

Domanda 1

Risposta non
ancora data

Non valutata

1. Argomenti procedurali in Scheme

Dati tre interi non negativi i, j, n , con $0 \leq i+j \leq n$, la procedura `seqs` restituisce la lista di tutte le possibili stringhe di lunghezza n composte dai caratteri 'a', 'b', 'c' e contenenti esattamente i caratteri 'a', j caratteri 'b' e i rimanenti 'c'.

Esempi:

(seqs 0 0 3) → ("ccc")

(seqs 1 1 3) → ("abc" "acb" "bac" "bca" "cab" "cba")

1.1. Completa la definizione della procedura `seqs`.

```
(define seqs
  (lambda (i j n) ; i, j, n: interi non negativi t.c. 0 ≤ i+j ≤ n
    (let (
      (x (if (= i 0) null (seqs (- i 1) j (- n 1))))
      (y (if (= j 0) null (seqs i (- j 1) (- n 1))))
      (z (if  null  ))
    )
      (if (= n 0)
        (list "")
        (append
          (map  x)
          (map  y)
          (map  z)
        )
      )
    ))
```

Ricomincia

Salva

Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? ▶[Esporta la domanda nel formato Moodle XML](#)[Minimizza tutto](#)

Opzioni per il tentativo

Comportamento della domanda ?

Feedback differito

Punteggio massimo

Ricomincia con queste opzioni

Domanda **1**Risposta non
ancora data

Non valutata

1. Argomenti procedurali in Scheme**1.2.** Considera la valutazione della procedura *segs* nel secondo esempio riportato sopra:Quali sono i valori associati alle variabili *x*, *y* introdotte nel costrutto *let* relativamente all'invocazione iniziale di *segs*?

- *x* = ;
- *y* = .

Ricomincia

Salva

Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? - ►[Esporta la domanda nel formato Moodle XML](#)[Minimizza tutto](#)**Opzioni per il tentativo**

Comportamento della domanda ?

Feedback differito

Punteggio massimo

Ricomincia con queste opzioni

Opzioni di visualizzazione

Se corretto

Visualizzato

Punteggio

Visualizza punteggio e max.

Cifre decimali nelle valutazioni

2

Feedback specifico

Visualizzato

Feedback generale

Visualizzato

Risposta corretta

Visualizzato

Storico delle risposte

Non visualizzata

Aggiorna opzioni di visualizzazione

Domanda 1

Risposta non
ancora data

Non valutata

2. Correttezza dei programmi ricorsivi

In relazione alla procedura *ufo* definita come segue per una rappresentazione binaria dell'argomento che inizia sempre con il bit più significativo uguale a 1 (una variante della versione discussa in classe):

```
(define ufo
  (lambda (bin)
    ; bin è una stringa di 0/1 che inizia con il bit '1' (bit più significativo = 1)
    (let (
      (k (- (string-length bin) 1))
    )
      (cond ((= k 0) ; bin = "1"
        1)
        ((char=? (string-ref bin k) #\0)
          (- (* 2 (ufo (substring bin 0 k))) 1))
        (else
          (+ (* 2 (ufo (substring bin 0 k))) 1))
        ))
    ))
```

si può dimostrare per induzione sulla lunghezza $n \geq 2$ della sequenza s di bit che per ogni stringa della forma $s = "1100 \dots 0"$, cioè con due bit '1' all'inizio seguiti da $n-2$ occorrenze di '0', si ha che:

$$(ufo\ s) \rightarrow 2^{n-1} + 1$$

2.1. In relazione alla dimostrazione impostata in questi termini:

- Formalizza la proprietà che esprime il caso / i casi base:

Assumendo come ipotesi induttiva che per una certa stringa $t = "1100 \dots 0"$ di lunghezza $k \geq 2$ si abbia:

$$(ufo\ t) \rightarrow 2^{k-1} + 1$$

- Formalizza la proprietà da dimostrare come passo induttivo:

[Nella formalizzazione puoi eventualmente utilizzare il simbolo $^$ per l'elevamento a potenza, \leq al posto di \leq e così via.]

