008AA Algoritmica e Laboratorio
Verifica del 24 maggio 2010 – SOLUZIONI

Esercizio 1

Supponete di avere una scacchiera con *n x n* caselle e una pedina che dovete muovere dall'estremità inferiore a quella superiore. Una pedina si può muovere una cella in alto, oppure una cella in diagonale destra, oppure una cella in diagonale sinistra. Le celle sono denotate da una coppia di coordinate *(x,y)*. Quando una cella *(x,y)* viene visitata, guadagnate $p(x,y) \geq 0$.

Calcolare un percorso da una qualunque casella dell'estremità inferiore ad una qualunque casella dell'estremità superiore, massimizzando il profitto. Definire anche una procedura che stampi il percorso calcolato.

RicercaCammino(p,n)

```
//soluzione problemi elementari e memorizzazione nella tabella g
for (j = 1 ; j <= n; j++)   g[n, j] = p(n, j);
//riempimento della tabella (dal basso verso l'alto)
for (i = n – 1; i >= 1; i--) {
     for (j = 1; j <= n; j++) {
          // ricerca del massimo tra g[i+1,j-1], g[i+1,j], g[i+1,j+1]
          g[i,j] = g[i+1,j];
          m[i,j] = 0; // matrice per la ricostruzione del percorso
          if (j > 1 && g[i+i,j-1] > g[i,j] ) {
               g[i,j] = g[i+1,j-1];
               m[i,j] = -1;
          }
          if (j < n && g[i+i,j+1] > g[i,j] ) {
               g[i,j] = g[i+1,j+1];
               m[i,j] = 1;
          }
         g[i,j] = g[i,j] + p(i,j); //profitto della cella (i,j)
     }
}
// Cerca la casella iniziale con massimo guadagno
max = g[1,1];
start = 1;
for (j = 2; j <= n; j++) {
     if (g[1,j] > max) {
            max = g[1, j]
            start = j;
}
// Stampa del percorso
j = start
for (i = 1; i <= n-1; i++) {
      PRINT (i,j) -> (i+1, j+m[i,j])
       j = j + m[i,j]
}
```

Esercizio 2

Sia dato un array *A[0,n-1]* di stringhe (visto come **`char **A`**) il cui valore può essere soltanto: **`{vero, forse, falso}.`**

Scrivere una funzione **`C`** che riceve in input l'array *A* e la sua lunghezza *n*, e in tempo *O(n)* riarrangia le stringhe di A in modo che tutte le stringhe **`falso`** precedano le stringhe **`forse`**, e queste ultime precedano le stringhe **`vero`**.

```
void Sort3(char **A, int n)
{
   int num_falso, num_forse;
   int pos_falso, pos_forse, pos_vero;
   int i;

   char **B = (char **) malloc(n * sizeof(char *));
   num_falso = num_forse = 0;
   for(i=0; i < n; i++){
       if(strcmp(A[i],"falso") == 0) num_falso++;
       if(strcmp(A[i],"forse") == 0) num_forse++;
       }

   pos_falso = 0;  //posizione partenza stringhe "falso"
   pos_forse = num_falso;  //posizione partenza stringhe "forse"
   pos_vero = num_falso + num_forse; //posizione partenza stringhe "vero"
   for(i=0; i < n; i++){
       if(strcmp(A[i],"falso") == 0){
           B[pos_falso] = A[i];
           pos_falso++;
       } else if(strcmp(A[i],"forse") == 0){
           B[pos_forse] = A[i];
           pos_forse++;
       } else {
           B[pos_vero] = A[i];
           pos_vero++;
       }
   }
   for(i=0; i < n; i++) A[i] = B[i];
   free(B);
}
```

Esercizio 3

Progettare un algoritmo efficiente che stabilisca se un albero binario è un albero di Fibonacci, e analizzarne la complessità.
Suggerimento: *un albero binario è un albero di Fibonacci se e solo se tutti i nodi interni hanno fattore di bilanciamento uguale a 1.*

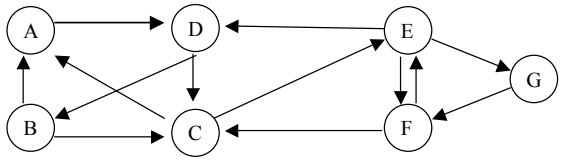
L'algoritmo restituisce la coppia (un booleano e un intero): *<è un albero di Fibonacci, altezza>*

```
Fibonacci(u)
  if (u == null) return <true; -1>;
  if (u.sx == null && u.dx == null) return <true; 0>;
  <isFibSx, hSx> = Fibonacci(u.sx);
  <isFibDx, hDx> = Fibonacci(u.dx);
  isFib = isFibSx && isFibDx && (|hSx - hDx| == 1);
  h = max(hSx, hDx) + 1;
  return <isFib, h>;
```

L'algoritmo esegue una visita dell'albero, dunque la sua complessità in tempo è $T(n) = \Theta(n)$.

Esercizio 4

È dato il grafo seguente, rappresentato con liste di adiacenza ordinate alfabeticamente.

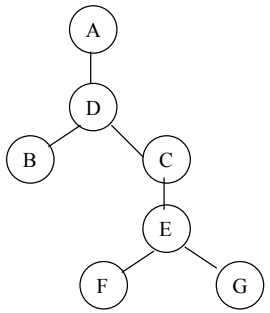


1) Indicare l'ordine di visita BFS e DFS dei vertici del grafo, partendo dal vertice A.
2) Disegnare gli alberi BFS e DFS ottenuti con le visite.
3) Indicare la classificazione degli archi indotta dalla visita DFS.

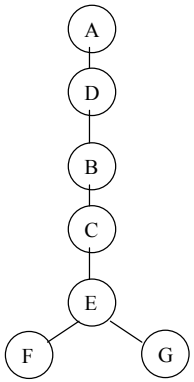
SOLUZIONE

1) visita BFS: A, D, B, C, E, F, G
   visita DFS: A, D, B, C, E, F, G

2) albero BFS



albero DFS



3) Classificazione degli archi (visita DFS)

Archi dell'albero: (A,D), (D,B) (B,C), (C,E), (E,F), (E,G)
Archi all'indietro: (B,A), (C,A), (E,D), (F,C), (F,E)
Archi in avanti: (D,C)
Archi trasversali a sinistra: (G,F)