**008AA – ALGORITMICA E LABORATORIO**

Appello del 6 giugno 2011: Soluzioni

**Esercizio 1.**

Sia dato un array $A[0, n-1]$ di stringhe (visto come `char **A`). Scrivere una funzione C che riceve in input l'array $A$ e la sua lunghezza $n$, e crea un altro array $B$ contenente tutte le stringhe di $A$ ***senza ripetizioni***. Ad esempio se $A = \{\texttt{paperino}, \texttt{pippo}, \texttt{pluto}, \texttt{pippo}\}$, deve risultare $B = \{\texttt{paperino}, \texttt{pippo}, \texttt{pluto}\}$.

**Soluzione.**

```
char ** eliminaDuplicatiStringhe(char **a, int size) {
   char ** b=malloc(size*sizeof(char*));
   int i=0;
   int j;
   int pos=1;
   int flag=0;
   int k;

   b[0] = a[0];

   for (j=1;j<size;j++) {
      while (i<pos && !flag) {
         if (strcmp(a[j],b[i]) == 0)  flag = 1;
         else i++;
      }
      if (!flag) {
         b[pos] = a[j];
         pos++;
         i=0;
      }
      else {
         flag=0;
         i=0;
      }
   }
   return b;
}
```

**Esercizio 2.**

Si consideri la seguente funzione:

```
F(n)
{
   if (n == 0) return 0;
   if (n == 1) return 1;
```

```
   else return F(n-1) + F(n-2) + 1;
}
```

1. Indicare il risultato della chiamata $F(10)$.
2. La complessità di $F$ è esponenziale rispetto alla dimensione dell'input. Progettare una funzione equivalente ad $F$ ma di di complessità $O(n)$.

**Soluzione.**

1. $F(10) = 143$.
2. ```
   F(n)
   {
      if (n == 0) return 0;
      if (n == 1) return 1;
      a = 0;
      b = 1;
      for (i = 2; i <= n; i++) {
         c = a + b + 1;
         a = b;
         b = c;
      }
      return c;
   }
   ```

**Esercizio 3.**
Cosa fa la procedura seguente? Qual è la sua complessità?

```
Mistero(a, sx, dx)
{
   if (sx == dx) return a[sx];
   cx = (sx + dx)/2;
   m1 = Mistero(a, sx, cx);
   m2 = Mistero(a, cx+1, dx);
   return m1 + m2;
}
```

**Soluzione.** La procedura è di tipo *Divide et Impera* e calcola la somma degli elementi contenuti nell'array `a`. La sua complessità è $O(n)$ e deriva dalla soluzione della seguente equazione di ricorrenza:

$$\begin{aligned} T(n) &= cost. && n = 1 \\ T(n) &= 2T(n/2) + cost. && n > 1 \end{aligned}$$
```

**Esercizio 4.**
Progettare un algoritmo che, dato un grafo $G = (V, E)$ non orientato realizzato con liste di adiacenza e dati tre vertici $x, y, z \in V$, restituisce **1** se $x, y$, e $z$ appartengono alla stessa componente connessa; **2** se $x, y$, e $z$ appartengono a due diverse componenti connesse; **3** se $x, y$, e $z$ appartengono a tre diverse componenti connesse.

1. Dare un programma in pseudocodice per l'algoritmo proposto.
2. Discutere la complessità dell'algoritmo proposto.

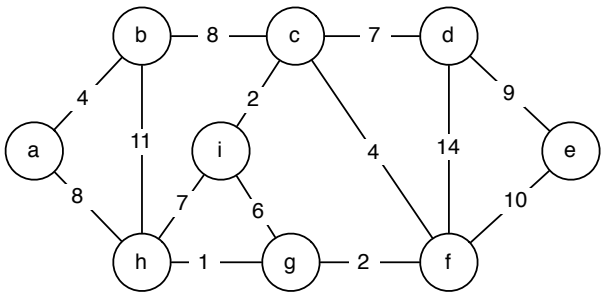
**Soluzione.**

**ContaComponentiConnesse**(G,x,y,z)

```
∀ v ∈ V   v.marca = inesplorato;
BFS(G, x);
if(y.marca != inesplorato && z.marca != inesplorato) return 1;
if(y.marca != inesplorato && z.marca == inesplorato) return 2;
if(y.marca == inesplorato && z.marca != inesplorato) return 2;
∀ v ∈ V   v.marca = inesplorato;
BFS(G, y);
if(z.marca != inesplorato) return 2;
else return 3;
```

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

**Esercizio 5.**
Per il grafo riportato qui sotto si calcoli un albero di copertura minimo mediante l'algoritmo di Kruskal. Per ogni iterazione si evidenzi l'arco che viene eventualmente inserito nell'albero.




**Soluzione.**

```
(g, h), peso 1:  MAR
(c, i), peso 2:  MAR
(f, g), peso 2:  MAR
(a, b), peso 4:  MAR
(c, f), peso 4:  MAR
(g, i), peso 6:  eliminato
(c, d), peso 7:  MAR
```

```
(h, i), peso 7:  eliminato
(a, h), peso 8:  MAR
(b, c), peso 8:  eliminato
(d, e), peso 9:  MAR
(e, f), peso 10:  eliminato
(b, h), peso 11:  eliminato
(d, f), peso 14:  eliminato
```

MAR: peso 37