se si sceglie $c' \geq c_0$.
- Passo induttivo: Assumiamo $T(m) \leq c'm$ per ogni $1 \leq m < n$ e dimostriamo $T(n) \leq c'n$.

Applichiamo relazione di ricorrenza: $T(n) \leq T(r-1) + T(n-r) + c$.
L'ipotesi induttiva implica $T(r-1) \leq c'(r-1)$ e $T(n-r) \leq c'(n-r)$.
Si ha quindi $T(n) \leq c'(r-1) + c'(n-r) + c = c'n - c'r + c$.

Affinché risulti $T(n) \leq c'n$ basta scegliere c' in modo che $c'r \geq c$ cioè $c' \geq c/r$. Non sappiamo quanto vale r ma sappiamo che $r \geq 1$ per cui basta scegliere $c' \geq c$.

- Dalla base dell'induzione e dal passo induttivo, sappiamo che basta scegliere $c' = \max\{c_0, c\}$ affinché valga $T(n) \leq c'n$ per $n \geq 1$.

ALGORITMI RICORSIVI SU ALBERI: DIMENSIONE

Calcolo della dimensione $d =$ numero di nodi

- Caso base: albero vuoto $\Rightarrow d = 0$
- Caso induttivo: $d = 1 +$ dimensione del sottoalbero sinistro $+$ dimensione del sottoalbero destro

```
1 Dimensione( u ):
2   IF (u == null) {
3     RETURN 0;
4   } ELSE {
5     dimensioneSX = Dimensione( u.sx );
6     dimensioneDX = Dimensione( u.dx );
7     RETURN dimensioneSX + dimensioneDX + 1;
8   }
```

Se si vuole conoscere la dimensione di tutto l'albero, si invoca Dimensione con u uguale alla radice

ALGORITMI RICORSIVI SU ALBERI: ALTEZZA

Calcolo dell'altezza h di un nodo:

- caso base per null $\Rightarrow h = -1$
- passo induttivo: $h = 1 +$ massima altezza dei figli

```
1 Altezza( u ):
2   IF (u == null) {
3     RETURN -1;
4   } ELSE {
5     altezzaSX = Altezza( u.sx );
6     altezzaDX = Altezza( u.dx );
7     RETURN max( altezzaSX, altezzaDX ) + 1;
8   }
```

Per calcolare l'altezza dell'albero, si invoca Altezza con u uguale alla radice

VISITA DI UN ALBERO BINARIO: INORDER

- **simmetrica** (*inorder*):

```
1 Simmetrica( u ):
2   IF (u != null) {
3     Simmetrica( u.sx );
4     elabora(u);
5     Simmetrica( u.dx );
6   }
```

$O(n)$ tempo per n nodi

VISITA DI UN ALBERO BINARIO: PREORDER

- **anticipata** (*preorder*):

```
1 Anticipata( u ):
2   IF (u != null) {
3     elabora(u);
4     Anticipata( u.sx );
5     Anticipata( u.dx );
6   }
```

$O(n)$ tempo per n nodi

VISITA DI UN ALBERO BINARIO: POSTORDER

- **posticipata** (*postorder*):

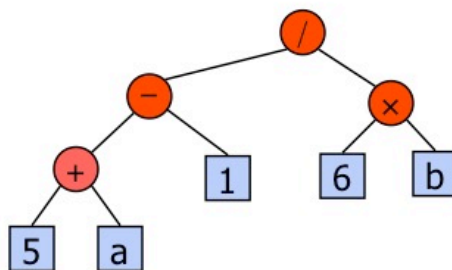
```
1 Posticipata( u ):
2   IF (u != null) {
3     Posticipata( u.sx );
4     Posticipata( u.dx );
5     elabora(u);
6   }
```

$O(n)$ tempo per n nodi

ESEMPIO DELL'USO DELLE VISITE

Esempio dell'uso delle visite: valutazione dell'espressione aritmetica rappresentata da un albero binario

- Albero binario associato ad una espressione:
 - Nodi interni: operatori
 - Nodi esterni: operandi
- Esempio: $((5 + a) - 1) / (6 \times b)$



USO DELLA VISITA POSTORDER PER VALUTARE L'ESPRESSIONE ARITMETICA RAPPRESENTATA DA UN ALBERO BINARIO

```
1 Valuta( u ):
2   IF (u==null) {
3     RETURN null;
4   }
5   IF (u.sx == null && u.dx==null) {
6     RETURN u.dato;
7   } ELSE {
8     valSinistra=Valuta( u.sx );
9     valDestra= Valuta( u.dx );
10    ris= Calcola(u.dato,valSinistra ,valDestra);
11    RETURN ris;
12  }
```

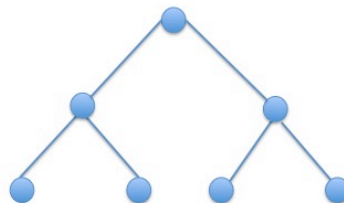
- La funzione *Calcola* invocata su *u.dato*, *valSinistra* e *valDestra*, applica l'operatore memorizzato nel nodo interno *u* ai valori *valSinistra* e *valDestra*.
- N.B.: la condizione del primo if è soddisfatta (*u* è null) solo se inizialmente la funzione *Valuta* è invocata su null. Se inizialmente *Valuta* è invocata su un nodo $u \neq null$ allora la condizione del primo if non sarà mai soddisfatta perché quando è invocata su una foglia, la funzione restituisce il contenuto della foglia.