[Ricomincia](#)
[Salva](#)
[Inserisci le risposte esatte](#)
[Invia e termina](#)
[Chiudi anteprima](#)
[Informazioni tecniche](#)
[Esporta la domanda nel formato Moodle XML](#)
[Minimizza tutto](#)

Domanda **1**Risposta non
ancora dataPunteggio max.:
1,00**2. Correttezza dei programmi ricorsivi****2.2.** Riporta nel riquadro sottostante la dimostrazione del caso / dei casi base e la dimostrazione del passo induttivo.**Dimostrazioni:**

Ricomincia

Salva

Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? - ►[Esporta la domanda nel formato Moodle XML](#)[Minimizza tutto](#)

Opzioni per il tentativo

Comportamento della domanda ?

Feedback differito

Punteggio massimo

1

Ricomincia con queste opzioni

Opzioni di visualizzazione

Se corretto

Visualizzato

Punteggio

Visualizza punteggio e max.

Cifre decimali nelle valutazioni

2

Domanda 1

Risposta non
ancora data

Non valutata

2. Correttezza dei programmi ricorsivi

In relazione alla procedura *ufo* definita come segue per una rappresentazione binaria dell'argomento che inizia sempre con il bit più significativo uguale a 1 (una variante della versione discussa in classe):

```
(define ufo
  (lambda (bin)
    ; bin è una stringa di 0/1 che inizia con il bit '1' (bit più significativo = 1)
    (let (
      (k (- (string-length bin) 1))
    )
      (cond ((= k 0) ; bin = "1"
        1)
        ((char=? (string-ref bin k) #\0)
          (- (* 2 (ufo (substring bin 0 k))) 1))
        (else
          (+ (* 2 (ufo (substring bin 0 k))) 1))
        ))
    ))
```

si può dimostrare per induzione sulla lunghezza $n \geq 2$ della sequenza s di bit che per ogni stringa della forma $s = "1011 \dots 1"$, cioè con i primi due bit "10" all'inizio seguiti da $n-2$ occorrenze di '1', si ha che:

$$(ufo\ s) \rightarrow 2^{n-1} - 1$$

2.1. In relazione alla dimostrazione impostata in questi termini:

- Formalizza la proprietà che esprime il caso / i casi base:

Assumendo come ipotesi induttiva che per una certa stringa $t = "1011 \dots 1"$ di lunghezza $k \geq 2$ si abbia:

$$(ufo\ t) \rightarrow 2^{k-1} - 1$$

- Formalizza la proprietà da dimostrare come passo induttivo:

[Nella formalizzazione puoi eventualmente utilizzare il simbolo $^$ per l'elevamento a potenza, \leq al posto di \leq e così via.]

Ricomincia

Salva

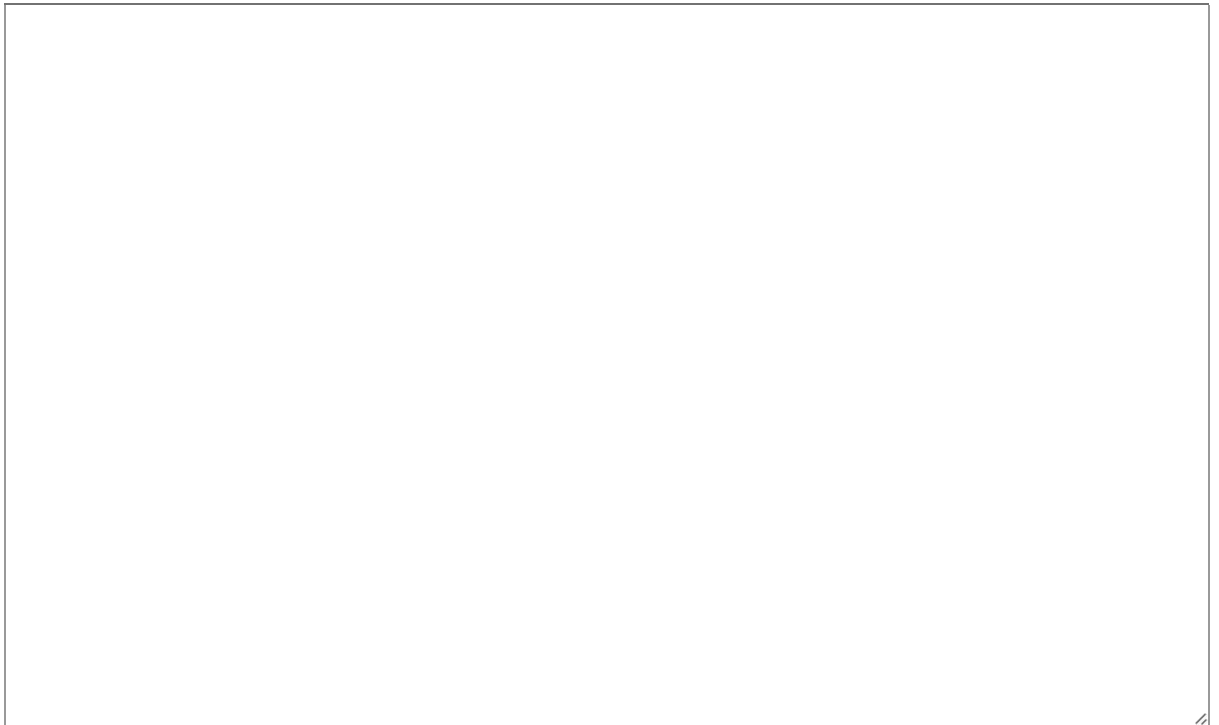
Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? ➤

