Dati e Algoritmi 1: A. Pietracaprina

Alberi Binari

Definizione

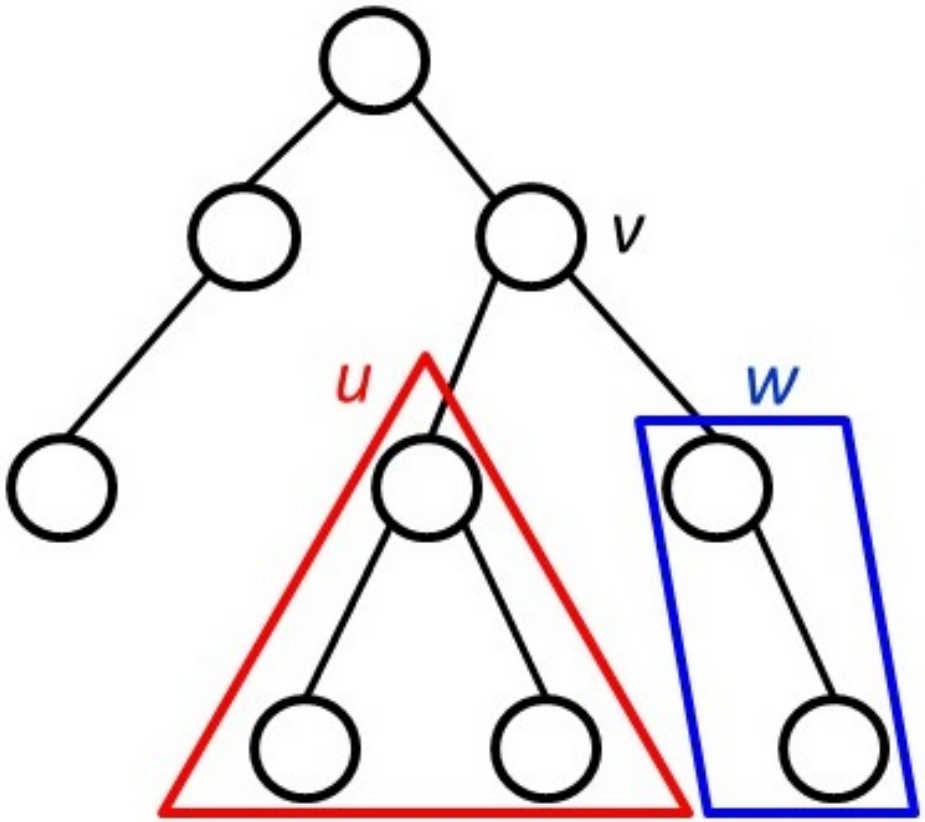
Un alberto binario T è un albero ordinato in cui

- ogni nodo interno ha ≤ 2 figli
- ogni nodo non radice è etichettato come figlio sinistro (sx) o destro (dx) di suo padre
- il figlio sx viene prima del figlio dx nell'ordinamento dei figli di un nodo.

Nota

A meno di casi particolari assumiamo che se c'è un solo figlio, questo sia il figlio sx.

Esempio e Terminologia



u : figlio sx di *v*

w : figlio dx di *v*

sottoalbero sx di *v*

sottoalbero dx di *v*

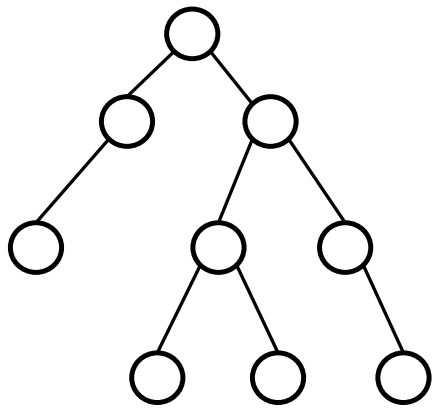
Alberi Binari Propri

Definizione

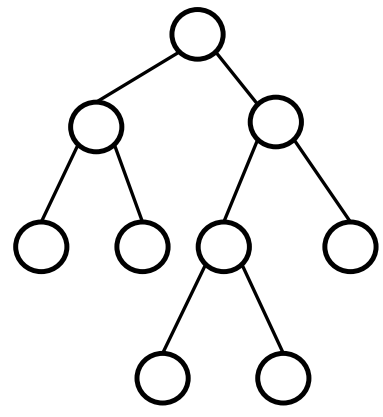
Albero binario proprio T: albero binario tale che:

- ogni nodo interno ha *esattamente* 2 figli

Albero Binario



Albero Binario Proprio



Osservazione

In letteratura gli alberi *propri* (*proper* in inglese) sono anche chiamati *pieni* (*full* in inglese).

