Corso di Programmazione

III Accertamento del 11 Luglio 2008 / A

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmi in Java

Traduci la seguente procedura Scheme in un corrispondente metodo statico formalizzato nel linguaggio *Java*:

```
(define diff
  (lambda (u v)  ; u, v stringhe
    (cond ((= (string-length u) 0) (string-length v))
          ((= (string-length v) 0) (string-length u))
          ((char=? (string-ref u 0) (string-ref v 0))
           (diff (substring u 1) (substring v 1)))
          (else
           (+ 1 (min (diff (substring u 1) v) (diff u (substring v 1)))))
          )))
```

Nota: il protocollo pubblico della classe `String` in Java comprende i metodi `length()`, `charAt(int)`, `substring(int)`, `substring(int,int)`. Inoltre, la classe `Math` mette a disposizione il metodo statico `min(int,int)`.

1. Programmi in Java

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

2. Memoization

Trasforma il programma in *Java* che risolve l'esercizio 1 applicando opportunamente la tecnica di *memoization.*

2. Memoization

```
public static int diff( String u, String v ) {

  int i = u.length();
  int j = v.length();

  int[][] history = new int[i+1][j+1];

  for ( int x=0; x<=i; x=x+1 ) {
    for ( int y=0; y<=j; y=y+1 ) {
      history[x][y] = UNDEFINED;
  }}
  return diffRec( u, v, history );
}


public static int diffRec( String u, String v, int[][] history ) {

  int i = u.length();
  int j = v.length();

  if ( history[i][j] == UNDEFINED ) {

    if ( u.length() == 0 ) {
      history[i][j] = v.length();
    } else if ( v.length() == 0 ) {
      history[i][j] = u.length();
    } else if ( u.charAt(0) == v.charAt(0) ) {
      history[i][j] = diffRec( u.substring(1), v.substring(1), history );
    } else {
      history[i][j] = 1 + Math.min( diffRec(u.substring(1),v,history),
                                    diffRec(u,v.substring(1),history) );
  }}
  return history[i][j];
}


public static final int UNDEFINED = -1;
```

3. Classi in Java

Qui di seguito è riportato il codiece Java della classe `Board` per rappresentare le disposizioni delle regine in una scacchiera *NxN*, nella forma discussa a lezione.

```
public class Board {

  private int[]      config;
  private boolean[]  columnUnderAttack;
  private boolean[]  diagDwUnderAttack;
  private boolean[]  diagUpUnderAttack;
  private int        rows;
  private int        n;


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

  public int boardSize() {
    return n;
  }

  public int queensOnBoard() {
    return rows;
  }

  public boolean underAttack( int col ) {
    int row = rows + 1;
    int size = n;
    return ( columnUnderAttack[col-1]
          || diagDwUnderAttack[row-col+size-1]
          || diagUpUnderAttack[row+col-2]
           );
  }

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

  public void addNextQueen( int col ) {
    int row = rows + 1;
    int size = n;
    config[row-1] = col;
    columnUnderAttack[col-1] = true;
    diagDwUnderAttack[row-col+size-1] = true;
    diagUpUnderAttack[row+col-2] = true;
    rows = row;
  }

  public void removeLastQueen() {
    int row = rows;
    int size = n;
    int col = config[row-1];
    config[row-1] = 0;
    columnUnderAttack[col-1] = false;
    diagDwUnderAttack[row-col+size-1] = false;
    diagUpUnderAttack[row+col-2] = false;
    rows = row - 1;
  }

}  // class Board
```

Allo scopo di 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.

Per quanto riguarda il protocollo, 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.

Per quanto riguarda la rappresentazione interna, la disposizione delle regine viene rappresentata da una matrice *NxN* di booleani, dove un elemento di valore `true` nella matrice indica la presenza di una regina nella casella codificata dai corrispondenti indici. Inoltre, nella nuova realizzazione le variabili di istanza `rowUnderAttack`, `columnUnderAttack`, `diagDwUnderAttack` e `diagUpUnderAttack` registrano il numero di regine collocate su una stessa riga, su una stessa colonna o su una stessa diagonale, mentre `queens` tiene conto del numero complessivo di regine sulla scacchiera.

