



Esercitazione su temi d'esame

30 Maggio 2018

1. Memoization (15 Luglio 2014)

La procedura `llcs3` determina la *lunghezza della sottosequenza comune più lunga* (LLCS) di tre stringhe:

```
(define llcs3
  (lambda (t u v)
    (cond ((or (string=? t "") (string=? u "") (string=? v ""))
           0)
          ((char=? (string-ref t 0) (string-ref u 0) (string-ref v 0))
           (+ 1 (llcs3 (substring t 1) (substring u 1) (substring v 1))))
          (else
           (max (llcs3 (substring t 1) u v)
                 (llcs3 t (substring u 1) v)
                 (llcs3 t u (substring v 1)))))))
```

Trasforma la procedura `llcs3` in un programma *Java* che applica opportunamente la tecnica *top-down* di *memoization*.

2. Programmazione in Java (15 Luglio 2014)

Come è noto dall'algebra lineare, una matrice quadrata Q si dice *simmetrica* se coincide con la propria trasposta, ovvero se $Q_{ij} = Q_{ji}$ per tutte le coppie di indici della matrice. Scrivi un metodo statico in Java per verificare se l'argomento è una matrice simmetrica (assumendo che tale argomento rappresenti una matrice *quadrata*).

Per quanto possibile, struttura il codice in modo tale da ridurre al minimo il numero di confronti effettuati dal programma nei casi in cui la matrice è effettivamente simmetrica.

3. Oggetti in Java (9 Settembre 2015)

Il modello della scacchiera realizzato dalla classe `Board` per affrontare il rompicapo delle n regine deve essere integrato introducendo due nuovi metodi: `isFreeRow(int)`, che dato un indice di riga i compreso fra 1 e la dimensione n della scacchiera, restituirà *true* se e solo se nella riga i non c'è alcuna regina; `addQueen(String)`, che svolge la stessa funzione di `addQueen(int, int)` ma ricevendo come argomento la codifica della posizione tramite una stringa di due caratteri (overloading), una lettera per la colonna e una cifra per la riga secondo le convenzioni consuete. Inoltre, `addQueen` e `removeQueen` non devono modificare lo stato della scacchiera se l'operazione è inconsistente perché due regine verrebbero a trovarsi sulla stessa casella oppure perché nella posizione data non c'è una regina da rimuovere. Per esempio, il metodo statico `listOfCompletions`, definito sotto a destra, stamperà tutte le soluzioni del rompicapo, se ve ne sono, compatibili con una disposizione iniziale di regine che non si minacciano reciprocamente.

In base a quanto specificato sopra, modifica opportunamente la classe `Board`.

```
Board( int n ) //costruttore

void addQueen( int i, int j )
void removeQueen( int i, int j )

int size()
int queensOn()
boolean underAttack( int i, int j )
String arrangement()

boolean isFreeRow( int i )
void addQueen( String pos )

//esempio: b.addQueen( "b6" )

public static void listOfCompletions( Board b ) {
    int n = b.size(); int q = b.queensOn();
    if ( q == n ) {
        System.out.println( b.arrangement() );
    } else {
        int i = 1;
        while ( !b.isFreeRow(i) ) {
            i = i + 1; // ricerca di una riga libera
        }
        for ( int j=1; j<=n; j=j+1 ) {
            if ( ! b.underAttack(i,j) ) {
                b.addQueen( i, j );
                listOfCompletions( b );
                b.removeQueen( i, j );
            }
        }
    }
}
```