Algortimica e Laboratorio A/B
Esercitazione del 26/4/2013

Esercizio: DecreaseKey.
Dato un heap H contenente priorità, definire un algoritmo che implementi la funzione DecreaseKey(c, d) che decrementa la priorità dell'elemento in posizione c dell'heap di una quantità d.

Soluzione:
Dopo il decremento l'invariante dello heap può essere violato dal nodo in posizione c verso i figli. Una soluzione è quella di chiamare RiorganizzaHeap(c) oppure solo la parte che confronta soltanto il nodo decrementato con i suoi discendenti.

```
DecreaseKey (c, d):
H[c] = H[c]- d;
WHILE (sin(c)< heapsize && c!= MAX(c)){
   migliore = MAX(c);
   scambia (c, migliore);
   c = migliore;
}
```

la funzione MAX(i) che seleziona il massimo tra il nodo di posizione i e i suoi due figli se esistono è una versione più snella di **Migliorepadrifigli** del libro a pg 51.

```
MAX(i):
j = k = sin(i);
IF (k+1 < heapsize) k=k+1;
IF (H[k] > H[j]) j=k;
IF (H[i] > H[j]) j=i;
RETURN j;
```

Esercizio: Zaino con la Programmazione Dinamica
Si consideri un problema dello Zaino in cui ci sono 3 elementi: articolo1, articolo2 e articolo3, il cui valore è rispettivamente di 60, 100 e 120 euro e il cui peso è di 1, 2 e 3 kg e lo zaino può sopportarne al massimo 5. Mostrare:

1) La tabella di programmazione dinamica che conduce alla soluzione ottima.
2) La soluzione trovata dall'algoritmo.

Soluzione:

	0	1	2	3	4	5
0	**0**	**0**	**0**	**0**	**0**	**0**
art1	**0**	**60**	**60**	**60**	**60**	**60**
art2	**0**	**60**	**100**	**160**	**160**	**160**
art3	**0**	**60**	**100**	**160**	**180**	**220**

La soluzione trovata dall'algoritmo corrispondente alla casella M[3] [5] è costituita dagli elementi {art2, art3} e si ricava sull'array partendo da M[3] [5] , passando da M[2] [2] e terminando a M[1] [0].

Esercizio: Conta percorsi su scacchiera

Supponete di avere una scacchiera con n x n caselle e una pedina.

Posizionata sulla generica casella (i,j), per la pedina sono possibili al più due mosse:

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

2) spostarsi verso destra nella casella (i, j+1), se j < n-1.

Progettare un algoritmo di programmazione dinamica che calcola il numero di percorsi possibili per spostare la pedina dalla casella (0,0) in alto a sinistra alla casella (n-1,n-1) in basso a destra (ad esempio, in una scacchiera 3 x 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 = 0 ; i <= n; i++)   L[i, 0] = 1;
for (j = 0 ; j <= n; j++)   L[0, j] = 1;
//riempimento della tabella
for (i = 1; i <= n; i++) {
  for (j = 1; j <= n; j++) {
     L[i,j] = L[i-1,j] + L[i, j-1];
   }
  }
return L[n-1,n-1]
```

Complessità $T(n) = O(n^2)$.

Esercizio: Ricerca cammino su scacchiera.

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 = 0 ; j < n; j++)   g[n, j] = p(n, j);
//riempimento della tabella (dal basso verso l'alto)
for (i = n – 2; i >= 0; i--) {
     for (j = 0; 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 > 0 && g[i+i,j-1] > g[i,j] ) {
               g[i,j] = g[i+1,j-1];
               m[i,j] = -1;
          }
          if (j < n-1 && 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[0,0];
start = 0;
for (j = 1; j < n; j++) {
     if (g[0,j] > max) {
             max = g[0, j]
             start = j;
}
// Stampa del percorso
j = start
for (i = 0; i < n-1; i++) {
      PRINT (i,j) -> (i+1, j+m[i,j])
       j = j + m[i,j]
}
```