Alberi

Indovina l'albero

Gli ordini di visita di un albero binario di 9 nodi sono i seguenti:

- A, E, B, F, G, C, D, I, H (anticipato)
- B, G, C, F, E, H, I, D, A (posticipato)
- B, E, G, F, C, A, D, H, I (simmetrico).

Si ricostruisca l'albero binario e si illustri *brevemente* il ragionamento.

Alberi

Albero livello-valore

Scrivere un algoritmo che preso in input un albero binario T i cui nodi sono associati ad un valore intero $T.key$, restituisca il numero di nodi dell'albero il cui valore è pari al livello del nodo. Vi ricordo che il livello del nodo è pari al numero di archi che devono essere attraversati per raggiungere il nodo dalla radice. Per cui la radice ha livello 0, i suoi figli hanno livello 1, etc.

Alberi

Cammino radice–discendente crescente

Dato un albero binario contenente interi, scrivere un algoritmo che restituisca la lunghezza del più lungo cammino monotono crescente radice-discendente, dove:

- il discendente non è necessariamente foglia;
- con lunghezza si intende il numero totale di *archi* attraversati;
- con monotona crescente si intende che i valori contenuti nei nodi della sequenza devono essere ordinati in senso crescente da radice a discendente.

Discuterne correttezza e complessità.

Alberi

Grado di sbilanciamento

Dato un albero binario T, il grado di sbilanciamento di un nodo v è pari alla differenza, in valore assoluto, fra il numero di foglie presenti nel sottoalbero sinistro di v e il numero di foglie presenti nel sottoalbero destro di v. Il grado di sbilanciamento dell'albero T è pari al massimo grado di sbilanciamento dei nodi di T.

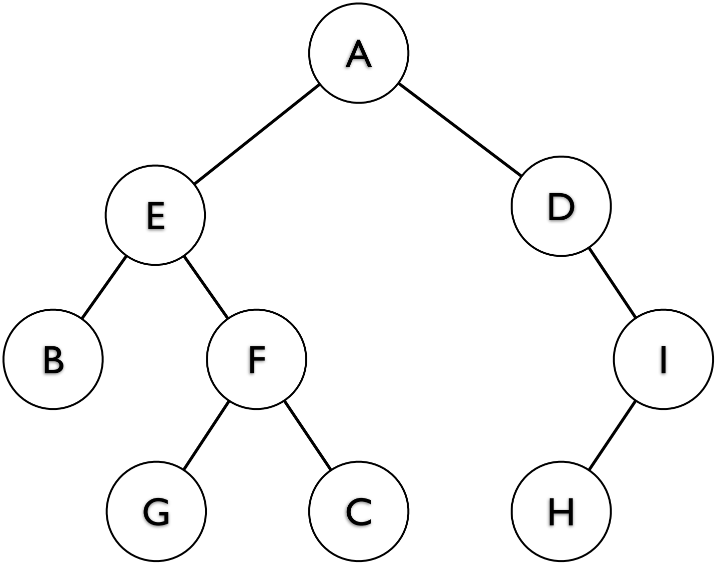
Scrivere un algoritmo che dato un albero T restituisce il grado di sbilanciamento dell'albero. Discuterne correttezza e complessità.

Alberi – Larghezza albero

La larghezza di un albero ordinato è il numero massimo di nodi che stanno tutti al medesimo livello. Si fornisca una funzione che calcoli in tempo ottimo la larghezza di un albero ordinato T di n nodi.

Spoiler alert!

Indovina l'albero



Albero livello-valore

Una semplice visita posticipata risolve il problema. La complessità è ovviamente $O(n)$.

```
int sameLevel(TREE T)
---
return sameLevelRec(T, 0)
```

```
int sameLevelRec(TREE T, int ℓ)
---
int count = 0
if T ≠ nil then
    count = sameLevelRec(T.right(), ℓ + 1) + sameLevelRec(T.left(), ℓ + 1)
    if T.key == ℓ then
        count = count + 1
return count
```

Alberi – Percorso cammino-discendente

```
int monotone(TREE T)
int maxl = 0
int maxr = 0
if T ≠ nil then
    if T.left() ≠ Nil and T.left().value > T.value then
        maxl = 1 + monotone(T.left())
    if T.right() ≠ Nil and T.right().value > T.value then
        maxr = 1 + monotone(T.right())
return max(maxl, maxr)
```

Alberi - Grado di sbilanciamento

int, **int** unbalance(Tree T)

if $T = \textbf{nil}$ **then**
  return $(0, 0)$
if $T.\mathit{left} = \textbf{nil}$ **and** $T.\mathit{right} = \textbf{nil}$ **then**
  return $(1, 0)$
$L_{leafs}, L_{max} \leftarrow \mathsf{unbalance}(T.\mathit{left})$
$R_{leafs}, R_{max} \leftarrow \mathsf{unbalance}(T.\mathit{right})$
return $(L_{leafs} + R_{leafs}, \mathsf{max}(L_{max}, R_{max}, |L_{leafs} - R_{leafs}|))$

Larghezza (1)

```
int breadth(TREE t)

int breadth = 1
int level = 1
int count = 1
QUEUE Q = Queue()
Q.enqueue(t)
t.level = 0
while not Q.isEmpty() do
    TREE u = Q.dequeue()
    if u.level ≠ level then
        level = u.level
        count = 0
    count = count + 1
    breadth = max(breadth, count)
    TREE v = u.leftmostChild()
    while v ≠ nil do
        v.level = u.level + 1
        Q.enqueue(v)
        v = v.rightSibling()
return breadth
```

Larghezza (2)

```
int breadth(Tree t)
int count = 1               % # nodi nel livello corrente da visitare; radice
int breadth = 1             % Massima larghezza trovata finora; radice
Queue Q = Queue()
Q.enqueue(t)
while not Q.isEmpty() do
    Tree u = Q.dequeue()
    Tree v = u.leftmostChild()
    while v ≠ nil do
        Q.enqueue(v)
        v = v.rightSibling()
    count = count − 1
    if count == 0 then                    % Nuovo livello
        count = Q.size()
        breadth = max(breadth, count)
return breadth
```