Introduzione agli Algoritmi
Esame Scritto a canali unificati con spunti per la soluzione

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
3 Aprile 2025

Esercizio 1: Si consideri il seguente algoritmo ricorsivo che, dato un numero intero n, calcola il massimo valore $k \geq 0$ tale che $2^k \leq n$.

```
def PotenzeDiDue(n):
    if n<=1:
        return 0
    else:
        return 1 + PotenzeDiDue(n//2)
```

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

Scriviamo l'equazione di ricorrenza, nel caso base e nel caso generale.
Se $n \leq 1$, la funzione restituisce 0. Ne segue il caso base:

$$T(0) = T(1) = \Theta(1).$$

Se $n > 1$, la funzione esegue delle operazioni costanti e richiama se stessa su un input di dimensione dimezzata.
Possiamo quindi scrivere l'equazione di ricorrenza come:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

dove:

- $T(n)$ **rappresenta il tempo di esecuzione per un input di dimensione** n**.**
- $T(n/2)$ **è il tempo di esecuzione per l'input ridotto a metà.**
- $\Theta(1)$ **è il costo delle operazioni costanti fuori dalla chiamata ricorsiva.**

La soluzione che si ottiene tramite il metodo dell'albero è $\Theta(\log n)$**, in quanto ogni nodo dell'albero ha un solo figlio, contribuisce con un costo** $\Theta(1)$ **e l'altezza dell'albero è logaritmica.**
Per quanto riguarda l'applicabilità del teorema principale, questo si può usare (per poterlo affermare si deve mostrare che valgono tutte le ipotesi), l'equazione ricade nel caso 2 e quindi viene confermata la soluzione già trovata.

Esercizio 2 (10 punti): Sia dato un array A di n interi. Si scriva un algoritmo **ricorsivo** che restituisca una lista puntata semplice contenente gli elementi di A nello stesso ordine.

Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi formalmente il costo computazionale, che dovrebbe essere $O(n)$.

L'osservazione chiave da fare per risolvere questo esercizio è che, per mantenere lo stesso ordine che gli elementi hanno nell'array e mantenere basso il costo computazionale, è necessario inserire in testa alla lista gli elementi a partire dall'ultimo andando verso il primo. Questo si può fare facilmente definendo una funzione ricorsiva che prende come parametri, oltre all'array e alla lista, anche la lunghezza corrente dell'array che stiamo considerando.
Più nel dettaglio, si devono considerare ricorsivamente i vari elementi dell'array. Se l'array A non ha elemento i-esimo viene restituito $None$, in caso contrario nell'i-esima chiamata ricorsiva viene creato e restituito un nodo contenente nel campo chiave il valore $A[i]$ e nel campo $next$ il puntatore alla testa di una lista che contiene i valori della sottolista di A che va dall'elemento $i+1$ alla fine.
Un possibile codice Python della funzione descritta è il seguente:

```
def esercizio2(A, i=0):
    if i== len(A):
        return None
    p = Nodo(A[i])
    p.next= esercizio2(A, i+1)
    return p
```

L'algoritmo richiama se stesso su un array che ha un elemento in meno. La ricorsione che cattura questo comportamento è quindi $T(n) = T(n-1) + \Theta(1)$ con $T(0) = \Theta(1)$ che, risolta ad esempio con il metodo iterativo, dà $\Theta(n)$. Questo risultato va mostrato formalmente, risolvendo l'equazione di ricorrenza.

Esercizio 3 (10 punti): Si consideri un albero binario di ricerca T in cui a ciascun nodo è associata una chiave numerica memorizzato tramite record e puntatori.

Scrivere un algoritmo efficiente che, dato in input il puntatore alla radice di T e due valori reali a e b, con $a < b$, restituisca *true* se e solo se TUTTE le chiavi di T sono comprese nell'intervallo $[a, b]$.

Dell'algoritmo proposto:

a) si dia la descrizione a parole,

b) si scriva lo pseudocodice,

c) si giustifichi formalmente il costo computazionale.

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

Possiamo osservare che, in ogni struttura dati, le chiavi sono ovviamente comprese tra la chiave minima e la chiave massima. In un albero binario di ricerca, trovare il minimo ed il massimo è particolarmente semplice. Pertanto, è immediato scrivere una funzione che determini questi due valori, e restituisca *true* se e solo se il minimo trovato è $\geq a$ ed il massimo trovato è $\leq b$, *false* altrimenti.
Il costo computazionale è dominato dal costo della ricerca di massimo e minimo (che è $O(h)$, dove h è l'altezza dell'albero), che è in $O(n)$, come richiesto. Vanno specificati i contributi di ogni istruzione.