Algoritmi e Principi dell'Informatica

Appello del 27 giugno 2017

II prova: durata 1 ora e 15 minuti.

Esercizio 1 (7 punti)

Si consideri il linguaggio $L = \{x.y.x^R.x.y.x^R \mid x \in \{a, b\}^+, y \in \{a, b, c\}^*\}$. Si descrivano una MT a k nastri e una macchina RAM che accettano L, valutandone le complessità temporali e spaziali, a criterio di costo costante e logaritmico.

Esercizio 2 (9 punti)

Un ipotetico nucleo di valutazione del sistema universitario di un'ipotetica nazione ha deciso di valutare i docenti universitari utilizzando la seguente procedura:

- A ciascun docente *i* è associato un indice di produttività IP_i (un valore reale non negativo).
- I docenti sono ordinati in modo che i loro IP siano in ordine non decrescente.
- Il nucleo di valutazione definisce una soglia, cioè un numero non negativo *k*, per cui avanzeranno in carriera i soli docenti con un IP strettamente maggiore di *k*.

Si scriva un algoritmo che, dato il vettore ordinato IP[1..n] e la soglia k, calcola **il numero totale di ricercatori che possono essere promossi.** Se ne discuta la complessità. La valutazione terrà conto della complessità dell'algoritmo definito.

Soluzioni

Esercizio 1

Per prima cosa notiamo come per verificare che una sottostringa abbia la forma $x.y.x^R$ basti controllare che il primo e l'ultimo carattere siano uguali e in {a, b}.

Una MT M può riconoscere L calcolandone prima la lunghezza in unario sul primo nastro, per poi sfruttarne il contenuto per copiare la prima metà della stringa in ingresso sul secondo nastro e la seconda metà sul terzo nastro. La passata finale controlla che il secondo e il terzo nastro contengano la stessa stringa, che deve iniziare e terminare con lo stesso carattere a o b.

Chiaramente $T_M(n) = \Theta(n) = S_M(n)$.

Una macchina RAM R copia in memoria la stringa in ingresso, calcolandone la lunghezza (poi semilunghezza, dividendo per 2) con un contatore in M[1]. A questo punto usa 2 indici (es. M[2] e M[3]) per confrontare le due semistringhe, controllando anche che inizino e finiscano con lo stesso simbolo a o b.

A costo costante $T_R(n) = \Theta(n) = S_R(n)$.

A costo logaritmico $T_R(n) = \Theta(n \log n)$; $S_R(n) = \Theta(n)$.

Esercizio 2

Una soluzione semplice, di complessità O(n), scandisce il vettore PI e si ferma al primo elemento con valore maggiore di *k* (il vettore è ordinato). Tale algoritmo ha complessità Θ(n) solo nel caso in cui tutti i docenti abbiano IP minore di k (caso pessimo).

Una soluzione più efficiente sfrutta la *ricerca binaria* per contare i valori di IP strettamente maggiori di k. Si usa quindi una funzione ricorsiva ispirata alla ricerca binaria, che chiamiamo *SopraSoglia(IP, k, i, j)*, che a ogni passo ricorsivo restituisce il numero di elementi maggiori di *k* nella partizione considerata:

```
int SopraSoglia(float IP[1..n], float k, int i, int j ) {
  if ( i > j )
        return 0;                    //  partizione vuota
  int m = ( i + j ) / 2;
  if (IP[m] ≤ k )
       return SopraSoglia( IP, k, m+1, j );  // esamino la partizione
                                a  destra  di  IP[m]  (gli  elementi
                                precedenti sono tutti ≤ k)
  else
      return (j-m+1) + SopraSoglia( IP, k, i, m-1 ); // elementi di
                                     IP[m…j]  tutti  sopra  soglia;
                                     verifico la partizione sx
}
```

Così come per la ricerca binaria, la complessità di questo algoritmo si può esprimere con l'equazione di ricorrenza T(n) = T(n/2) + O(1), che ha soluzione T(n) = O(log n)