Dati e Algoritmi I (Pietracaprina)

Esercizi sugli Alberi

Problema 1 *Dimostrare che un albero non vuoto con n nodi interni, dove ogni nodo interno ha almeno 2 figli, ha almeno $n+1$ foglie*

Soluzione. La dimostrazione procede per induzione su n. Come base osserviamo che la proprietà é vera per un albero con un solo nodo (foglia), quindi con $n = 0$. Supponiamo che la proprietà sia vera per alberi con al più $n-1$ nodi interni, $n \geq 1$, e consideriamo un albero T con n nodi interni. Sia $d \geq 2$ il numero di figli della radice. Per $1 \leq i \leq d$ sia n_i il numero di nodi interni ed m_i il numero di foglie nell'i-esimo sottoalbero figlio della radice. Sia inoltre m il numero di foglie di T. Si ha allora che

$$\begin{aligned} n &= 1 + \sum_{i=1}^{d} n_i \\ m &= \sum_{i=1}^{d} m_i. \end{aligned}$$

Poichè $n_i < n$ per ogni i, possiamo applicare l'ipotesi induttiva e dedurre che

$$m = \sum_{i=1}^{d} m_i \geq \sum_{i=1}^{d} (n_i + 1) = d + \sum_{i=1}^{d} n_i = n + d - 1 \geq n + 1,$$

in quanto $d \geq 2$. $\square$

Problema 2 *Sia T un albero binario proprio. Dato un nodo $v \in T$, si definisca* **imbalance**(v) *la differenza in valore assoluto tra il numero di foglie nei sottoalberi sinistro e destro di v (se v è una foglia* **imbalance**$(v) = 0$*). Si definisca anche* **imbalance**$(T) = \max_{v \in T}$ **imbalance**(v).

a. *Dimostrare un limite superiore all'imbalance di un albero binario proprio con n nodi, e descrivere un albero il cui imbalance raggiunge tale limite.*

b. *Disegnare un albero binario proprio T in cui* **imbalance**$(T) =$ **imbalance**(v) *e v non è la radice dell'albero.*

c. *Sviluppare un algoritmo efficiente per determinare* **imbalance**(T)*, e analizzarne la complessità in tempo.*

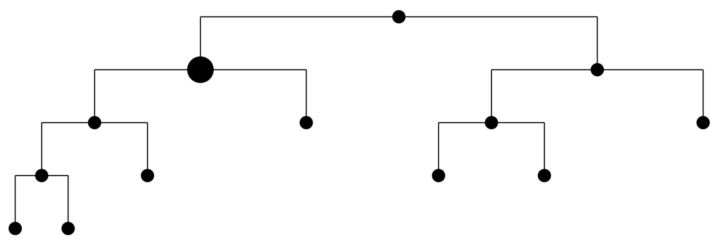
Soluzione.

a. Se l'albero è proprio e ha n nodi, avrà $m = (n+1)/2$ foglie. Sia v un nodo interno. Allora, poichè l'albero è proprio, vi sarà almeno una foglia nel sottoalbero sinistro e almeno una foglia nel sottoalbero destro di v. Quindi

$$\textbf{imbalance}(v) \leq m - 2 = (n-3)/2.$$

Un albero con n nodi il cui imbalance raggiunge tale limite è quello in cui la radice ha come figlio sinistro una foglia, e come figlio destro un albero binario proprio con $n-2$ nodi.

b. L'imbalance del seguente l'albero è 2 ed è ottenuto nel nodo evidenziato da un pallino più grande.



c. Per calcolare **imbalance**(T) utilizziamo il seguente algoritmo ricorsivo che esegue una visita in postorder dell'albero determinando per ogni nodo v l'imbalance del sottoalbero T_v e il numero di foglie in T_v. Per determinare **imbalance**(T) l'algoritmo sarà invocato a partire dalla radice r.

Algoritmo `MaxImbalance(v)`
input $v \in T$
output $b =$ **imbalance**(T_v) `e` $m =$ `numero di foglie in` T_v

if `(T.isExternal(v))` **then return** {`0,1`}
{`bleft, mleft`} $\longleftarrow$ `MaxImbalance(T.left(v))`
{`bright, mright`} $\longleftarrow$ `MaxImbalance(T.right(v))`
`b` $\longleftarrow$ max {`bleft, bright, |mleft - mright|`}
`m` $\longleftarrow$ `mleft+mright`
return {`b,m`}

La complessità dell'algoritmo è la stessa di quella della visita in preoder, ovvero lineare nel numero di nodi dell'albero.

