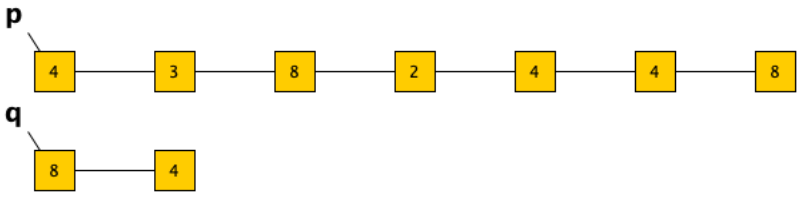
Introduzione agli Algoritmi
Secondo Esonero riservato a studenti secondo Canale

docente: A. Monti
Sapienza Università di Roma
Maggio 2024

Esercizio 1 (15 punti): Abbiamo una lista a puntatori, ciascun nodo della lista è stato generato tramite la classe $Nodo$ vista a lezione. Ogni nodo ha quindi ha due campi: il campo key contenente interi ed il campo $next$ con il puntatore al prossimo nodo della lista (o $None$ per l'ultimo nodo).
Progettare un algoritmo che, dato il puntatore p alla testa della lista, senza modificarla restituisce il puntatore alla testa di una nuova lista q che contiene un nodo per ogni occorrenza che nella lista originaria appare più volte. Le chiavi della lista q devono risultare ordinate in modo decrescente.
Di seguito una lista p e la lista q che l'algoritmo deve restituire.



L'algoritmo deve avere complessità di tempo $O(n \log n)$ dove n è il numero di nodi presenti nella lista. Dell'algoritmo proposto:

a) si dia la descrizione a parole

b) si scriva lo pseudocodice

c) si giustifichi il costo computazionale

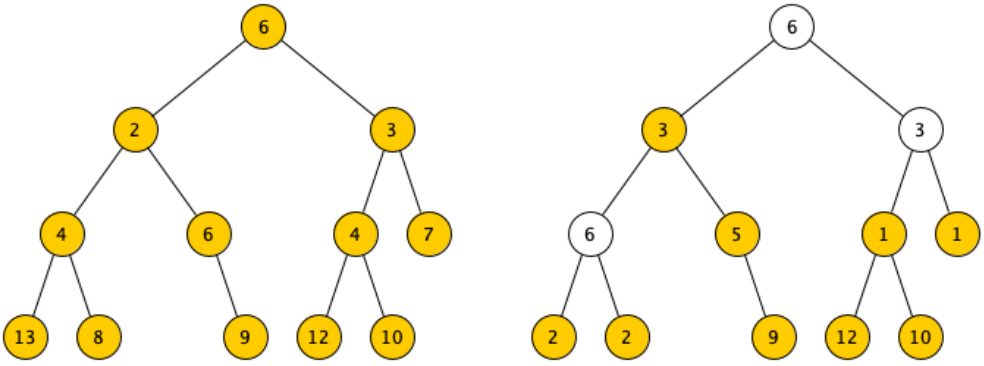
Scorro la lista p e inserisco le chiavi dei suoi nodi in una lista A d'appoggio. Ordino poi gli elementi della lista A in modo che le chiavi ripetute di p compaiano in A in locazioni adiacenti. Inizializzo la nuova lista q come vuota e scorro le chiavi in A, ogni chiave di A che incontro e che risulta la prima di chiavi ripeture la inserisco in testa alla lista q (in questo modo i duplicati compariranno in q in ordine decrescente). Al termine restituisco q.

Ecco di seguito una possibile implementazione in python dell'algoritmo

```
def es(p):
    A = []
    while p!= None:
        A.append(p.key)
        p = p.next
    A.sort()
    q = None
    for i in range(len(A)-1):
        if (A[i] == A[i+1]) and (i==0 or A[i-1]!=A[i]):
            q = Nodo(A[i],q)
    return q
```

Il $while$ viene iterato n volte ed ha dunque costo $\Theta(n)$. L'ordinamento di A (che contiene n chiavi) richiede tempo $\Theta(n \log n)$. Il for viene iterato meno di n volte e richiede quindi costo $O(n)$. La complessità dell'algoritmo è quindi $\Theta(n \log n)$.

Esercizio 2 (15 punti): Dato il puntatore r al nodo radice di un albero binario con nodi contenenti valori interi, progettare un algoritmo ricorsivo che in tempo $\Theta(n)$ determini il numero di nodi dell'albero che hanno entrambi i figli con lo stesso valore.
Ad esempio, per l'albero in figura a sinistra l'algoritmo deve rispondere 0 perché nessun nodo ha due figli con lo stesso valore. Per l'albero in figura a destra la risposta deve essere 3 grazie alla presenza dei 3 nodi evidenziati in bianco.



L'albero è memorizzato tramite puntatori a record di tre campi: il campo key contenente un valore intero ed i campi $left$ e $right$ con i puntatori al figlio sinistro e al figlio destro, rispettivamente (questi puntatori valgono $None$ in mancanza del figlio).
Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale risolvendo l'equazione di ricorrenza prodotta dal vostro algoritmo.

NOTA BENE: nello pseudocodice dell'algoritmo ricorsivo **non** si deve far uso di variabili globali.

Effettuo una visita ricorsiva dell'albero. Per il sottoalbero vuoto la risposta è zero. Ogni nodo restituisce al padre la soluzione per il suo sottoalbero vale a dire: il numero soluzione per il sottoalbero di sinistra più il numero soluzione per il sottoalbero di destra e più 1 solo se ha due figli che hanno lo stesso valore.

Ecco di seguito una possibile implementazione in python dell'algoritmo

```
def es(r):
    if r==None:
        return 0
    a= es(r.left)
    b= es(r.right)
    c=0
    if r.left!= None and r.right != None and r.left.key==r.right.key:
        c=1
    return a+b+c
```

Il costo computazionale è quello della visita di un albero con n nodi. L'equazione di ricorrenza relativa alla visita è:

- $T(n) = T(k) + T(n-1-k) + \Theta(1)$
- $T(0) = \Theta(1)$

dove k, è il numero di nodi presenti nel sottoalbero sinistro, $0 \leq k < n$. L'equazione si può risolvere con il metodo di sostituzione dando come soluzione $\Theta(n)$.
Di conseguenza il costo dell'algoritmo è, come richiesto, $\Theta(n)$.