

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Ricorsione e argomenti procedurali

Dati due interi non negativi i, j , la procedura `paths` restituisce la lista di tutti i percorsi di Manhattan fra due incroci che nella mappa distano i isolati verso il basso e j isolati verso destra. I percorsi sono rappresentati da una stringa composta da cifre e dalle lettere `d` (down) e `r` (right), dove le cifre che precedono la lettera `d` (`r`) indicano il numero di spostamenti consecutivi verso il basso (rispettivamente a destra). Per esempio:

`(paths 2 2)` \rightarrow `("2d2r" "1d1r1d1r" "1d2r1d" "1r2d1r" "1r1d1r1d" "2r2d")`

dove la terza stringa della lista codifica il percorso costituito da uno spostamento in basso, due a destra e un ultimo spostamento in basso. Completa il programma riportato nel riquadro per definire la procedura `paths`.

```
(define paths
  (lambda (i j)
    ; i, j ≥ 0
    (cond ((and (= i 0) (= j 0)) (list "0"))
          ((= i 0) (list (string-append (number->string j) "r")))
          ((= j 0) (list (string-append (number->string i) "d")))
          (else (append (d-paths 1 (- i 1) j) (r-paths 1 i (- j 1))))
    )))

(define d-paths
  (lambda (s i j)
    ; i, j ≥ 0; s > 0
    (cond ((= i 0)
          (list (string-append (number->string s) "d" (number->string j) "r")))
          (else
           (append
            (d-paths (+ s 1) (- i 1) j)

            (map _____
                  (r-paths 1 i (- j 1)))
            )
          )
    )))

(define r-paths
  (lambda (s i j)
    ; i, j ≥ 0; s > 0

    (cond ( _____
          (list (string-append (number->string s) "r" (number->string i) "d")))
          (else
           (append

            (map _____
                  _____ )

            (r-paths _____ )
            )
          )
    )))
```

2. Memoization

Si vuole applicare la tecnica *top-down* di *memoization* al programma dell'esercizio 1. Assumi di avere a disposizione la classe `StringList`, già utilizzata in laboratorio, che rende disponibili strumenti analoghi alle primitive Scheme per operare con liste di stringhe, specificamente una costante `NULL` per la lista vuota e i metodi statici `listNull`, `listCar`, `listCdr` e `listCons`, corrispondenti a `null?`, `car`, `cdr` e `cons`. Assumi inoltre che siano già state realizzate le procedure `append` e `prefix`: la prima restituisce l'*append* delle due liste passate come argomento; la seconda restituisce una lista in cui a tutte le stringhe della lista passata come secondo argomento viene premesso il prefisso passato come primo argomento. Nel riquadro è impostata la soluzione: completa il metodo statico `paths` e definisci il metodo `dPaths` che realizza la funzione di `d-paths` tramite memoization. (`rPaths` è analogo e non è richiesto.)

```
public static StringList paths( int i, int j ) { // i, j ≥ 0
    if ( ( i == 0 ) && ( j == 0 ) ) {
        return StringList.listCons( "0", StringList.NULL );
    } else if ( i == 0 ) {
        return StringList.listCons( j+"r", StringList.NULL );
    } else if ( j == 0 ) {
        return StringList.listCons( i+"d", StringList.NULL );
    } else {
        StringList[][][] dh = new StringList[ i+1 ][ i+1 ][ j+1 ];
        for ( int s=0; s<=i; s=s+1 ) {
            for ( int u=0; u<=i; u=u+1 ) {
                for ( int v=0; v<=j; v=v+1 ) {
                    dh[s][u][v] = null;
                }
            }
        }
        StringList[][][] rh = new StringList[ j+1 ][ i+1 ][ j+1 ];
        for ( int s=0; s<=j; s=s+1 ) {
            for ( int u=0; u<=i; u=u+1 ) {
                for ( int v=0; v<=j; v=v+1 ) {
                    rh[s][u][v] = null;
                }
            }
        }
        return append( dPaths( 1, i-1, j, ..... ),
                       rPaths( 1, i, j-1, ..... ) );
    }
}

private static StringList append( StringList sl1, StringList sl2 ) { ... }
private static StringList prefix( String p, StringList sl ) { ... }
```

3. Ricorsione e iterazione

Definisci in Java un metodo statico `boolean[] charSet(Node root)` che data la radice `root` di un albero di *Huffman* restituisce un array di booleani per rappresentare l'insieme dei caratteri associati alle foglie dell'albero; specificamente, la componente di indice k dell'array deve riportare il valore *true* se e soltanto se il carattere di codice ASCII k è effettivamente presente nell'albero. Come ulteriore vincolo, il programma che definisce il metodo `charSet` non può far uso della ricorsione, ma deve basarsi su costrutti iterativi e su opportune strutture di supporto.

4. Verifica formale della correttezza

Dato un intero $n \geq 0$, il seguente metodo statico calcola la quarta potenza di n utilizzando solo somme e confronti. Nel programma sono riportate preconditione, postcondizione e invariante. Introduci opportune espressioni negli spazi a tratto punteggiato in modo tale che i valori assunti dalle variabili soddisfino le relazioni specificate dalle asserzioni.

```
public static int q( int n ) {           // Pre:  $n \geq 0$ 

    int  u = 0,                          v = _____ ;

    int  x = _____ , y = _____ ;

    while ( ( x < v ) || ( u < n ) ) {     // Inv:  $0 \leq u \leq n, 0 \leq x \leq v = u^2, y = x^2$ 
        if ( x < v ) {
            y = y + x;  x = x + 1;  y = y + _____ ;
        } else {
            v = v + u;  u = u + 1;  v = v + _____ ;
        }
    }

    return _____ ;                 // Post: valore restituito =  $n^4$ 
}
```

5. Programmazione orientata agli oggetti in Java

Definisci in Java un metodo statico `double averageCodeLength(Node root)` che data la radice `root` di un albero di *Huffman* restituisce la lunghezza media (numero medio di bit) delle codifiche dei caratteri che compaiono nel documento a partire dal quale l'albero è stato costruito. Più precisamente, si tratta di moltiplicare la lunghezza del codice di ciascun carattere, corrispondente al livello a cui si trova la foglia rispetto alla radice dell'albero, per il peso di quel carattere, cioè il numero di occorrenze registrato in corrispondenza al nodo; quindi si sommano tutti i prodotti e si divide il risultato per il numero complessivo di caratteri del documento. Chiaramente si suppone di utilizzare l'albero di Huffman costruito in fase di compressione, albero in cui sono rappresentati anche i pesi dei nodi.