Completa la nuova realizzazione della classe `Board` impostata nel riquadro della pagina seguente. In particolare, definisci il corpo dei metodi `underAttack(int,int)`, `addQueen(int,int)` e `removeQueen(int,int)` tenendo conto delle indicazioni fornite sopra.

```
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[][] arrangement() {
    return config;
  }
```

```
  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;
  }

}  // class Board
```

Corso di Programmazione

III Accertamento del 11 Luglio 2008 / B

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmi in Java

Traduci la seguente procedura Scheme in un corrispondente metodo statico formalizzato nel linguaggio *Java*:

```
(define diff
  (lambda (r s)  ; r, s stringhe
    (cond ((or (= (string-length r) 0) (= (string-length s) 0))
           (string-length (string-append r s)))
          ((char=? (string-ref r 0) (string-ref s 0))
           (diff (substring r 1) (substring s 1)))
          (else
           (min (+ (diff (substring r 1) s) 1)
                (+ (diff r (substring s 1)) 1)))
          )))
```

Nota: il protocollo pubblico della classe `String` in Java comprende i metodi `length()`, `charAt(int)`, `substring(int)`, `substring(int,int)`. Inoltre, la classe `Math` mette a disposizione il metodo statico `min(int,int)`.

1. Programmi in Java

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

2. Memoization

Trasforma il programma in *Java* che risolve l'esercizio 1 applicando opportunamente la tecnica di *memoization.*

2. Memoization

```
public static int diff( String r, String s ) {

  int i = r.length();
  int j = s.length();

  int[][] history = new int[i+1][j+1];

  for ( int x=0; x<=i; x=x+1 ) {
    for ( int y=0; y<=j; y=y+1 ) {
      history[x][y] = UNDEFINED;
  }}
  return diffRec( r, s, history );
}


public static int diffRec( String r, String s, int[][] history ) {

  int i = r.length();
  int j = s.length();

  if ( history[i][j] == UNDEFINED ) {

    if ( (r.length() == 0) || (s.length() == 0) ) {
      history[i][j] = ( r + s ).length();
    } else if ( r.charAt(0) == s.charAt(0) ) {
      history[i][j] = diffRec( r.substring(1), s.substring(1), history );
    } else {
      history[i][j] = Math.min( diffRec(r.substring(1),s,history) + 1,
                                diffRec(r,s.substring(1),history) + 1 );
  }}
  return history[i][j];
}


public static final int UNDEFINED = -1;
```

3. Classi in Java

Qui di seguito è riportato il codiece Java della classe `Board` per rappresentare le disposizioni delle regine in una scacchiera *NxN*, nella forma discussa a lezione.

```
public class Board {

  private int[]      config;
  private boolean[]  columnUnderAttack;
  private boolean[]  diagDwUnderAttack;
  private boolean[]  diagUpUnderAttack;
  private int        rows;
  private int        n;


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

  public int boardSize() {
    return n;
  }

  public int queensOnBoard() {
    return rows;
  }

  public boolean underAttack( int col ) {
    int row = rows + 1;
    int size = n;
    return ( columnUnderAttack[col-1]
          || diagDwUnderAttack[row-col+size-1]
          || diagUpUnderAttack[row+col-2]
           );
  }

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

  public void addNextQueen( int col ) {
    int row = rows + 1;
    int size = n;
    config[row-1] = col;
    columnUnderAttack[col-1] = true;
    diagDwUnderAttack[row-col+size-1] = true;
    diagUpUnderAttack[row+col-2] = true;
    rows = row;
  }

  public void removeLastQueen() {
    int row = rows;
    int size = n;
    int col = config[row-1];
    config[row-1] = 0;
    columnUnderAttack[col-1] = false;
    diagDwUnderAttack[row-col+size-1] = false;
    diagUpUnderAttack[row+col-2] = false;
    rows = row - 1;
  }

}  // class Board
```

Allo scopo di 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.

Per quanto riguarda il protocollo, 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.

Per quanto riguarda la rappresentazione interna, la disposizione delle regine viene rappresentata da una matrice *NxN* di booleani, dove un elemento di valore `true` nella matrice indica la presenza di una regina nella casella codificata dai corrispondenti indici. Inoltre, nella nuova realizzazione le variabili di istanza `rowUnderAttack`, `columnUnderAttack`, `diagDwUnderAttack` e `diagUpUnderAttack` registrano il numero di regine collocate su una stessa riga, su una stessa colonna o su una stessa diagonale, mentre `queens` tiene conto del numero complessivo di regine sulla scacchiera.

Completa la nuova realizzazione della classe `Board` impostata nel riquadro della pagina seguente. In particolare, definisci il corpo dei metodi `underAttack(int,int)`, `addQueen(int,int)` e `removeQueen(int,int)` tenendo conto delle indicazioni fornite sopra.

```
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[][] arrangement() {
    return config;
  }
```

```
  public boolean underAttack(int row,int col) {
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
  }

  public void addQueen( int row, int col ) {
    if ( config[row-1][col-1] ) { return; }
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
  }

  public void removeQueen( int row, int col ) {
    if ( !config[row-1][col-1] ) { return; }
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
  }
} // class Board
```

vedi esercizio 3/A