

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Procedure in Scheme

Considera il seguente programma in Scheme:

```
(define s
  (lambda (t)
    (let ((x (car t)) (y (cadr t)) (z (caddr t)))
      (list (+ x 1) (+ y z) (+ z 2))
    )))

(define h
  (lambda (n)
    (f n (list 0 0 1) s)
  ))

(define f
  (lambda (n t g)
    (if (= (car t) n)
        (cadr t)
        (f n (g t) g)
    )))
```

Completa le seguenti valutazioni, dove x è un numero naturale, riportando il risultato nell'apposito spazio:

(h 1)	→	_____
(h 3)	→	_____
(h 12)	→	_____
(h x)	→	_____ (*)

2. Programmazione in Scheme

Definisci una procedura *shared* in Scheme che, date due liste u , v (strettamente) ordinate di numeri interi positivi, restituisca la lista ordinata degli elementi comuni a u e v . Per esempio:

(shared '(1 3 5 6 7 8 9 10) '(0 1 2 3 4 5 7 9)) → (1 3 5 7 9)

3. Verifica formale della correttezza

In relazione alla soluzione dell'esercizio 1, dimostra che il risultato calcolato dalla procedura h in funzione della variabile intera $x \geq 0$ corrisponde a quanto riportato nella riga (*). In particolare, si può dimostrare per induzione che

$$\forall k, i \in \mathbb{N} . (f(k+i) = (k^2 + 2k + 1) + s) \rightarrow (k+i)^2$$

Formalizza la dimostrazione, seguendo i passi indicati qui sotto:

Formalizza il caso base / i casi base:

Formalizza l'ipotesi induttiva:

Formalizza la proprietà da dimostrare come passo induttivo:

Sviluppa la dimostrazione per induzione:

Applica la proprietà dimostrata per induzione per verificare che la caratterizzazione (*) è corretta:

4. Programmazione Dinamica

Il seguente programma ricorsivo in *Java* calcola una concatenazione di minima lunghezza delle differenze fra due stringhe. Trasforma il programma per realizzarne una versione iterativa applicando opportunamente la tecnica di *programmazione dinamica*.

```
public static int diff( String u, String v ) {  
    if ( (u.length() == 0) || (v.length() == 0) ) {  
        return ( u + v ).length();  
    } else if ( u.charAt(0) == v.charAt(0) ) {  
        return diff( u.substring(1), v.substring(1) );  
    } else {  
        return Math.min( diff(u.substring(1),v), diff(u,v.substring(1)) ) + 1;  
    }  
}
```

5. Programmazione orientata agli oggetti in Java

Per consentire una maggiore libertà nella disposizione delle regine sulla scacchiera e nell'ordine secondo cui vengono introdotte o rimosse, si apportano alcune modifiche al protocollo e alla rappresentazione interna della classe Board. In particolare, i metodi `underAttack(int)`, `addNextQueen(int)` e `removeLastQueen()` vengono sostituiti dai metodi `underAttack(int,int)`, `addQueen(int,int)`, `removeQueen(int,int)`, rispettivamente, dove la coppia di parametri interi rappresenta gli indici di riga e di colonna di una casella. Risulta così possibile disporre una regina in qualunque casella della scacchiera, purché non sia già occupata, anche se questa è "sotto scacco" da parte di un'altra regina. È inoltre possibile rimuovere qualunque regina che sia stata disposta sulla scacchiera.

```
public class Board {

    private boolean[][] config;
    private int[] rowUnderAttack;
    private int[] columnUnderAttack;
    private int[] diagDwUnderAttack;
    private int[] diagUpUnderAttack;
    private int queens;
    private int n;

    public Board( int size ) {
        config = new boolean[size][size];
        for ( int i=0; i<size; i=i+1 ) {
            for ( int j=0; j<size; j=j+1 ) {
                config[i][j] = false;
            }
        }
        rowUnderAttack = new int[size];
        for ( int i=0; i<size; i=i+1 ) {
            rowUnderAttack[i] = 0;
        }
        columnUnderAttack = new int[size];
        for ( int i=0; i<size; i=i+1 ) {
            columnUnderAttack[i] = 0;
        }
        diagDwUnderAttack = new int[2*size-1];
        for ( int i=0; i<2*size-1; i=i+1 ) {
            diagDwUnderAttack[i] = 0;
        }
        diagUpUnderAttack = new int[2*size-1];
        for ( int i=0; i<2*size-1; i=i+1 ) {
            diagUpUnderAttack[i] = 0;
        }
        queens = 0;
        n = size;
    }

    public int boardSize() {
        return n;
    }

    public int queensOnBoard() {
        return queens;
    }

    public boolean underAttack(int row,int col) {

        return (
            (rowUnderAttack[row-1] > 0)
            || (columnUnderAttack[col-1] > 0)
            || (diagDwUnderAttack[row-col+n-1] > 0)
            || (diagUpUnderAttack[row+col-2] > 0)
        );
    }

    public void addQueen( int row, int col ) {

        if ( config[row-1][col-1] ) { return; }
        config[row-1][col-1] = true;
        rowUnderAttack[row-1]
            = rowUnderAttack[row-1] + 1;
        columnUnderAttack[col-1]
            = columnUnderAttack[col-1] + 1;
        diagDwUnderAttack[row-col+n-1]
            = diagDwUnderAttack[row-col+n-1] + 1;
        diagUpUnderAttack[row+col-2]
            = diagUpUnderAttack[row+col-2] + 1;
        queens = queens + 1;
    }

    public void removeQueen( int row, int col ) {

        if ( !config[row-1][col-1] ) { return; }
        config[row-1][col-1] = false;
        rowUnderAttack[row-1]
            = rowUnderAttack[row-1] - 1;
        columnUnderAttack[col-1]
            = columnUnderAttack[col-1] - 1;
        diagDwUnderAttack[row-col+n-1]
            = diagDwUnderAttack[row-col+n-1] - 1;
        diagUpUnderAttack[row+col-2]
            = diagUpUnderAttack[row+col-2] - 1;
        queens = queens - 1;
    }

    public boolean[][] arrangement() {
        return config;
    }

} // class Board
```

Qui di seguito è riportato il codice *Java* del programma sviluppato a lezione per calcolare il numero di soluzioni del problema delle regine per una scacchiera $n \times n$. Apporta le modifiche necessarie per renderlo compatibile con il nuovo protocollo della classe Board specificato sopra.

```
public static int queensArrangements( int n ) {
    return queensCompletions( new Board(n) );
}

public static int queensCompletions( Board board ) {
    if ( board.queensOnBoard() == board.boardSize() ) {
        return 1;
    } else {
        return completionsFrom( 1, board );
    }
}

public static int completionsFrom( int c, Board board ) {
    if ( c > board.boardSize() ) {
        return 0;
    } else if ( board.underAttack(c) ) {
        return completionsFrom( c+1, board );
    } else {
        board.addNextQueen( c );
        int x = queensCompletions( board );
        board.removeLastQueen();
        return x + completionsFrom( c+1, board );
    }
}
```