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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
5 Luglio 2023

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

```
def Es1(n):
    s = 0
    if n > 4:
        m = n//2
        s = Es1(n//4) * Es1(m - n//4)
        i = 1
        while i * i <= n:
            i += 1
            s += i
    return s
```

a) Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione giustificando dettagliatamente l'equazione ottenuta.

b) Qualora sia possibile, si risolva la ricorrenza utilizzando il teorema principale, dettagliando il caso del teorema ed i passaggi logici. Se il teorema principale non è applicabile spiegarne il motivo.

a) **Il caso base si ha quando $n \leq 4$, ed ha costo costante. La funzione ricorsiva su input $n > 4$ richiama se stessa 2 volte su un'istanza di dimensione $n/4$. Il ciclo $while$ viene eseguito $\Theta(\sqrt{n})$ volte. Perciò, la relazione di ricorrenza da risolvere è:**

$$T(n) = 2T\left(\frac{n}{4}\right) + \Theta\left(\sqrt{n}\right).$$

b) **Per applicare il metodo principale notiamo che $n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}}$ mentre $f(n) = \Theta(\sqrt{n}) = n^{\frac{1}{2}}$ siamo dunque nel secondo caso del teorema principale e si ha $T(n) = (\sqrt{n}\log n)$.**

È necessario dettagliare tutti i passaggi per arrivare all'equazione di ricorrenza e per risolverla.

Esercizio 2 (10 punti):
In un array ordinato A di n interi compaiono tutti gli interi da 0 ad $n-2$. Esiste dunque nell'array un unico elemento duplicato.

Si progetti un algoritmo ITERATIVO che, dato A, in tempo $\Theta(\log n)$ restituisca l'elemento duplicato.

Ad esempio, per $A = [0, 1, 2, 3, 4, 4, 5, 6, 7]$ l'algoritmo deve restituire 4.

Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;

b) si giustifichi il costo computazionale.

Osserviamo che l'elemento duplicato è il primo la cui chiave non corrisponde al proprio indice. Possiamo quindi ricorrere alla ricerca binaria (ITERATIVA, come richiesto): dati gli indici i e j dell'intervallo in cui si ricerca il duplicato, ci poniamo nella posizione centrale m e, se risulta $A[m] = m$ allora si va a cercare nel sottointervallo di destra (da $m+1$ a j), se invece risulta $A[m] > m$ si cerca nel sottointervallo di sinistra (da i a m). Ci si ferma quando resta un

unico elemento.

Segue un possibile codice Python dell'idea proposta.

```
def es2(A):
    i, j = 0, len(A) - 1
    while i < j:
        m = (i + j)//2:
        if A[m] == m:
            i = m + 1
        else:
            j = m
    return A[i]
```

Il costo dell'algoritmo è quello della ricerca binaria che è $\Theta(\log n)$, ma è necessario dettagliarne il calcolo.

Esercizio 3 (10 punti): Data una lista puntata, diremo che i due record che occupano le posizioni $(2i-1)$-esima e $2i$-esima ($i \geq 1$) vengono *accorpati* eliminando dalla lista quello in posizione pari, e trasformando la chiave del record in posizione dispari nella somma dei due elementi accorpati. Se lista ha un numero dispari di nodi, allora l'ultimo nodo, che non può accorparsi, viene semplicemente eliminato.

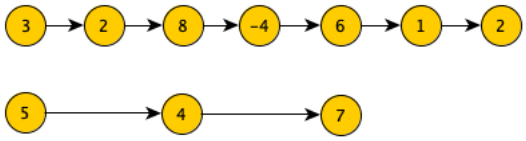
Sia dato il puntatore p al nodo di testa di una lista puntata memorizzata tramite puntatori a nodi a due campi: il campo key contenente il valore ed il campo $next$ contenente il puntatore al nodo successivo (il campo $next$ dell'ultimo nodo della lista vale $None$).

Si progetti un algoritmo RICORSIVO che, in tempo $\Theta(n)$, ne accorpi a coppie tutti i suoi nodi (il primo con il secondo, il terzo col quarto ecc.).

Ad esempio, la lista seguente va modificata come in figura.

Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato;



b) si giustifichi il costo computazionale.

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

Ecco un possibile codice Python dell'algoritmo che non utilizza variabili globali:

```
def es3(r):
    # se la lista è vuota o composta da un unico elemento
    # bisogna restituire None
    if p == none or p.next == None:
        return None
    # vanno accorpati i primi due nodi della lista
    # e il campo next del nodo rimasto deve puntare
    # alla lista accorpata degli elementi dal terzo in poi
    p.key+ = p.next.key
    p.next = es3(p.next.next)
    return p
```

Il costo computazionale sarà quello della visita di una lista con n nodi. In particolare, l'equazione di ricorrenza è:

- $T(n) = T(n-2) + \Theta(1)$
- $T(0) = \Theta(1)$

L'equazione si può risolvere con uno dei metodi studiati, escluso quello principale; i calcoli vanno esplicitamente svolti, e la soluzione è $\Theta(n)$.