Interfaccia BinaryTree

```
public interface BinaryTree<E> extends Tree<E> {
  /** Returns the Position of p's left child (or null if it doesn't exists) */
  Position<E> left(Position<E> p);
  /** Returns the Position of p's right child (or null if it doesn't exists) */
  Position<E> right(Position<E> p);
  /** Returns the Position of p's sibling (or null if no sibling exists) */
  Position<E> sibling(Position<E> p);
}
```

Proprietà di un albero binario proprio T non vuoto

$$
\begin{aligned}
n &= \text{numero di nodi in } T \\
m &= \text{numero di foglie in } T \quad (n_E \text{ in [GTG14]}) \\
n-m &= \text{numero di nodi interni in } T \quad (n_I \text{ in [GTG14]}) \\
h &= \text{altezza di } T
\end{aligned}
$$

Proprietà:

1. $m = n - m + 1$
2. $h + 1 \leq m \leq 2^h$
3. $h \leq n - m \leq 2^h - 1$
4. $2h + 1 \leq n \leq 2^{h+1} - 1$
5. $\log_2(n+1) - 1 \leq h \leq \frac{n-1}{2}$

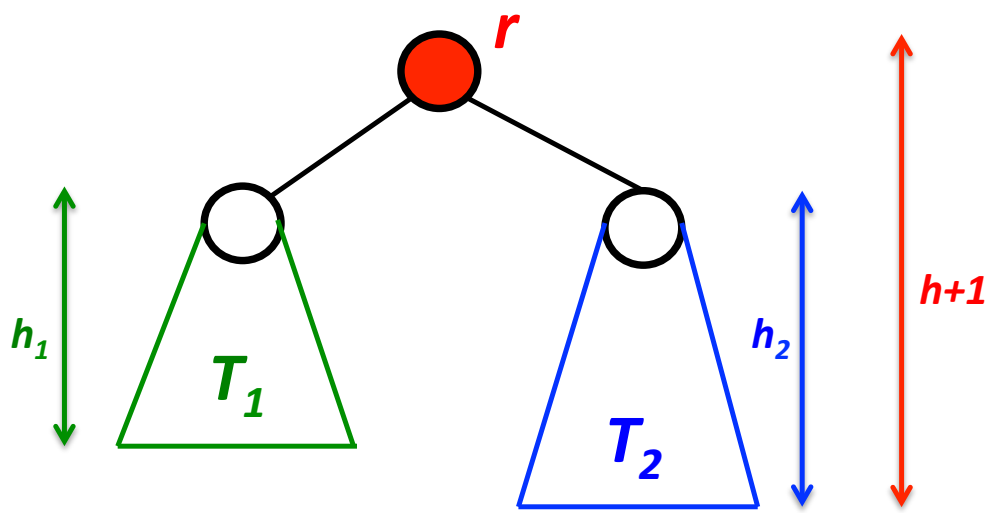
Osservazioni

- Le Proprietà 3, 4 e 5 sono corollari di 1 e 2
- In [GTG14]:
  - Proprietà 1: Proposition 8.8 $\Rightarrow$ Esercizio R-8.8
  - Proprietà 2, 3, 4 e 5: Proposition 8.7 $\Rightarrow$ Esercizio R-8.7
    (Nel testo la proposizione enuncia proprietà analoghe anche per alberi non propri.)

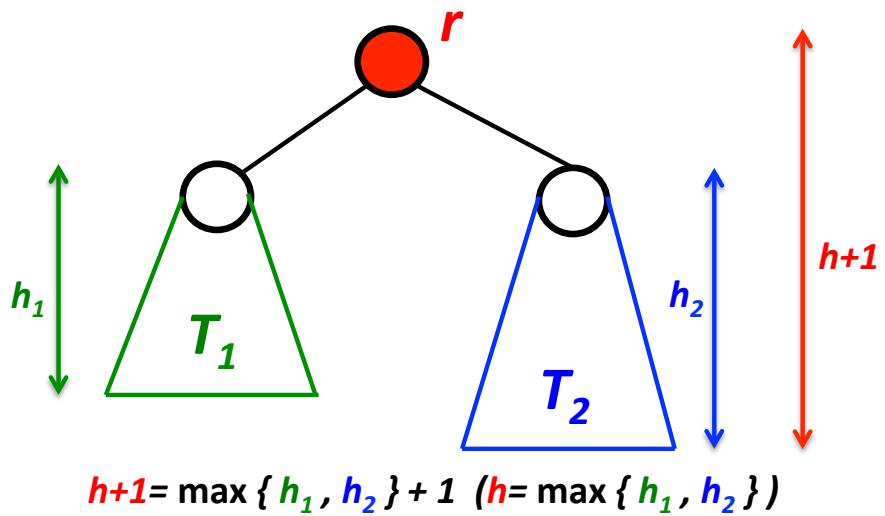
Dimostrazione della Proprietà 1: $m = n - m + 1$