[Esporta la domanda nel formato Moodle XML](#)
[Minimizza tutto](#)

Domanda **1**Risposta non
ancora dataPunteggio max.:
1,00**2. Correttezza dei programmi ricorsivi****2.2.** Riporta nel riquadro sottostante la dimostrazione del caso / dei casi base e la dimostrazione del passo induttivo.**Dimostrazioni:**

Ricomincia

Salva

Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? - ►[Esporta la domanda nel formato Moodle XML](#)[Minimizza tutto](#)

Opzioni per il tentativo

Comportamento della domanda ?

Feedback differito

Punteggio massimo

1

Ricomincia con queste opzioni

Opzioni di visualizzazione

Se corretto

Visualizzato

Punteggio

Visualizza punteggio e max.

Cifre decimali nelle valutazioni

2

Domanda 1

Risposta non
ancora data

Non valutata

3. Memoization

Una sequenza s di *double* si definisce *smorzantesi* (*damping*) se ogni suo elemento ha un valore che ricade *strettamente* all'interno dell'intervallo delimitato dai due elementi precedenti, quando ci sono entrambi. Formalmente:

$$\min(s[i-2], s[i-1]) < s[i] < \max(s[i-2], s[i-1]) \quad \text{per } i \geq 2.$$

Data una sequenza s , rappresentata da un array di *double*, il seguente programma ricorsivo in Java determina la lunghezza della sottosequenza smorzantesi più lunga di s (*llds* = *length of the longest damping subsequence*):

```
// Length of the Longest Damping Subsequence
public static int llds( double[] s ) {
    return lldsRec( s, 0, Double.NaN, Double.NaN ); // NaN = not a number (predefinito per double)
}

private static int lldsRec( double[] s, int i, double u, double v ) {
    if ( i == s.length ) { // coda di s vuota
        return 0;
    } else if ( Double.isNaN(u) || // u non è un numero
               ((Math.min(u,v) < s[i]) && (s[i] < Math.max(u,v))) ) {
        return Math.max( 1+lldsRec(s,i+1,v,s[i]), lldsRec(s,i+1,u,v) ); // s[i] può essere scelto o meno
    } else {
        return lldsRec( s, i+1, u, v ); // s[i] non può essere scelto
    }
}
```

