008AA – ALGORITMICA E LABORATORIO

Appello del 20 luglio 2011: Soluzioni

Esercizio 1.
Si scriva una funzione `StrConf()` che, date in input due stringhe, restituisca -1 se la prima è più corta della seconda, 0 se sono della stessa lunghezza, e +1 se la prima è più lunga della seconda. Non si usi alcuna funzione di libreria per il calcolo della lunghezza di una stringa.

Soluzione.

```
int StrConf(char * str1, char* str2)
{
   int i, j;
   int c1 = 0, c2 = 0;
   for (i = 0; *(str1+i) != '\0'; i++) c1++;
   for (J = 0; *(str2+j) != '\0'; j++) c2++;
   if (c1 < c2)  return -1;
   if (c1 == c2) return 0;
   else return 1;
```

Esercizio 2.
Sia data la seguente funzione

```
Mistero(n)
{
   if (n < 1) return 0;
   m = Sqrt(n); // m prende la parte intera della radice quadrata di n
   x = 0;
   for (i = 0; i < m; i++)  {
       x = x + 3;
   }
   return x + Mistero(n/4) * Mistero(n/4);
}
```

1. Indicare e risolvere l'equazione di ricorrenza che esprime la complessità in tempo al caso pessimo della funzione `Mistero`.
2. Scrivere una semplice alternativa al codice della funzione `Mistero` che restituisca lo stesso valore, ma abbia un ordine di complessità provatamente inferiore. Fornire la relativa analisi di complessità.

Soluzione.

1. $T(n) = 2\,T(n/4) + O(\sqrt{n})$.

   La soluzione della ricorrenza è $T(n) = O(\sqrt{n}\log n)$ (Teorema fondamentale, secondo caso).

2.
```
Mistero(n)
{
   m = Sqrt(n); // m prende la parte intera della radice quadrata di n
   x = 0;
   for (i = 0; i < m; i++)  {
       x = x + 3;
   }
   g = F(n/4)
   return x + g * g;
}
```

$T(n) = T(n/4) + O(\sqrt{n})$.

La soluzione della ricorrenza è $T(n) = O(\sqrt{n})$ (Teorema fondamentale, terzo caso).

Esercizio 3.
Date le funzioni hash

$$h_1(k) = k \bmod 13 \qquad h_2(k) = (k \bmod 3) + 1\,,$$

si consideri una tabella hash (di dimensione $m = 13$ e inizialmente vuota) a indirizzamento aperto con hash doppio

$$h(k,i) = (h_1(k) + i * h_2(k)) \bmod 13\,.$$

Mostrare il contenuto della tabella dopo l'inserimento delle chiavi 8, 3, 9, 4, 2, 16, 5, 15.

Soluzione.

$$\begin{aligned} h(k,i) &= (h_1(k) + i * h_2(k)) \bmod 13 \\ &= (k \bmod 13 + i * ((k \bmod 3) + 1)) \bmod 13\,. \end{aligned}$$

Chiave	**sequenza**
8	8
3	3
9	9
4	4
2	2
16	3, 5
5	5, 8, 11
15	2, 3, 4, 5, 6

-	-	2	3	4	16	15	-	8	9	-	5	-

Esercizio 4.
È dato un grafo orientato $G = (V, E)$, i cui nodi sono *pesati* (ossia contengono valori interi memorizzati in un campo `peso`), e *colorati* (ossia hanno due possibili colori, **rosso** e **nero**, memorizzati in un campo `colore`). Dati due vertici x, y, e un intero k, si deve stabilire se esiste un cammino da x a y i cui vertici sono **tutti rossi** e di peso ciascuno **minore o uguale** a k.

1. Dare una realizzazione dell'algoritmo in pseudocodice.
2. Valutare la complessità dell'algoritmo proposto.

Soluzione.
Utilizziamo una visita BFS, con la seguente variazione: quando si visita un vertice u si considerano solo i vertici nella sua lista di adiacenza che sono di colore rosso e di peso minore o uguale a k. La visita è invocata sul vertice x. Se alla fine della vista il vertice y è stato raggiunto, la risposta dell'algoritmo è TRUE, altrimenti FALSE. Il problema si può risolvere anche modificando la visita DFS.

Cammino_Rosso_Pesato(G, x, y, k)

```
∀ v ∈ V    v.marca = inesplorato;
F = nuova Coda();
x.marca = scoperto;
F.enqueue(x);
while (F ≠ ∅) {
   u = F.dequeue();
   ∀ v nella lista di adiacenza di u {
      if (v.marca == inesplorato && v.colore == rosso && v.peso <= k) {
          v.marca = scoperto;
          F.enqueue(v);
      }
   }
}
if (y.marca == scoperto) return TRUE;
else return FALSE;
```

Complessità: $T(|V|, |E|) = O(|V| + |E|)$.