Per induzione sull'altezza dell'albero $h \geq 0$

- Base: $h = 0 \Rightarrow m = 1$ e $n = 1 \Rightarrow$ OK
- Passo induttivo: fissiamo $h \geq 0$ e assumiamo (hp. induttiva) che la proprietà valga $\forall$ albero di altezza $h' \leq h$.
  Consideriamo un albero T di altezza $h + 1 \geq 1$



Dimostrazione della Proprietà 1 (continua)



m_i = numero di foglie in T_i, $i = 1, 2$ $(m = m_1 + m_2)$

n_i = numero di nodi in T_i, $i = 1, 2$ $(n = n_1 + n_2 + 1)$

Si ha quindi:

$$\begin{aligned} m &= m_1 + m_2 \\ &\overset{(hp.ind.)}{=} n_1 - m_1 + 1 + n_2 - m_2 + 1 \\ &= n - m + 1 \end{aligned}$$

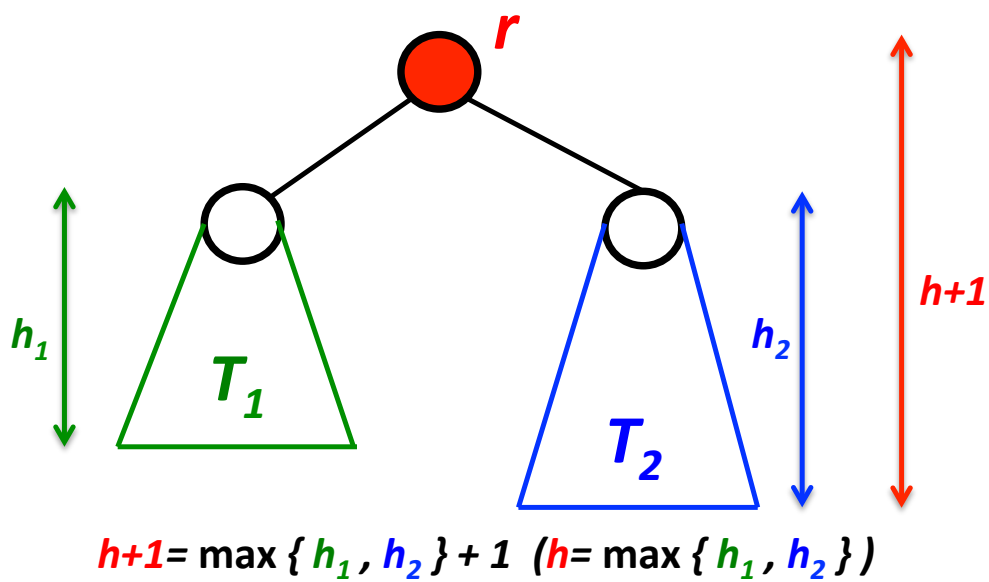
□

Dimostrazione della Proprietà 2: $h + 1 \leq m \leq 2^h$

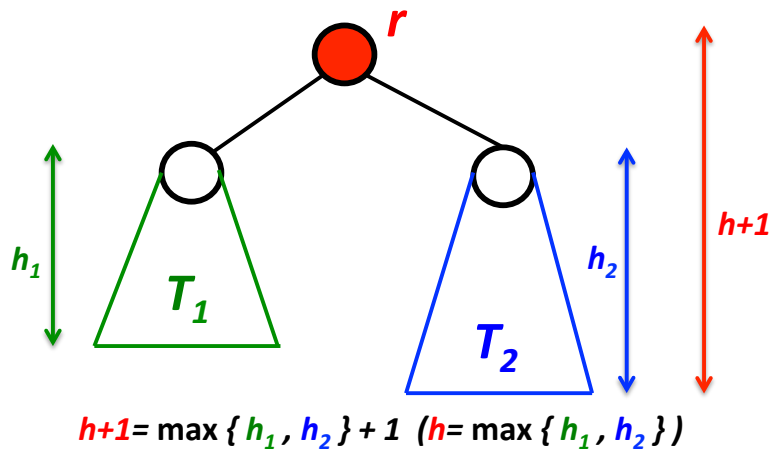
Dimostriamo separatamente le due disuguaglianze:

(a) $m \leq 2^h$. Per induzione sull'altezza dell'albero $h \geq 0$.

