008AA – ALGORITMICA E LABORATORIO

Seconda prova di verifica, 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.

Supponete di avere una scacchiera con $n \times n$ caselle e una pedina. Quando posizionata sulla generica casella (i, j) per la pedina sono possibili al più due mosse:

- spostarsi verso il basso nella casella $(i+1, j)$, se $i < n$;
- spostarsi verso destra nella casella $(i, j+1)$, se $j < n$.

Progettare un algoritmo di programmazione dinamica che calcola il numero di percorsi possibili per spostare la pedina dalla casella $(1, 1)$ in alto a sinistra alla casella (n, n) in basso a destra (ad esempio, in una scacchiera 3×3 i percorsi possibili sono 6). Valutare la complessità dell'algoritmo proposto.

Soluzione.

```
CONTA_PERCORSI(n)

   L = nuova matrice n x n
   //soluzione problemi elementari e memorizzazione nella tabella
   for (i = 1 ; i <= n; i++)   L[i, 1] = 1;
   for (j = 1 ; j <= n; j++)   L[1, j] = 1;
   //riempimento della tabella
   for (i = 2; i <= n; i++) {
      for (j = 2; j <= n; j++) {
          L[i,j] = L[i-1,j] + L[i, j-1];
      }
   }
   return L[n,n];
```

Complessità: $T(n) = O(n^2)$.

Esercizio 3.
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.

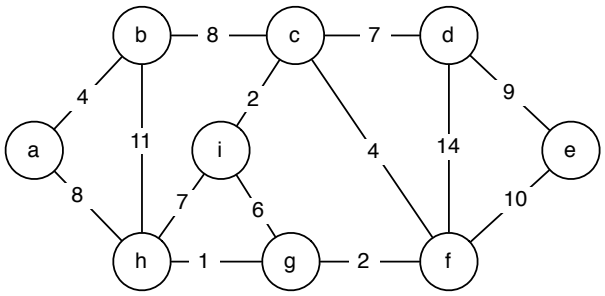
Soluzione.

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 4.
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
```