ALGORITMO PER VERIFICARE SE UN ALBERO BINARIO È COMPLETAMENTE BILANCIATO

Definizioni:

- Albero binario **proprio**: ogni nodo interno ha sempre due figli non vuoti
- Albero **completamente bilanciato**: albero proprio con tutte le **foglie** alla **stessa profondità**

Esempio:



ALGORITMO PER VERIFICARE SE UN ALBERO BINARIO È COMPLETAMENTE BILANCIATO

- Def. ricorsiva di albero completamente bilanciato:
 - Un albero binario vuoto è completamente bilanciato
 - Una albero binario con almeno un nodo è completamente bilanciato se e solo se il sottoalbero destro e il sottoalbero sinistro della radice sono completamente bilanciati e hanno la stessa altezza (per convenzione, un albero vuoto ha altezza -1)
- N.B. In un albero completamente bilanciato l'altezza dell'albero corrisponde alla profondità di tutte le foglie
- Indichiamo con $T(u)$ il sottoalbero di T radicato in u
- Risolviamo un problema più generale per $T(u)$, calcolandone anche l'altezza oltre che a dire se è completamente bilanciato o meno
- La ricorsione restituisce una coppia (booleano, intero)
- Tempo di risoluzione: $O(n)$ tempo per n nodi

```
1 CompletamenteBilanciato( u ):
2   IF (u == null) {
3     RETURN <TRUE, -1>;
4   } ELSE {
5     <bilSX,altSX> = CompletamenteBilanciato( u.sx );
6     <bilDX,altDX> = CompletamenteBilanciato( u.dx );
7     bil = bilSX && bilDX && (altSX == altDX);
8     altezza = max(altSX, altDX) + 1;
9     RETURN <bil,altezza>;
10  }
```

ALGORITMI RICORSIVI SU ALBERI: PROFONDITÀ DI UN NODO

- La radice ha profondità 0
- I figli della radice hanno profondità pari a 1, e così via
- Un nodo ha profondità p ha i figli a profondità $p + 1$

Versione iterativa dell'algoritmo per calcolare la profondità di un nodo u

```
p = 0;
WHILE (u.padre != null) {
  p = p + 1;
  u = u.padre;
}
```

Definizione ricorsiva di profondità di un nodo:

- La radice ha profondità 0
- I nodi diversi dalla radice hanno profondità pari alla profondità del padre + 1

Versione ricorsiva dell'algoritmo per calcolare la profondità di un nodo u

```
1 Profondita( u ):
2   IF (u.padre==null) {
3     RETURN 0;
4   }
5   RETURN profondita(u.padre)+1;
```

TRASMISSIONE DELL'INFORMAZIONE TRA CHIAMATE RICORSIVE

- **postorder** : l'informazione è trasferita dalle foglie alla radice
 - la soluzione del problema per $T(u)$ può essere ottenuta dalla soluzioni dei sottoproblemi per $T(u.sx)$ e $T(u.dx)$
- **passaggio dei parametri** : informazione passata attraverso i parametri dalla radice alle foglie
 - la soluzione del problema per $T(u)$ può essere ottenuta utilizzando l'informazione raccolta dalla radice fino al nodo u

Esempio: stampa la profondità di tutti i nodi

```
1 Profondita( u, p ):
2   IF ( u != null ) {
3     PRINT profondità di u è pari a p;
4     Profondita( u.sx, p+1 );
5     Profondita( u.dx, p+1 );
6   }
```

Il parametro p indica la profondità del nodo u . Se vogliamo stampare le profondità di tutti i nodi dobbiamo invocare la funzione con u uguale all'indirizzo della radice dell'albero e $p = 0$.

ALGORITMO PER TROVARE I NODI CARDINE

Trasferiamo informazione simultaneamente dalle foglie alla radice e dalla radice verso le foglie combinando i due approcci della slide precedente

- Nodo u è cardine se e solo se $profondita(u) = altezza(T(u))$

```
1 Cardine( u, p ):
2   IF ( u == null ) {
3     RETURN -1;
4   } ELSE {
5     altezzaSX = Cardine( u.sx, p+1 );
6     altezzaDX = Cardine( u.dx, p+1 );
7     altezza = max( altezzaSX, altezzaDX ) + 1;
8     IF ( p == altezza ) PRINT u.dato;
9     RETURN altezza;
10  }
```