- Base: $h = 0 \Rightarrow m = 1$: OK
- Passo induttivo: fissiamo $h \geq 0$ e assumiamo (hp. induttiva) che la proprietà valga $\forall$ albero di altezza $h' \leq h$.
  Consideriamo un albero T di altezza $h + 1 \geq 1$



Dimostrazione della Proprietà 2 (continua)



m_i = numero di foglie in T_i, $\quad i = 1, 2 \quad (m = m_1 + m_2)$

Si ha quindi

$$\begin{array}{rcl} m & = & m_1 + m_2 \\ & \overset{(hp.ind.)}{\leq} & 2^{h_1} + 2^{h_2} \\ & \leq & 2 \times 2^h = 2^{h+1} \end{array}$$

□

Esercizio

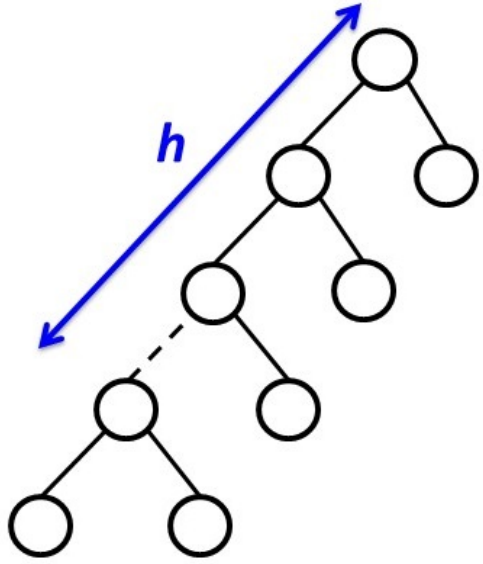
Dimostrare l'altra disuguaglianza della Proprietà 2: $m \geq h + 1$

Osservazione

In assenza di altre ipotesi, il migliore upper bound che possiamo dare all'altezza di un albero binario è