3.1. Completa il programma riportato qui sotto, che applica una tecnica top-down di *memoization* per rendere più efficiente la computazione ricorsiva avviata da *llds*. Assumi a tal fine che la procedura *initStruct* (metodo statico) inizializzi correttamente la struttura di supporto per registrare i valori delle computazioni ricorsive effettuate.

```
public static int llds( double[] s ) {
    int n = s.length;
     mem = initStruct( n );

    return lldsRec( s, 0, n, n, mem );
}

private static int lldsRec( double[] s, int i, int j, int k,  mem ) {

    if ( mem  == UNKNOWN ) {

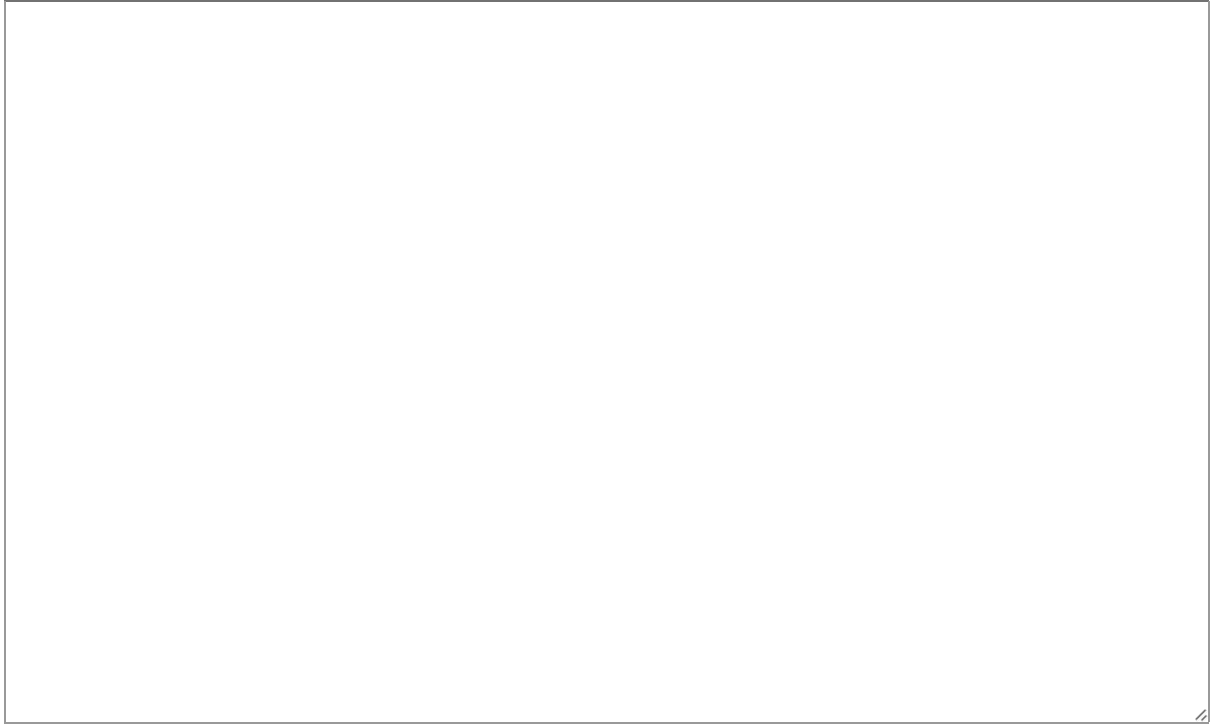
        int n = s.length;
        if ( i == n ) {
            mem  = 0;
        } else if (  ||
                   ((Math.min(s[j],s[k]) < s[i]) && (s[i] < Math.max(s[j],s[k]))) ) {
            mem  = ;
        } else {
            mem  = ;
        }
    }

    return mem ;
}

private static final int UNKNOWN = -1;
```

Domanda **1**Risposta non
ancora dataPunteggio max.:
1,00**3. Memoization**

3.2. Definisci opportunamente il metodo statico `initStruct` utilizzato per inizializzare la struttura di supporto relativa al programma che applica la tecnica di memoization.

Soluzione:

Ricomincia

Salva

Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? ➤[Esporta la domanda nel formato Moodle XML](#)[Minimizza tutto](#)

Opzioni per il tentativo

Comportamento della domanda ?

Feedback differito

Punteggio massimo

1

Ricomincia con queste opzioni

Opzioni di visualizzazione

Se corretto

Visualizzato

Punteggio

Visualizza punteggio e max.