□

Problema 3 *Sia T un albero binario e si definisca $\Delta(T)$ come la massima distanza tra due nodi v e u tali che v è l'immediato successore di u nella visita inorder dell'albero. (Si ricordi che la distanza tra due nodi generici u e v è definita come la lunghezza del cammino nell'albero che collega u e v, ovvero $d_u + d_v - 2d_w$, dove w è il lowest common ancestor di u e v, e d_u, d_v e d_w sono le profondità, rispettivamente, di u, v e w.)*

a. *Sia T un albero binario. Dimostrare che se v è l'immediato successore di u nella vista inorder di T, allora o v è il lowest common ancestor di u e v, oppure, viceversa, u è il lowest common ancestor di u e v.*

b. *Determinare $\Delta(T)$ nei seguenti due casi.* **Caso 1***: T albero binario completo di altezza h con 2^h nodi al livello h.* **Caso 2***: T costituito da una catena di $h+1$ nodi $v_0, v_1, \ldots v_h$ dove v_0 è la radice di T e v_{i+1} è il figlio destro di v_i, per $0 \leq i < h$.*

c. *Sviluppare e analizzare un algoritmo efficiente che dato un albero binario T determina $\Delta(T)$.*

Soluzione.

a. Sia v l'immediato successore di u nella visita inorder di T, e sia w il loro lowest common ancestor. Se per assurdo w fosse diverso da u e v, allora u sarebbe nel sottoalbero sinistro di w, e v in quello destro. Ma in questo caso, w verrebbe tra u e v nella visita inorder e quindi v non potrebbe essere il successore di u, che contraddice quanto ipotizzato all'inizio.

b. **Caso 1**: Dal punto precedente si ricava che $\Delta(T) \leq h$ per ogni albero binario di altezza h. Se T ha altezza h e 2^h nodi al livello h (quindi è anche completo) allora la radice e il suo predecessore sono a distanza h, il che implica che $\Delta(T) = h$. **Caso 2**: la visita inorder di T tocca i nodi nell'ordine $v_0, v_1, \ldots, v_h$, e quindi $\Delta(T) = 1$.

c. Siano v e u due nodi di T tali che v è l'immediato successore di u nella visita inorder dell'albero e la loro distanza è pari a $\Delta(T)$. Sia r la radice di T. Il punto (a) implica che per v e u vale uno dei seguenti 4 casi:

- v e u sono entrambi nel sottoalbero sinistro di r;
- v e u sono entrambi nel sottoalbero destro di r;
- $v = r$ e u è l'ultimo nodo nella visita inorder del sottoalbero sinistro di r;
- $u = r$ e v è il primo nodo nella visita inorder del sottoalbero destro di r.

Dai 4 casi si ricava il seguente algoritmo ricorsivo `FindDelta`(T, w), che invocato su un nodo $w \in T$ restituisce $\Delta(T_w)$ e due valori h_a e h_z tali che h_a è la distanza tra w e il primo nodo nella visita inorder di T_w, e h_z è la distanza tra w e l'ultimo nodo nella visita inorder di T_w. Inizialmente, l'algoritmo viene invocato con $w = r$, dove r è la radice di T.

```
Algoritmo FindDelta(T,w)
if (T.hasLeft(w)) then
  (dL,hLa,hLz) ⟵ FindDelta(T,T.left(w))
  dL ⟵ max {dL,hLz+1};
  ha ⟵ hLa+1;
else
  dL ⟵ 0;
  ha ⟵ 0;
if (T.hasRight(w)) then
  (dR,hRa,hRz) ⟵ FindDelta(T,T.right(w))
  dR ⟵ max {dR,hRa+1};
  hz ⟵ hRz+1;
else
  dR ⟵ 0;
  hz ⟵ 0;
d ⟵ max {dL,dR};
return (d,ha,hz);
```

Si noti che la struttra dell'algoritmo è simile a quella di un qualsiasi algoritmo di visita, e quindi la sua complessità è lineare nel numero di nodi dell'albero.

Per quanto riguarda la correttezza, possiamo provarla per induzione sull'altezza di T_w (con riferimento all'invocazione `FindDelta`(T, w)). Se T_w ha altezza 0 allora w è un nodo foglia e l'algoritmo restituisce, correttamente, la terna $(0, 0, 0)$. Fissiamo $h \geq 0$ e supponiamo, per ipotesi induttiva, che l'algoritmo funzioni correttamente per tutti i T_w di altezza $\leq h$. Supponiamo ora che T_w abbia altezza $h + 1$. Poichè $h + 1 \geq 1$ il nodo w deve avere almeno uno dei due figli. Distinguiamo i seguenti tre casi.

Supponiamo che w sia interno ma che abbia il solo figlio sinistro che chiamiamo y. Allora l'algoritmo restituisce la terna `(d,ha,hz)` dove

- `d` è il massimo tra $\Delta(T_y)$ e (1+`hLz`). Si noti che (1+`hLz`) è la distanza tra w e l'ultimo nodo nella visita inorder di T_y, ovvero la distanza tra w e il suo predecessore nella visita inorder di T_w
- `ha` è 1 più la distanza tra y e il primo nodo nella visita inorder di T_y, che è anche il primo nodo nella visita inorder di T_w.
- `hz`=0 in quanto w è l'ultimo nodo nella visita inorder di T_w.