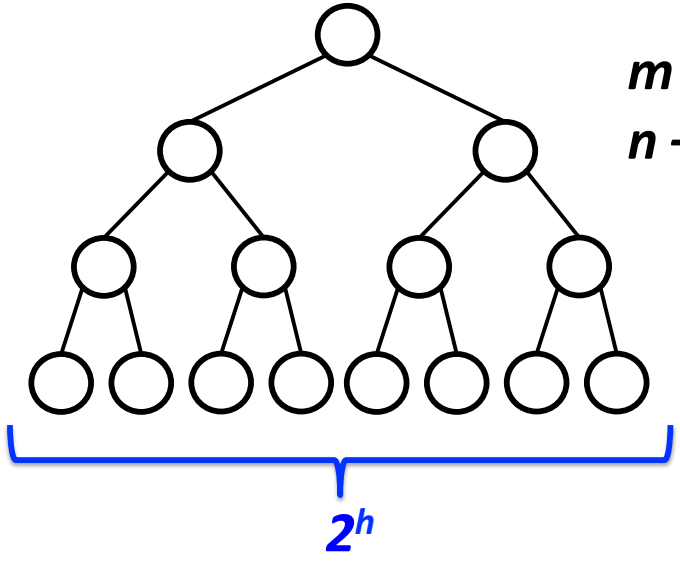
$$O(n).$$

Alberi Binari Propri Estremi



$m = h + 1$
$n - m = h$

$n = 2h + 1$
$h = (n-1)/2$



$m = 2^h$
$n - m = 2^h - 1$

$n = 2^{h+1} - 1$
$h = \log_2(n+1) - 1$

Visite di Alberi Binari

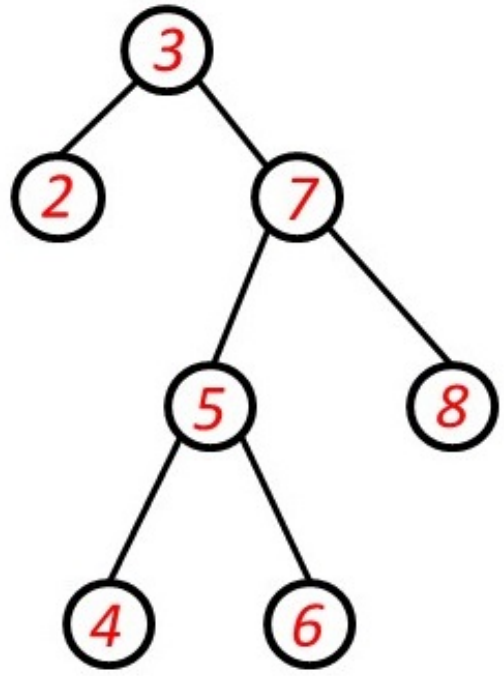
Oltre alle visite in preorder e postorder, per gli alberi binari si definisce anche la *visita inorder*, basata sulla regola:

prima il figlio sx, poi il padre, poi il figlio dx.

Algoritmo `inorder(`T`,`v`)`
Input: T, $v \in T$
Output: visita inorder di T_v
if (T`.left(`v`)` $\neq$ `null`) **then** `inorder(`T`,`T`.left(`v`)`;
visita v;
if (T`.right(`v`)` $\neq$ `null`) **then** `inorder(`T`,`T`.right(`v`)`;

- Chiamata iniziale: inorder(T, $T.root()$).
- Complessità $= \Theta(n)$, dove n è il numero di nodi in T. Stessa analisi delle visite in preorder/postorder.

Esempio



Sequenza inorder: 2 3 4 5 6 7 8

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

Applicazioni delle visite: espressioni aritmetiche

Definizione (Parse Tree)

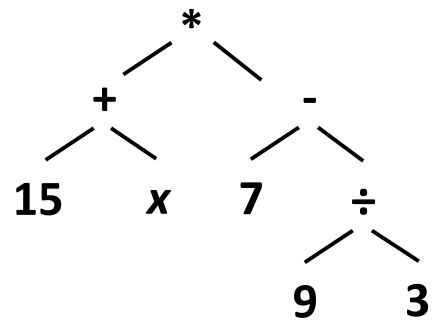
Il Parse Tree T associato a una espressione aritmetica E (con operatori solo binari) è un albero binario proprio i cui nodi foglia contengono le costanti/variabili di E e i nodi interni contengono gli operatori di E, in modo tale che:

- Se $E = a$, con a costante/variabile, allora T è costituito da un'unica foglia contenente a
- Se $E = (E_1 \ \mathrm{Op} \ E_2)$, la radice di T contiene Op e ha come sottoalbero sx (risp., dx) il Parse Tree associato a E_1 (risp., E_2).

Espressione E:

$$((15 + x) * (7 - (9 \div 3)))$$

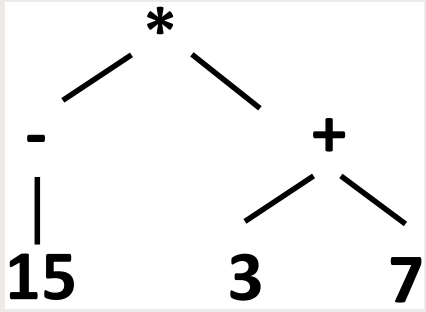
Parse Tree T per E:



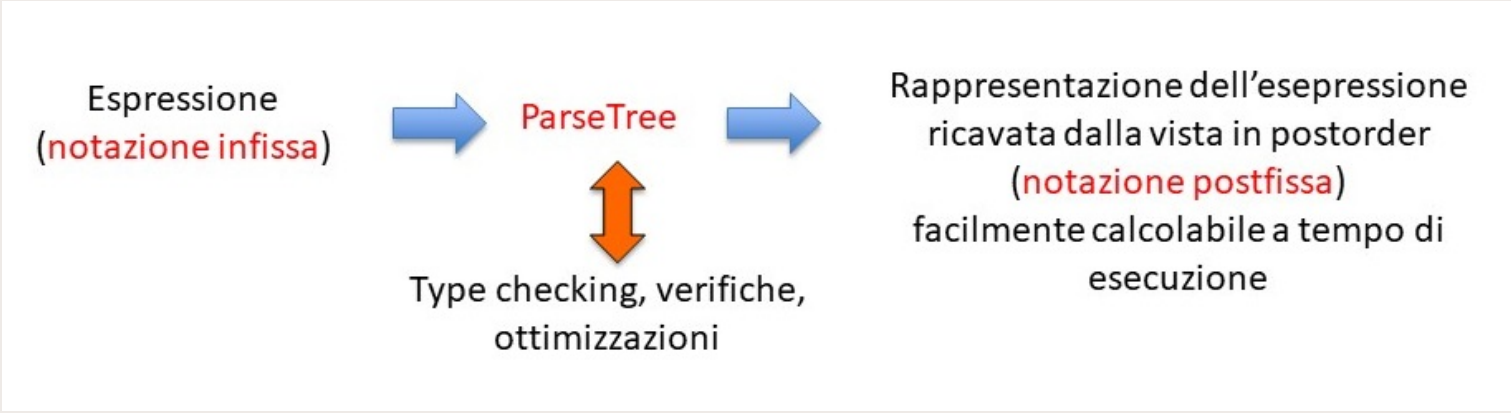
N.B.: Le parentesi sono implicite nella struttura dell'albero

