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

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

9 Febbraio 2026

Esercizio 1 (10 punti): Sia dato un albero binario completo di altezza h, e si risponda alle seguenti domande:

a) Qual è l'equazione di ricorrenza che descrive il costo computazionale di una visita in-ordine su T in funzione di h? Perché?

b) Risolvere l'equazione di ricorrenza dettagliando i calcoli; usare il metodo iterativo se la vostra matricola ha l'ultima cifra dispari ed il metodo dell'albero altrimenti.

c) Esprimere il risultato anche in funzione di n, esplicitando le considerazioni fatte per arrivare al risultato.

a) Assumendo che un albero vuoto abbia altezza 0 ed un albero con la sola radice abbia altezza 1, si può sfruttare la definizione ricorsiva degli alberi binari di altezza h (fatti di una radice e due sottoalberi di altezza al più $h-1$), per cui la ricorrenza è:

$$T(h) = \begin{cases} \Theta(1) & \textbf{se } h \leq 1, \\ 2Th-1) + \Theta(1) & \textbf{se } n > 2. \end{cases}$$

b) Risolvendo l'equazione con uno dei metodi richiesti si ottiene che $T(h) = \Theta(2^h)$.
c) Si osservi che per T vale che $n = \Theta(2^h)$ ed $h = \Theta(\log n)$, ritrovando il familiare risultato che il costo della visita sia $\Theta(n)$.

Esercizio 2 (10 punti): Sia dato un array A di dimensione n in cui ogni elemento è una stringa "(" o ")". Si progetti un algoritmo **iterativo** che risponda True se A rappresenta una sequenza di parentesi bilanciate, ossia se l'ordine di apertura e chiusura delle parentesi è corretto rispetto a come di solito si usano nelle espressioni aritmetiche, False altrimenti.

Ad esempio: dando come input l'array ["(","(",")","(","(",")",")",")"], l'algoritmo deve restituire True, mentre dando ["(",")",")","(","("] deve restituire False.

Dell'algoritmo proposto:

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

b) sia dia un esempio di esecuzione quando l'input risulti l'array $A = [\text{"("}, \text{"("}, \text{")"}, \text{"("}, \text{")"}]$,

c) si giustifichi il costo computazionale.

a. **Poichè le uniche parentesi ammesse sono quelle tonde, non c'è bisogno di fare alcun controllo se non che ad ogni parentesi aperta ne deve corrispondere una chiusa. Una tale verifica si può fare usando un semplice contatore:**

```
def ParentesiBilanciate(A):    # A è un array di stringhe
  contatore = 0;
    # contatore conta la differenza tra parentesi aperte e chiuse
```

```
for i in range n:
   if a[i] =="(": contatore++
   else:
      if contatore>0: contatore--
      else: return false;
if contatore==0: return true
  #true solo se le parentesi sono bilanciate
else: return false
```

b. **Nel caso di $A = [\text{"}(\text{"}, \text{"}(\text{"}, \text{"})\text{"}, \text{"})\text{"}, \text{"})\text{"}]$, $n = 5$. Entriamo nella funzione con contatore=0; sia per $i = 0$ che per $i = 1$ il contatore viene incrementato di 1 raggiungendo quindi il valore di 2; ciò significa che in questo momento ci sono due parentesi aperte in più rispetto alle chiuse. Quando $i = 2$, la stringa rappresenta una parentesi chiusa e il contatore viene decrementato: è solo una la differenza tra le parentesi aperte e e le chiuse. Si passa ad $i = 3$ e si trova un'altra parentesi chiusa, che fa annullare il contatore. Infine, quando $i = 4$, l'ultima parentesi chiusa fa diventare il contatore negativo. Perciò, all'uscita del ciclo for, il valore restituito sarà False.**

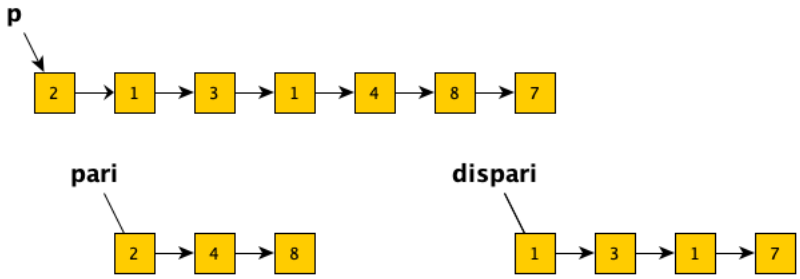
c. **Il costo computazionale è dato dalla somma dei contributi delle singole istruzioni. È facile vedere che esso è $\Theta(n)$. È richiesto di calcolare il costo formalmente.**

Esercizio 3 (10 punti): Sia data una **lista concatenata** memorizzata tramite record e puntatori. Ogni nodo della lista contiene un campo val, che memorizza un intero, ed un puntatore $next$ al nodo successivo.

Si scriva un algoritmo **ricorsivo** che, data la testa della lista p restituisca la tupla $(pari, dispari)$ dove $pari$ e $dispari$ sono le teste di due liste concatenate definite come segue:

- la lista $pari$ contiene tutti e soli i nodi della lista originale il cui campo val è $pari$;
- la lista $dispari$ contiene tutti e soli i nodi della lista originale il cui campo val è $dispari$.

In entrambe le liste i nodi devono comparire nello stesso ordine in cui apparivano nella lista originale.
I nodi della lista originale devono essere riutilizzati: non è consentito creare nuovi nodi, ma solo modificare opportunamente i puntatori
In figura è mostrato un esempio di una lista p e delle due liste restituite dall'algoritmo.



L'algoritmo deve avere costo computazionale $O(n)$ dove n è il numero di nodi della lista concatenata originaria.

Dell'algoritmo proposto:

a) si descriva a parole il funzionamento dell'algoritmo;
b) si scriva lo pseudocodice;

c) si giustifichi il costo computazionale scrivendo la ricorrenza che lo caratterizza.

NOTA BENE: La funzione proposta **non** deve far uso di variabili globali.

a. **L'idea dell'algoritmo è la seguente: se la lista è vuota, si restituisce** $(None, None)$**. Altrimenti, sia** p **il puntatore al nodo di testa della lista concatenata. Si deve applicare ricorsivamente la funzione che sposta i nodi dalla coda alla testa, quindi alla lista** $p.next$**. In questo modo si ottiene una coppia (**$pari, dispari$**) contenente le due liste concatenate ottenute della lista originaria senza il primo elemento. Come ultimo passo, si deve gestire il nodo in testa alla lista originaria: se il valore** $p.val$ **è pari, il nodo** p **viene aggiunto in testa alla lista** $pari$**; altrimenti viene inserito in testa alla lista** $dispari$**. In entrambi i casi si restituisce la coppia aggiornata (**$pari, dispari$**).**

b. **Ecco un possibile pseudocodice:**

```
def es3(p)
    if p is None:
        return (None, None)
    pari, dispari = es3(p.next)
    if p.val %2==0:
        p.next = pari
        return (p, dispari)
    else:
        p.next = dispari
        return (pari, p)
```

c. Il costo computazionale è quello di una visita ricorsiva della lista. Ogni chiamata ricorsiva esegue un numero costante di operazioni: un confronto, un'assegnazione di puntatori e una chiamata ricorsiva. La relazione di ricorrenza è quindi:

$$T(n) = T(n-1) + O(1)$$

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