Cifre decimali nelle valutazioni

2

Domanda **1**Risposta non
ancora data

Non valutata

4. Ricorsione e iterazione

Dato l'albero di Huffman specifico per un particolare documento, il seguente programma ricorsivo permette di calcolare il numero complessivo di bit della codifica di Huffman di quel documento:

```
public static int huffmanSize( Node root ) {
    return huffmanSizeRec( root, 0 );
}
private static int huffmanSizeRec( Node n, int d ) {
    if ( n.isLeaf() ) {
        return d * n.weight();
    } else {
        return huffmanSizeRec( n.left(), d+1 ) + huffmanSizeRec( n.right(), d+1 );
    }
}
```

4.1. Completa la procedura impostata qui sotto per trasformare la computazione ricorsiva di *huffmanSizeRec* in una computazione iterativa, utilizzando uno stack di *Frame*.

```
public static int huffmanSizelter( Node root ) {
    Stack<Frame> stack = new Stack<Frame>();
    stack.push( new Frame(root,0) );
    int hSize = 0;
    do {
        Frame current =  ;

        Node n = current.node ;
        int d = current.depth ;
        if (  ) {
            hSize =  ;
        } else {
             ;
             ;
        }
    } while (  );

    return hSize;
}
```

Ricomincia

Salva

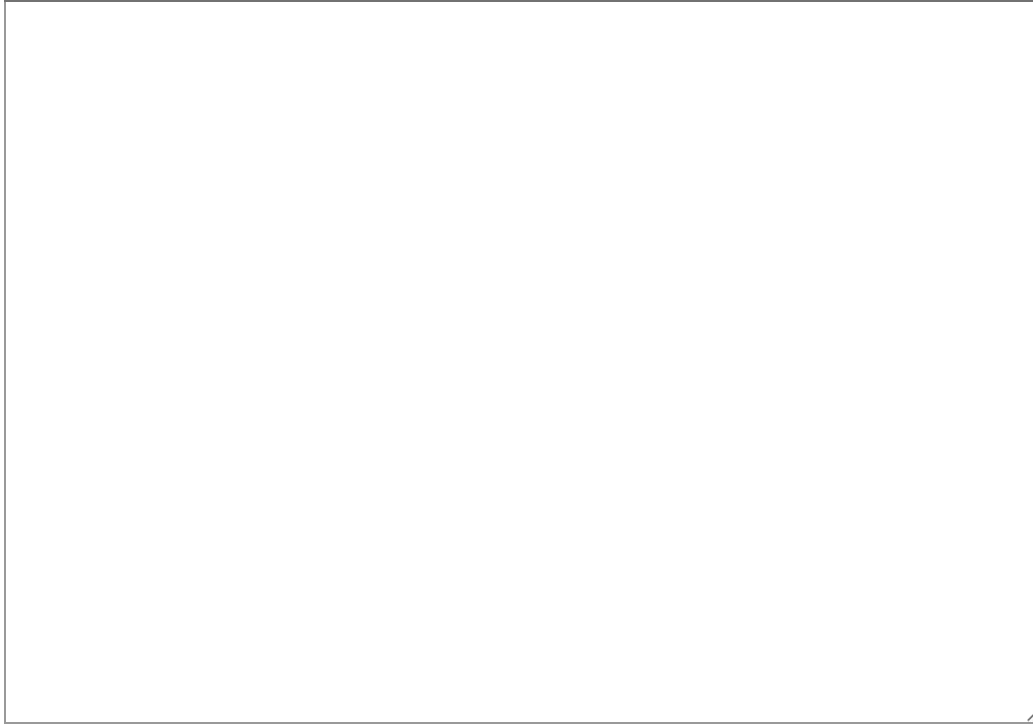
Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? - ►

[Esporta la domanda nel formato Moodle XML](#)
[Minimizza tutto](#)
[Opzioni per il tentativo](#)

Domanda **1**Risposta non
ancora dataPunteggio max.:
1,00**4. Ricorsione e iterazione****4.2.** Definisci opportunamente la classe *Frame* per rappresentare gli oggetti da inserire nello stack.**Soluzione:**

Ricomincia

Salva

Inserisci le risposte esatte

Invia e termina

Chiudi anteprima

[Informazioni tecniche](#) ? - ►[Esporta la domanda nel formato Moodle XML](#)[Minimizza tutto](#)

Opzioni per il tentativo

Comportamento della
domanda

Feedback differito

Punteggio massimo

1

Ricomincia con queste opzioni

Opzioni di visualizzazione

Se corretto

Visualizzato

Punteggio

Visualizza punteggio e max.

Cifre decimali nelle valutazioni

2