Osservazioni

1. Se si ammettono operatori unari, l'albero non è più proprio.
   Esempio: $-15 * (3 + 7)$



2. Nei compilatori



Espressioni aritmetiche: valutazione dell'espressione

Idea: si utilizza lo schema della visita in postorder

Algoritmo `evaluateExpression`(T,v)
Input: Parse Tree T for E, $v \in T$
Output: Valore dell'espressione associata al sottoalbero T_v

```
if T.isInternal(v) then
    op ← v.getElement();
    x ← evaluateExpression(T, T.left(v));
    y ← evaluateExpression(T, T.right(v));
    return x op y;
else
    return v.getElement();
```

- **Chiamata iniziale:** evaluateExpression(T, $T.root()$).
- **Complessità:** $\Theta(n)$, dove n è il numero di nodi in T. Stessa analisi della visita in postorder.

Espressioni aritmetiche: generazione dell'espressione

Idea: si utilizza lo schema della visita inorder

Algoritmo `infix`(T,v)
Input: Parse Tree T for E, $v \in T$
Output: Espressione E_v associata a T_v (in forma di stringa)

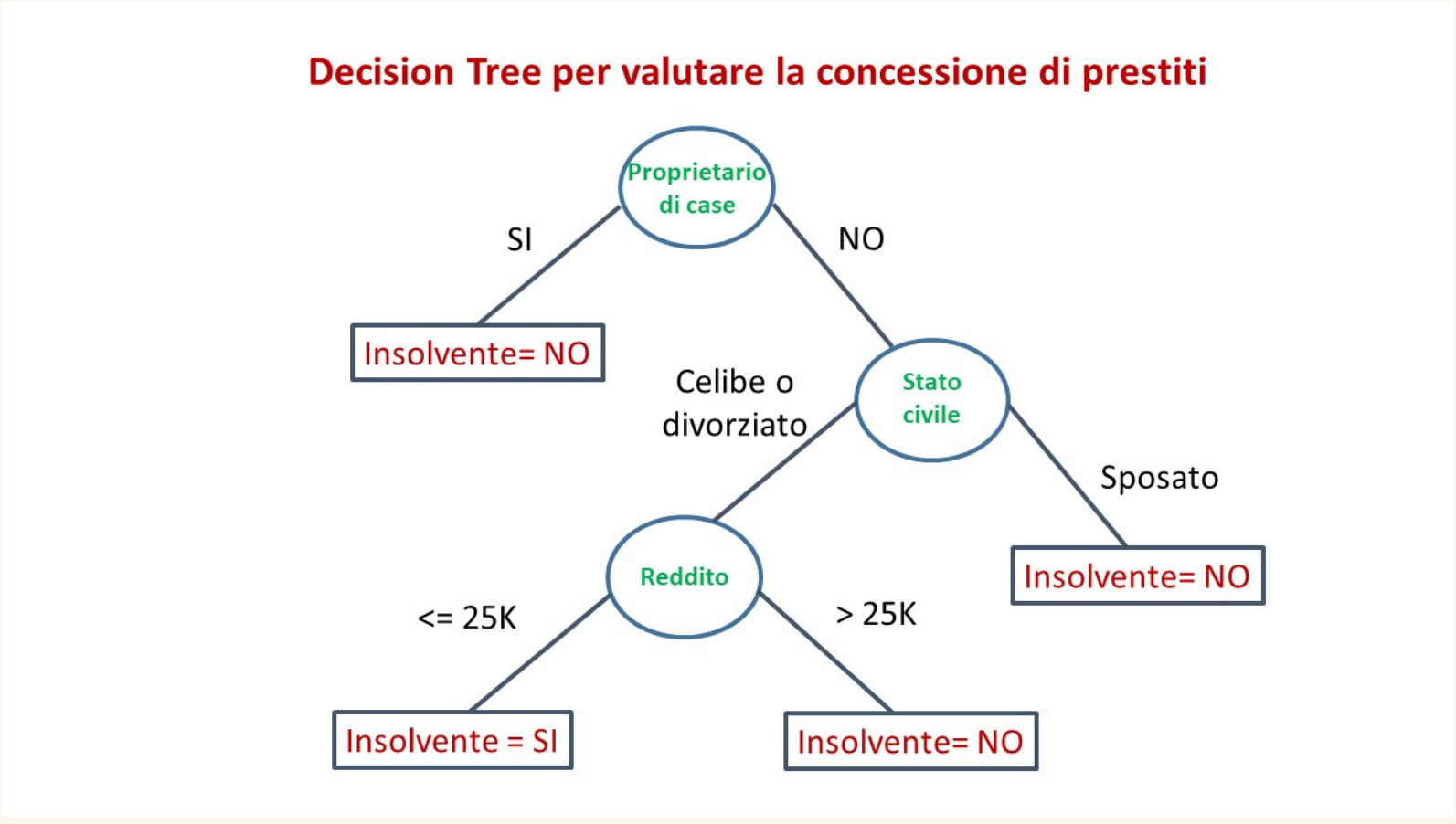
```
if T.isExternal(v) then
    return v.getElement();
else
    return
      '('+infix(T,T.left(v))+v.getElement()+infix(T,T.right(v))+')';
```

