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 $T.value$, restituisca il numero di nodi dell'albero il cui valore è uguale 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

Si consideri 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 quelle 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, restituisca il grado di sbilanciamento dell'albero. Discuterne correttezza e complessità.

Nota: In pseudocodice, è possibile restituire una coppia di valori:

(int, int) fun1(TREE T)

```
[...]
return (a, b)
```

int fun2(TREE T)

```
int, int x, y = fun1(TREE T)
return y
```

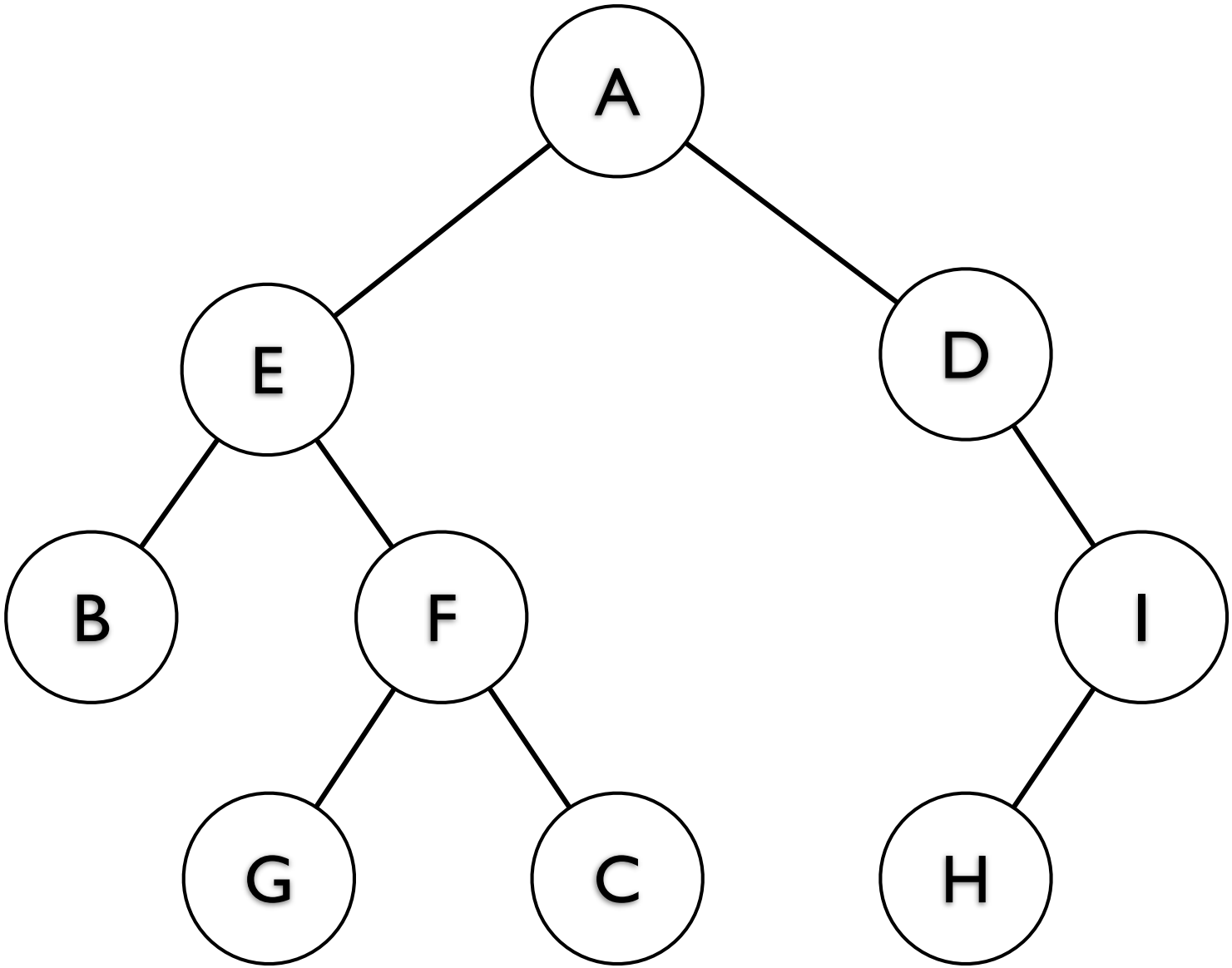
Ricorrenza $2T(n/8) + 2T(n/4) + n$

Trovare un limite asintotico superiore e un limite asintotico inferiore alla seguente ricorrenza, facendo uso del metodo di sostituzione:

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

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 tot = 0
if T ≠ nil then
    tot = sameLevelRec(T.right(), ℓ + 1) + sameLevelRec(T.left(), ℓ + 1)
    if T.value == ℓ then
        tot = tot + 1
return tot
```

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 unbalance(TREE T)

int, **int** $leafs, max =$ unbalanceRec(TREE T)
return max

(**int**, **int**) unbalanceRec(TREE T)

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

Ricorrenza $2T(n/8) + 2T(n/4) + n$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

È facile dimostrare che ovviamente, la funzione $T(n)$ è $\Omega(n)$; infatti,

$$T(n) = 2T(n/8) + 2T(n/4) + n \geq n \geq c_1 n$$

per $c_1 \leq 1$. Non è necessario dimostrare il caso base, perché abbiamo rimosso gli elementi ricorsivi e quindi non abbiamo bisogno di induzione.

Ricorrenza $2T(n/8) + 2T(n/4) + n$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

Una prima osservazione che si potrebbe fare per "indovinare" un limite superiore è la seguente:

$$\begin{aligned} T(n) &= 2T(n/8) + 2T(n/4) + n \\ &\leq 2T(n/4) + 2T(n/4) + n \\ &\leq 4T(n/4) + n \end{aligned}$$

In base a questo risultato, il master theorem dice che $T(n) = O(n \log n)$.

Ricorrenza $2T(n/8) + 2T(n/4) + n$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/8) + 2T(n/4) + n & n > 1 \end{cases}$$

A questo punto, non ci resta che vedere quale dei due risultati ($\Omega(n)$ e $O(n \log n)$ è stretto. Proviamo con $O(n)$.
Ipotesi induttiva: $\forall k < n : T(k) \leq ck$. Proviamo che il risultato è valido anche per n.

$$\begin{aligned} T(n) &= 2T(n/8) + 2T(n/4) + n \\ &\leq 2cn/8 + 2cn/4 + n \\ &= 3/4cn + n \\ &\leq cn \end{aligned}$$

L'ultima disequazione è vera se $3/4c + 1 \leq c$, ovvero se $c \geq 4$.