Il caso in cui w abbia il solo figlio destro è analogo.

Se w ha sia il figlio sinistro, che chiamiamo y, che il figlio destro, che chiamiamo x, allora l'algoritmo restituisce la terna `(d,ha,hz)` dove

- `d` è il massimo tra $\Delta(T_y)$, $\Delta(T_x)$, (1+`hLz`) e (1+`hRa`), dove (1+`hLz`) è la distanza tra w e il suo predecessore nella visita inorder di T_w, e (1+`hRa`) è la distanza tra w e il suo successore nella visita inorder di T_w.
- `ha` è 1 più la distanza tra y e il primo nodo nella visita inorder di T_y, che è anche il primo nodo nella visita inorder di T_w.
- `hz` è 1 più la distanza tra x e l'ultimo nodo nella visita inorder di T_x, che è anche l'ultimo nodo nella visita inorder di T_w.

E' facile concludere che in tutti e tre i casi analizzati sopra l'algoritmo risulta corretto.

□

Problema 4 *Si definisca* albero d-ario proprio *un albero ordinato in cui ogni nodo interno ha esattamente d figli.*

a. *Disegnare due alberi ternari ($d = 3$) propri T_1 e T_2 con 8 nodi interni ciascuno, tali che T_1 ha altezza massima e T_2 ha altezza minima. Quante foglie hanno T_1 e T_2?*

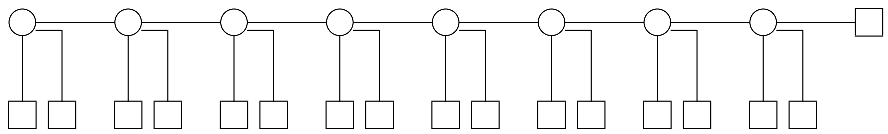
b. *Sia T un albero d-ario proprio di altezza h con n nodi interni ed m foglie. Usando gli esempi precedenti e il buon senso trovare l'unica tra le seguenti relazioni che può valere per T arbitrario e $d \geq 2$:*

$$\text{(i) } m = (d-1)^h + 1; \text{ (ii) } m = (d-1)n + 1; \text{ (iii) } m = 5d + 2$$

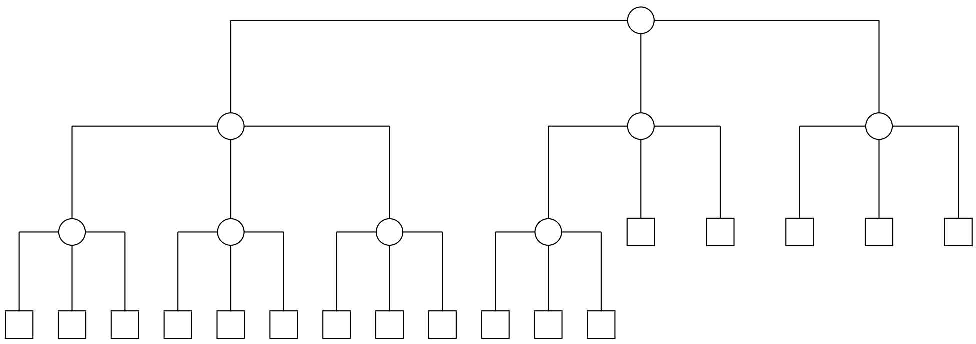
c. *Dimostrare la relazione trovata al punto precedente per induzione su h.*

Soluzione.

a. L'albero T_1 è il seguente (le foglie sono rappresentate da quadrati e i nodi interni da cerchi).



L'albero T_2 è il seguente:



Entrambi gli alberi hanno 17 foglie.

b. La (i) non è soddisfatta dall'albero T_1. La (iii), pur essendo soddisfatta sia da T_1 che da T_2, non ha senso in quanto non dipende da n. L'unica che può valere è la (ii).

c. Dimostriamo che $m = (d-1)n+1$ per induzone sull'altezza h dell'albero. La base $h = 0$ è vera in quanto un albero di altezza 0 ha 1 foglia e 0 nodi interni. Sia vera la relazione per alberi di altezza sino ad $h-1$ e consideriamo un albero T di altezza $h > 0$. Sia T_i l'i-esimo sottoalbero figlio della radice di T, $1 \leq i \leq d$, e si indichi con n_i il numero di nodi interni di T_i e con m_i il numero di foglie di T_i. Poichè ogni T_i ha altezza al più $h-1$, per ipotesi induttiva vale che $m_i = (d-1)n_i + 1$. Inoltre

$$
\begin{aligned}
m &= \sum_{i=1}^{d} m_i \\
n &= \sum_{i=1}^{d} n_i + 1,
\end{aligned}
$$

e quindi

$$m = \sum_{i=1}^{d}((d-1)n_i + 1) = (d-1)n + 1.$$

□