Complementi di Algoritmi e Strutture Dati

(III anno Laurea Triennale - a.a. 2016/17)

Prova scritta 29 giugno 2017

NB: I punteggi sono indicativi.

Esercizio 1 - Tabelle hash (Punti 5) Supponiamo di usare una tabella hash di $m = 11$ caselle (numerate da 0 a 10) e rehashing con la seguente famiglia di funzioni:

$$f(k, i) = (k + i) \mod m$$

Supponiamo inoltre che la tabella abbia inizialmente occupati i posti di indice $1, 4, 5, 6$ (con elementi di chiave $11, 4, 15, 6$) e liberi gli altri posti. Graficamente, la situazione è questa:

indice	0	1	2	3	4	5	6	7	8	9	10
contenuto		11			4	15	6				

1. Inserire nella tabella l'elemento di chiave $k = 16$. Dire se ci sono collisioni (in caso affermativo dire quali collisioni ci sono) e mostrare la tabella dopo l'inserimento.
2. Successivamente, cancellare dalla tabella l'elemento di chiave $k = 15$. Dire quali caselle vengono accedute e mostrare la tabella dopo la cancellazione.
3. Successivamente, eseguire una ricerca con chiave $k = 26$. Dire quali caselle vengono accedute e dire il risultato della ricerca (trovato oppure non trovato).
4. Inserire infine un elemento di chiave $k = 26$. Dire se ci sono collisioni, nel caso quali, e mostrare la tabella dopo l'inserimento.

Nota bene: ogni operazione va eseguita sul risultato delle operazioni precedenti.

Esercizio 2 - Sorting (Punti 8) Consideriamo una versione di Quicksort che lavora su una lista linkata s invece che su un array.
Sulla lista supponiamo di avere, implementate con costo costante, le classiche operazioni:
- *isEmpty()* – test se vuota,
- *deleteFirst()* – ritorna primo elemento cancellandolo,
- *addToEnd(int x)* – inserisce elemento x in fondo.

La scelta del pivot consiste nel prendere (togliendola) la testa della lista s:
  $p =$deleteFirst(s);
Il partizionamento consiste nello scorrere (quel che rimane del)la lista s ed ogni volta trasferire la testa in s_1 se minore del pivot e in s_2 se maggiore o uguale:
  // dalla lista s ho già tolto il pivot p
  s_1 = new list(); // qui metterò elementi $< p$
  s_2 = new list(); // qui metterò elementi $\geq p$
  while (NOT s.isEmpty())
    $x = s$.deleteFirst();
    if $x < p$ s_1.addToEnd(x);
    else s_2.addToEnd(x);
Ovviamente vengono poi eseguite le chiamate ricorsive su s_1 e s_2 e la lista risultato (ordinata) è la concatenazione $s_1 \cdot p \cdot s_2$.

Si chiede di:

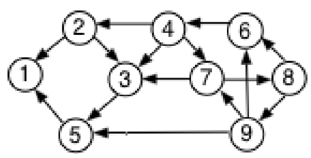
1. Far vedere che il partizionamento termina ed è corretto
2. Questo partizionamento ha complessità tempo lineare in n (lunghezza della lista)? Perchè? Che cosa si può dire riguardo alla complessità spazio?
3. Simulare l'esecuzione dell'intero algoritmo **Quicksort** (in questa versione) su una lista linkata che contiene nell'ordine gli elementi

   $$[2, 6, 8, 2, 1, 5, 7, 4, 5, 3, 9, 0].$$

   Disegnare l'albero delle chiamate ricorsive. Per ogni chiamata indicare:

   - la lista di input
   - se non si tratta di un caso base (foglia nell'albero delle chiamate) indicare il pivot p (si intende che le due sotto-liste s_1, s_2 saranno le liste di input dei due nodi figli sinistro e destro).
   - la lista risultato (parziale)

Esercizio 3 - Grafi (Punti 7) Si esegua, sul seguente grafo:



l'algoritmo per il calcolo delle componenti connesse. In particolare, si diano:

- i tempi di inizio e fine visita ottenuti per ogni nodo in seguito alla visita in profondità (si effettui la visita a partire dal nodo 8 e si considerino gli adiacenti di un nodo in ordine numerico crescente)
- la sequenza delle componenti fortemente connesse $\texttt{Ord}^{\leftrightarrow}$ ottenuta
- il grafo quoziente.

Esercizio 4 - Design e analisi di algoritmi (Punti 8) Un distributore di bibite contiene al suo interno n monete i cui valori (interi positivi) sono memorizzati in un array $c[1..n]$. Si consideri il problema di decidere se è possibile erogare un resto esattamente uguale a R (intero positivo) utilizzando un opportuno sottoinsieme delle n monete a disposizione. Sia $P(k, r)$ ($1 \leq k \leq n, 0 \leq r \leq R$) il sottoproblema di decidere se è possibile erogare un resto esattamente uguale a r utilizzando le monete $\{1, \ldots, k\}$. Si noti che, a differenza del problema del resto visto a lezione, non è necessario erogare il resto con il numero minimo di monete.

1. Dare una definizione induttiva di $P(k, r)$, giustificandone la correttezza.
2. Descrivere un algoritmo di programmazione dinamica per risolvere il problema.
3. Determinare il costo computazionale dell'algoritmo descritto, motivando la risposta.
4. Spiegare cone modificare l'algoritmo per determinare anche quali sono le monete da erogare.

Esercizio 5 - NP-completezza (Punti 6) Si consideri il seguente problema TWO-CLIQUE: dato un grafo G e due interi $h, k > 0$, determinare se G contiene due clique disgiunte di dimensione rispettivamente h e k.

1. Provare che TWO-CLIQUE è in NP.
2. Provare che TWO-CLIQUE è NP-hard.