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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
21 Gennaio 2025

Esercizio 1 (10 punti):

Supponiamo di disporre di un processore speciale in grado di fondere due array ordinati, aventi complessivamente n elementi, in tempo $O(\sqrt{n})$.

Si determini il costo computazionale di un algoritmo di ordinamento, che chiamiamo NEW-MERGESORT, che sia identico all'algoritmo di ordinamento MERGESORT classico, salvo che la fase di fusione dei sotto-array ordinati viene realizzata mediante il processore speciale di cui sopra.

Si scriva l'equazione di ricorrenza di NEW-MERGESORT, spiegandola in dettaglio, e la si risolva con uno dei metodi studiati, specificando i pasaggi.

L'equazione di ricorrenza è $T(n) = 2T(n/2) + \Theta(\sqrt{n})$, poiché la parte non ricorsiva è l'unica ad essere modificata. Il caso base rimane $T(1) = \Theta(1)$.
L'equazione può essere risolta ad esempio con il metodo iterativo, trovando così la soluzione $\Theta(n)$. Infatti, il termine $\Theta(\sqrt{n})$ può essere portato fuori dalla sommatoria, e l'esponente del 2 può essere permutato con la radice quadrata.

Esercizio 2 (10 punti): Sia dato un array E di n coppie di interi del tipo (i, e), in cui il primo elemento di ciascuna coppia i è un indice tra 0 ed $n-1$ assegnato ad una persona, ed il secondo elemento e rappresenta l'età della persona di indice i; sia poi data una matrice quadrata e simmetrica A di dimensione $n \times n$ a valori 0 ed 1 in cui $A[j,k] = A[k,j]$ é uguale ad 1 se e solo se la persona di indice j conosce la persona di indice k.

Si scriva un algoritmo il più efficiente possibile che verifichi se esiste almeno una coppia di conoscenti coetanei. L'algoritmo dovrebbe dare in output una coppia con gli indici dei due conoscenti coetanei se ce ne sono, oppure la coppia $(0, 0)$.

Ad esempio, sia dato il seguente array E: $\begin{array}{cccc} 2 & 1 & 3 & 0 \\ \hline 35 & 31 & 35 & 35 \end{array}$ e la matrice A:

	0	1	2	3
0	0	1	0	1
1	1	0	1	0
2	0	1	0	0
3	1	0	0	0

L'output è la coppia $(0, 3)$ in quanto queste due persone si conoscono ed hanno la stessa età; si osservi che le coppie $(0, 2)$ e $(1, 2)$ non sono soluzioni valide, infatti le persone nella prima coppia hanno la stessa età ma non si conoscono, mentre quelle nella seconda coppia si conoscono ma non hanno la stessa età.

Si scriva lo pseudocodice opportunamente commentato dell'algoritmo progettato e se ne calcoli formalmente il costo asintotico.

Una prima soluzione prevede l'esame della matrice e, per ogni coppia di indici (j, k) per cui $A[j,k] = 1$, il controllo sull'età in E. Il costo è $\Theta(n^2)$ nel caso peggiore (in cui non

si trova alcuna coppia e la matrice deve essere visitata per intero) e $\Theta(1)$ nel migliore (in cui già la prima coppia di indici è una soluzione valida).

Un'altra soluzione consiste nell'ordinare E rispetto alle età, così individuando i duplicati e controllando nella matrice se si conoscono solo le persone associate agli indici corrispondenti alle età duplicate. Quindi il costo computazionale dipende dalle coppie di coetanei, che può essere costante nel caso migliore e quadratico in n (per esempio se sono tutti coetanei e non si conoscono) nel caso peggiore.
Per quanto riguarda l'ordinamento da usare, con un algoritmo per confronti efficiente si avrebbe un tempo di esecuzione in $\Theta(n \log n)$, da aggiungere al tempo di verifica già discusso, ottenendo un costo compreso tra $\Omega(n \log n)$ ed $O(n^2)$. Se però calcoliamo le età minima, m, e massima, M, possiamo considerare l'input compreso nell'intervallo $[m, M]$ e applicare il countingsort. Assumendo l'età massima costante, l'ordinamento ora è lineare nel numero delle persone, e l'algoritmo ha costo $\Omega(n)$ ed $O(n^2)$. Tuttavia, con questo algoritmo abbiamo introdotto un array aggiuntivo che, quando possibile, andrebbe evitato.

Tutte le soluzioni proposte hanno medesimo costo nel caso peggiore e quindi sono ugualmente valide. Ecco uno pseudocodice:

```
Algoritmo verificaConoscentiCoetanei(E, A):
    // Ordinare l'array E per età usando Heap Sort
    HeapSort(E)

    // Scorrere l'array ordinato per trovare coppie di coetanei che si conoscono
```

```
per ogni i da 0 ad n-2:
    per ogni j da i+1 a n-1:
        se E[i].età == E[j].età:
            se A[E[i].indice][E[j].indice] == 1:
                return (E[i].indice, E[j].indice)
return (0, 0)
```

Esercizio 3 (10 punti): Sia T un albero binario radicato memorizzato tramite record e puntatori. L'albero è non vuoto e ogni nodo contiene un valore intero come chiave. Definiamo il *costo di un cammino radice-foglia* come la somma delle chiavi dei nodi che compongono il cammino.

Si progetti un algoritmo **ricorsivo** per trovare il valore minimo del costo di un cammino radice-foglia in un albero T dato in input tramite il puntatore alla sua radice.
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'algoritmo prevede di visitare l'albero in post-ordine. Durante la visita l'algoritmo raccoglie informazioni sui costi minimi dei suoi due sottoalberi. Per ogni nodo, il costo minimo viene calcolato combinando la sua chiave con il minimo tra i costi calcolati per i figli. Come caso base il costo delle foglie è dato dal loro valore. Alla fine, la radice restituirà il costo minimo complessivo del cammino.

Il codice Python della procedura appena descritta è il seguente:

```
def esercizio3(r):
    # Caso base: se il nodo è una foglia, il costo è il suo valore
    if r.left is None and r.right is None:
        return r.key
    # Inizializza i costi dei sottoalberi a infinito
    sinistra = float('inf')
    destra = float('inf')
    # Calcola il costo minimo dei cammini nei sottoalberi
    if r.left is not None:
        sinistra = esercizio3(r.left)
    if r.right is not None:
        destra = esercizio3(r.right)
    # Restituisce il cammino di costo minimo
     return r.key + min(sinistra, destra)
```

La relazione di ricorrenza che descrive il costo dell'algoritmo è la seguente:

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

con $0 \leq k < n$**, che risolta da** $\Theta(n)$ **e va risolta formalmente.**