Qualche problema sugli Alberi

corso di laurea in **Matematica**

Informatica Generale, Lezione **18(b)**

Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

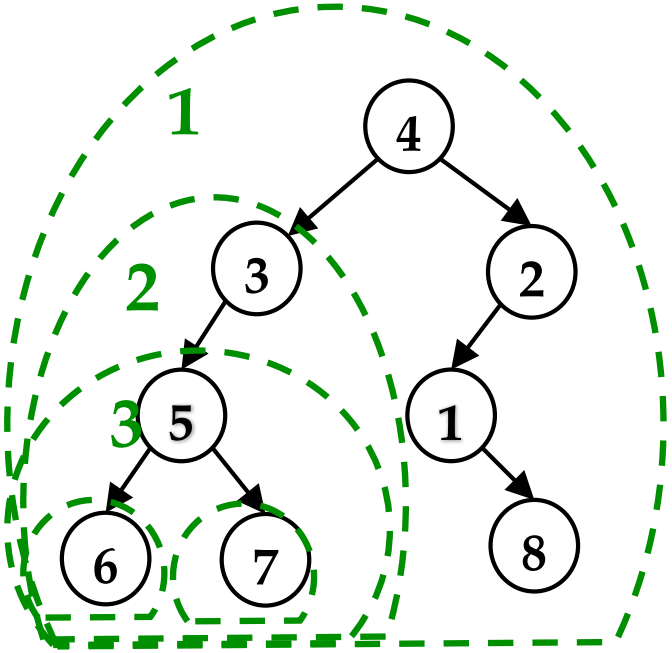
Bilanciamento: soluzione naïf

Scriviamo una funzione che verifica se un albero è **bilanciato in altezza**. Possiamo ancora una volta seguire la logica post-order e scrivere una semplicissima funzione.

Osserviamo però che per calcolare **l'altezza di un albero**, si **ricalcoliamo tante volte** i valori dell'altezza per gli **stessi sottoalberi**.

Si ottiene una **funzione $\Theta(n^2)$**.

```
def bilanciato(B):
    if B == EMPTY: return True
    if bilanciato(lft[B])
        and bilanciato(rgt[B])
        and abs(h(lft[B]) - h(rgt[B])) ≤ 1:
            return True
    return False
```



È possibile **calcolare tutto** quanto ci serve in **un'unica scansione dell'albero**?

Bilanciamento: soluzione lineare

Come in altre occasioni, il trucco consiste nel **comunicare informazione** tra diverse **chiamate ricorsive**.

In questo caso, l'informazione va passata **all'indietro** e possiamo usare i **valori di ritorno**: **calcoliamo simultaneamente** altezza e bilanciamento.

$\Theta(n)$

```
def bilanciatoAux(B):
    if B == EMPTY: return True, -1
    bl, hl = bilanciatoAux(lft[B])
    br, hr = bilanciatoAux(rgt[B])
    if bl and br and abs(hl – hr) ≤ 1:
            return True
    return False

def bilanciatoSmart(B):
    b, h = bilanciatoAux(B)
    return b
```

si può evitare la seconda chiamata se bl è false.

cammino fino a x

[Scritto 17/6/16]

Problema: Scrivere un algoritmo che preso un albero B e un valore x torna la **sequenza delle chiavi** di un cammino dalla radice a un nodo con chiave x e NULL se x non viene trovato.

Idea: costruire una lista al ritorno, dopo aver trovato x.
Si propaga NULL altrimenti.

➧ **Esercizio**: scriverla sotto l'ipotesi che B sia un ABR.

$\Theta(n)$

```
def pathToX(B, x):
    if B == EMPTY: return NULL
    if key[B] == x:
        return cons(x, NULL)
    L = pathToX(left(B), x)
    if L ≠ NULL:
        return cons(x, L)
    L = pathToX(right(B), x)
    if L ≠ NULL:
        return cons(x, L)
    return NULL
```

nodi di livello k

[secondo esonero 2016]

Problema: Scrivere un algoritmo che preso un albero B restituisce la lista dei nodi al livello k.

Molti studenti, leggendo "livelli" si sono lanciati su una visita a livelli, **con tutte le complicazioni del caso** (ad esempio riconoscere nodi allo stesso livello dentro la coda)

$\Theta(n^2)$

```
def levelK(B, k):
    if B == EMPTY: return NULL
    if k == 0: return cons(key[B], NULL)
    return append(levelK(left(B), k-1),
                  levelK(right(B), k-1)
```

Questa funzione **è quadratica**, se append non si riesce a fare in $\Theta(1)$ come accade con le liste semplici.

Notare che si va **prima a destra**, poi a sinistra, per far uscire la lista dritta.

Prima si esegue la **chiamata interna**.

$\Theta(n)$

```
def levelK(B, L, k):
    if B == EMPTY: return L
    if k == 0: return cons(key[B], L)
    return levelK(left(B),
        levelK(right(B), L, k-1), k-1)
```