- **Chiamata iniziale:** infix(T, $T.root()$).
- **Complessità:** $\Theta(n)$, dove n è il numero di nodi in T. Stessa analisi della visita inorder.

Caso di studio 2: Decision tree

- Machine Learning: Area dell'informatica che si occupa dello sviluppo di algoritmi e modelli per apprendere pattern presenti nei dati e fare predizioni.
- Classification (supervised learning): dato un *tranining set* di record *esempi* caratterizzati da un insieme di *feature* e una *classe*, si cerca un modello che predica la classe a partire dalle feature, per applicarlo poi successivamente per assegnare una classe a nuovi record non classificati.
- I Decision tree sono modelli molto usati per la classificazione: forniscono buona accuratezza; sono semplici da derivare; e sono descrittivi. Vengono spesso utilizzati in ensemble.

Caso di studio 2: Decision tree (continua)



Esercizio

Si vuole progettare un algoritmo *ricorsivo* `heightSum` per calcolare la somma delle altezze di tutti i nodi di un albero binario proprio T. Detta `heightSum(T,v)` la generica invocazione su un nodo $v \in T$, per risolvere il problema si eseguirà `heightSum(T,T.root())`.

1. Descrivere tramite pseudocodice `heightSum(T,v)`, specificandone con attenzione l'input e l'output.
2. Analizzare la complessità di `heightSum(T,T.root())` in funzione del numero n di nodi in T.

Svolgimento

Idea: si utilizza lo schema della visita in postorder e si richiede che `heightSum(T,v)`, oltre alla somma delle altezze dei nodi del sottoalbero T_v (sottoalbero con radice v), restituisca anche l'altezza di v. Altrimenti, dopo aver calcolato ricorsivamente le somme delle altezze nei sottoalberi figli di v non si potrebbe calcolare la somma delle altezze per T_v.

Algoritmo `heightSum`(T, v)
Input: $v \in T$
Output: Somma delle altezze dei nodi di T_v, altezza di v
if T`.isExternal(`v`)` **then**
  return $(0, 0)$;
$(sL, hL) \leftarrow$ `heightSum`$(T, T$.left$(v))$;
$(sR, hR) \leftarrow$ `heightSum`$(T, T$.right$(v))$;
$h \leftarrow \max\{hL, hR\} + 1$;
$s \leftarrow sL + sR + h$;
return (s, h);

Complessità: $\Theta(n)$, dove n è il numero di nodi in T (riferita alla chiamata con $v =$ T.root()). Stessa analisi della visita in postorder.

Osservazione

È un esercizio importante che mostra come a volte per ottenere una strategia algoritmica efficiente sia opportuno *calcolare un'informazione più ricca di quella richiesta* (l'altezza oltre alla somma delle altezze).

