Introduzione agli Algoritmi

Esame Scritto a canali unificati con spunti per la soluzione

docenti: T. Calamoneri, A. Monti

Sapienza Università di Roma

4 Giugno 2025

Esercizio 1 (10 punti): Si consideri la seguente funzione:

```
def fun (n):  
    i=0  
    for a in range(n+1):  
        for b in range(n+1):  
            i=i+1  
    if n<2 return i  
    return 2*fun(n/4)
```

Si scriva l'equazione di ricorrenza che ne definisce il costo computazionale, la si risolva con il metodo iterativo, e si discuta se sia possibile applicarvi il teorema principale o no, motivando bene la risposta.

C'è una sola chiamata su un quarto degli elementi e il tempo di esecuzione delle istruzioni al di fuori delle chiamate è $\Theta(n^2)$. Quindi la relazione di ricorrenza, nella sua forma generale, è $T(n) = T(n/4) + \Theta(n^2)$; il caso base rispetta

$T(0) = T(1) = \Theta(1)$. **Risolvendo con il metodo iterativo, si ottiene che $T(n) = \Theta(n^2)$.**

Questa soluzione è confermata dal teorema principale, di cui sfruttiamo il caso 3 e ricordiamo di dimostrare l'ipotesi aggiuntiva.

Esercizio 2 (10 punti): Sia dato un albero binario T , memorizzato tramite vettore dei padri. Per semplicità, assumiamo che gli n nodi di T siano etichettati con gli interi da 0 ad $n - 1$, così che sia già evidente la biezione tra ciascun nodo e il corrispondente indice. Si progetti un algoritmo **iterativo** che, dato l'array P , che rappresenta il vettore dei padri di T , restituisca il numero di foglie di T . Il costo dell'algoritmo proposto deve essere di $O(n)$, ed è consentito utilizzare $O(n)$ memoria ausiliaria.

Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato, in modo da chiarire il senso delle istruzioni,

b) si giustifichi formalmente il costo computazionale.

NOTA BENE: nello pseudocodice della funzione iterativa **non** si deve far uso di insiemi né di dizionari.

Osserviamo che, quando T è memorizzato tramite il vettore dei padri P , le foglie possono essere riconosciute in quanto non risultano essere padri di alcun nodo. Poiché in P non vi è alcun ordine preciso, una possibilità è quella di cercare all'interno di P tutti i nodi con questa proprietà. Tuttavia, il costo computazionale di questo algoritmo non potrà essere inferiore a $\Theta(n^2)$.

Un algoritmo più efficiente è una modifica del Counting Sort: sappiamo che le chiavi dei nodi vanno da 0 ad $n - 1$,

perciò possiamo usare un array ausiliario nel quale memorizzare la presenza di ciascun nodo in P . La risposta corrisponde al numero di indici il cui valore rimane a 0 nell'array ausiliario.

Un possibile codice Python della funzione descritta è il seguente:

```
def es2(P):
    n = len(P)
    # inizializzo i contatori dei figli dei vari nodi
    A = [0]*(n+1)
    for x in P:
        # incremento il numero di figli del nodo x
        A[x] += 1
    foglie = 0
    for i in range(n+1):
        # conto i nodi foglia dell'albero T
        if A[i] == 0:
            foglie += 1
    return foglie
```

E' possibile verificare formalmente che il costo computazionale è $O(n)$, così come la memoria ausiliaria utilizzata.

Esercizio 3 (10 punti):

Sia p il puntatore alla testa di una lista concatenata i cui nodi sono costituiti da un campo key contenente un valore intero ed un campo next contenente un puntatore al nodo successivo o *None* nel caso dell'ultimo nodo. Si progetti un algoritmo **ricorsivo** che, dato in input il puntatore p , cancelli dalla lista tutti i nodi con chiave negativa e restituisca il puntatore alla testa della nuova lista. L'algoritmo deve avere

costo computazionale $O(n)$ dove n è il numero di nodi che compongono la lista.

Ad esempio se la lista puntata da p è

$$p \rightarrow -4 \rightarrow -5 \rightarrow 2 \rightarrow 6 \rightarrow -3 \rightarrow 8 \rightarrow -7,$$

la lista restituita dall'algoritmo sarà: $p \rightarrow 2 \rightarrow 6 \rightarrow 8$.

Dell'algoritmo proposto:

- a) si dia la descrizione a parole,
- b) si scriva lo pseudocodice,
- c) si giustifichi formalmente il costo computazionale dando la ricorrenza che lo caratterizza e poi la sua soluzione.

NOTA BENE: nello pseudocodice della funzione ricorsiva **non** si deve far uso di variabili globali.

L'idea è semplicissima: se la lista è vuota si restituisce *None*, se la testa della lista ha come chiave un numero negativo, si applica la procedura al resto della lista e si restituisce il puntatore alla lista così ottenuta (così ottenendo di eliminare il primo nodo dalla lista); infine, se la testa della lista ha come chiave un numero positivo, si applica l'algoritmo al resto della lista, copiando il puntatore alla testa della lista modificata e restituito dalla chiamata ricorsiva nel campo *next* del primo elemento (così ottenendo di mantenere nella lista il primo nodo).

Ecco un possibile codice in Python:

```
def es3(p):
    if p == None:
        return None
    if p.key < 0:
        return es3(p.next)
    p.next = es3(p.next)
    return p
```

La relazione di ricorrenza che descrive il costo dell'algoritmo è quella che caratterizza la maggior parte degli algoritmi ricorsivi su liste puntate, cioè la seguente:

$$T(n) = T(n - 1) + O(1) \text{ se } n > 1, O(1) \text{ altrimenti}$$

la cui soluzione è $\Theta(n)$ e va risolta formalmente con uno dei metodi risolutivi studiati.