Famiglia

Trovare i limiti superiore e inferiore più stretti possibili per la seguente famiglia di equazioni di ricorrenza, per valori di a interi positivi.

$$T(n) = \begin{cases} aT(\lfloor n/2 \rfloor) + n^{a-1} & n \geq 2 \\ 1 & n < 2 \end{cases}$$

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:

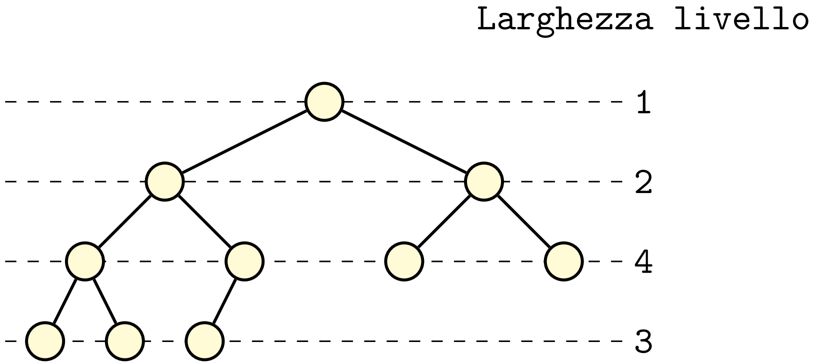
```
(int, int) fun1(Tree T)
[...]
return (a, b)
```

```
int fun2(Tree T)
int, int x, y = fun1(Tree T)
return y
```

Alberi – Larghezza albero

La larghezza di un albero T è il numero massimo di nodi di T che stanno tutti al medesimo livello.

Scrivere un algoritmo che restituisca la larghezza di un albero ordinato T contenente n nodi.



Spoiler alert!

Famiglia (24/07/20)

Data la forma della ricorrenza, è possibile utilizzare il Master Theorem, versione base.

- Per $a = 1$, abbiamo $\alpha = \log_2 1 = 0$, $\beta = 0$; siamo nel secondo caso, $T(n) = \Theta(\log n)$.
- Per $a = 2$, abbiamo $\alpha = \log_2 2 = 1$, $\beta = 1$; siamo nel secondo caso, $T(n) = \Theta(n \log n)$.
- Per $a = 3$, abbiamo $\alpha = \log_2 3 < 2$, $\beta = 2$; siamo nel terzo caso, $T(n) = \Theta(n^2)$.
- In generale, è possibile dimostrare che $\log_2 a < a - 1$ per tutti i valori interi $a \geq 3$; siamo nel terzo caso e si ottiene $T(n) = \Theta(n^{a-1})$.

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 $(L_{leafs} + R_{leafs}, \mathsf{max}(L_{max}, R_{max}, |L_{leafs} - R_{leafs}|))$

Larghezza (1)

```
int breadth(TREE t)

int breadth = 0
int level = 1
int count = 1
QUEUE Q = Queue()
Q.enqueue(t)
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
```