Esercizio C-8.41 in [GTG14]

Progettare tre algoritmi `preorderNext(T,v)`, `inorderNext(T,v)` e `postorderNext(T,v)` che dato un albero binario proprio T e un nodo $v \in T$ restituiscano il nodo visitato dopo v nella visita di T rispettivamente in preorder, inorder e postorder, o `null`, se v è l'ultimo nodo visitato, analizzandone la complessità.

Svolgimento (solo preoderNext(T,v))

Idea: capiamo dove può stare il successore di v:

- Se v è un nodo interno, allora il suo successore nella visita in preorder è il suo figlio sinisitro.
- Se v è una foglia, dobbiamo risalire sino a trovare il primo antenato u (cioè l'antenato più profondo) tale che v sta nel sottoalbero sinistro di u. In questo caso, il successore di v nella visita in preorder è il figlio destro di u. Se tale antenato u non esiste, significa che v è l'ultimo nodo visitato dalla vista in preorder di T

Algoritmo `preorderNext`(T, v)
Input: $v \in T$
Output: Successore di v in preorder (o null se non esiste)

```
if T.isInternal(v) then
    return T.left(v);
while (!T.isRoot(v)) do
    if v=T.left(T.parent(v)) then
        return T.right(T.parent(v));
    else v ← T.parent(v);
return null;
```

Complessità: osserviamo che:

- Il ciclo while esegue un numero di iterazioni $\leq \text{depth}_T(v)$.
- Ciascuna iterazione del while esegue $\Theta(1)$ operazioni.
- Per il resto l'algoritmo esegue $\Theta(1)$ operazioni.

Dato che la profondità di v è al più l'altezza h di T, concludiamo che la complessità è $O(h)$.

Esercizi

Esercizio

Dimostrare che in un albero non vuoto T con n nodi di cui m foglie, dove ogni nodo interno ha almeno 2 figli, si ha che $m \geq n - m + 1$.

Esercizio

Si vuole progettare un algoritmo *ricorsivo* `deepLeaf` per trovare la foglia più profonda in un albero binario proprio T. Detta `deepLeaf(T,v)` la generica invocazione su un nodo $v \in T$, per risolvere il problema si eseguirà `deepLeaf(T,T.root())`. Si tenga presente che per ogni sottoalbero T_v la profondità (in T_v) della sua foglia più profonda è pari all'altezza di T_v.

1. Descrivere tramite pseudocodice `deepLeaf(T,v)`, specificandone con attenzione l'input e l'output.
2. Analizzare la complessità di `deepLeaf(T,T.root())` in funzione del numero n di nodi in T.

Esercizi

Esercizio

Sia T un albero binario proprio dove ogni nodo $v \in T$ memorizza un valore v.`val` che può essere 1 o -1. Un nodo $v \in T$ si dice *strong* se la somma dei valori memorizzati nei nodi del sottoalbero T_v è > 0. Progettare un algoritmo ricorsivo per contare il numero di nodi strong in T, analizzandone la complessità.

Esercizio

Completare l'Esercizio C-8.41 in [GTG14].

Riepilogo (alberi generali e binari)

- Definizioni: albero, albero binario (proprio), antenati, discendenti, sottoalbero, profondità di un nodo, altezza di un nodo e di un albero.
- Relazione tra altezza di un albero e profondità delle foglie
- Algoritmi per il calcolo della profondità/altezza di un nodo e dell'altezza di un albero
- Relazioni tra numero di nodi interni, foglie e altezza in un albero binario proprio
- Visite: preorder, postorder, inorder e loro applicazioni
- Algoritmi di visita come template generali

Esempio domande prima parte

- Come sono definiti gli antenati di un nodo in un albero T?
- Descrivere l'algoritmo ricorsivo per il calcolo della profondità di un nodo analizzandone la complessità.
- Dimostrare per induzione che per ogni albero binario proprio T non vuoto con m foglie e altezza h vale $m \leq 2^h$ (induzione su h).
- Sia T un albero binario con $n \geq 3$ nodi e radice r. Sia u il figlio sinistro di r e v il figlio destro di r.
  1. Qual è il nodo successore di u nella visita in postorder di T?
  2. Qual è il nodo successore di v nella visita in postorder di T?

Errata

Cambiamenti rispetto alla prima versione dei lucidi:

- Lucido 13: corretta la figura in alto.