Corso di Programmazione

Esame del 22 Gennaio 2018

cognome e nome

1. Verifica formale della correttezza

In relazione alla procedura `ufo`, riportata qui a fianco, si può dimostrare che per ogni intero $k \geq 0$:

$$(\texttt{ufo}\ 3 \cdot 2^k) \quad \rightarrow \quad 1 + 2^{k+1}$$

Dimostra questa proprietà per induzione sui valori di k attenendoti allo schema delineato qui sotto.

```
(define ufo
  (lambda (n)
    (cond ((= n 1)
           1)
          ((even? n)
           (- (* 2 (ufo (quotient n 2))) 1))
          (else
           (+ (* 2 (ufo (quotient n 2))) 1))
          )))
```

- Formalizza la proprietà generale da dimostrare:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il caso / i casi base:

- Dimostra il passo induttivo:

2. Programmazione in Scheme

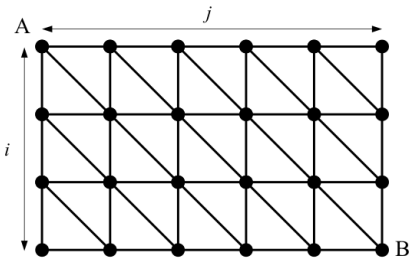
Dato un intero $n \geq 0$, la procedura `powers-of-two` restituisce la lista delle potenze di due distinte che lo compongono, la cui somma è n, ordinate in ordine decrescente. In altri termini, ciascuna delle potenze di due rappresentate nella lista corrisponde a uno dei bit '`1`' della notazione binaria di n, a partire da quello più significativo. Esempi:

```
(powers-of-two  0)  →  ()              (powers-of-two 16)  →  (16)
(powers-of-two  1)  →  (1)             (powers-of-two 22)  →  (16 4 2)
(powers-of-two  9)  →  (8 1)           (powers-of-two 53)  →  (32 16 4 1)
```

Definisci un programma in Scheme per realizzare la procedura `powers-of-two`.

3. Ricorsione ad albero

Completa la definizione della procedura `manhattan-var`, progettata per risolvere una variante del "problema di Manhattan" in cui i nodi sono connessi, oltre che dai consueti tratti orizzontali e verticali, anche da tratti diagonali che scendono verso destra come illustrato nella figura a fianco. Interessa conoscere in quanti modi diversi ci si può spostare dal nodo A al nodo B lungo un percorso di lunghezza minima che attraversi *esattamente k* tratti diagonali (e non di più: osserva che gli spostamenti in diagonale abbreviano i percorsi rispetto al caso di spostamenti orizzontali a destra e verticali in basso, ma non è consentito utilizzarne più di k; se $k = 0$, in particolare, ci si riconduce alla soluzione del problema originale). Esempi:




```
(manhattan-var 3 2 0) → 10   (manhattan-var 3 2 2) → 3   (manhattan-var 2 2 2) → 1
```

```
(define manhattan-var          ; val: intero
  (lambda (i j k)              ; i, j, k: interi non negativi tali che k ≤ i e k ≤ j
    (let (
          (x (if (< k i) (manhattan-var (- i 1) j k) 0))

          (y (if ..... (manhattan-var i (- j 1) k) 0))

          (z (if (< 0 k) (manhattan-var ..... ) 0))
          )
      (if (or (= i 0) (= j 0))

          .....

          (+ ..... )
          ))
    ))
```

4. Memoization

Applica la tecnica *top-down* di memoization alla procedura ricorsiva dell'esercizio **3** al fine di realizzare un programma più efficiente in Java. (Nel caso in cui la soluzione dell'esercizio **3** non fosse completa puoi omettere i dettagli relativi alle parti mancanti, sostituendole con dei riferimenti simbolici in corrispondenza degli spazi punteggiati e utilizzando i simboli che hai introdotto nei punti appropriati del codice in Java.)

5. Ricorsione e iterazione

Dati un intero *k* e un *albero di Huffman*, il metodo statico `findByCodeLength` restituisce la stringa dei caratteri i cui codici di Huffman hanno lunghezza *k*. In particolare, la visita dell'albero, finalizzata alla ricerca dei caratteri con la proprietà desiderata, è realizzata attraverso uno schema iterativo basato su uno *stack* di oggetti di tipo `Frame`, corrispondente alla classe introdotta in alto nel riquadro. Completa la definizione del metodo `findByCodeLength`.

```
class Frame {   // Definisce il tipo degli elementi inseriti nello stack

  private final Node nde;
  private final int dph;

  public Frame( Node n, int h ) {
    nde = n;  dph = h;
  }

  public Node node() {  return nde;  }

  public int depth() {  return dph;  }

}   // class Frame



   . . .      . . .


public static String findByCodeLength( int k, Node root ) {

  Stack<Frame> stack = new Stack<Frame>();
  stack.push( new Frame(root,0) );

  String found = "";

  do {
    Frame current = stack.pop();

    Node n = ..................................................................... ;

    int h = current.depth();

    if ( ..................................................................... ) {

      if ( ..................................................................... == k ) {

        found = found + n.character();
      }

    } else if ( ..................................................................... < k ) {

      stack.push( new Frame( ..................................................... ) );

      ..................................................................... ;

    }

  } while ( ..................................................................... );

  return found;
}
```

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nelle intestazioni e su ciascun ulteriore foglio che intendi consegnare.