Corso di Programmazione

Esame del 3 Settembre 2019

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmazione in Scheme

Data una lista di interi s, la procedura 1 is calcola la *lista* delle sottosequenze crescenti più lunghe di s. Per esempio:

```
(1is '(27 90 7 29 49 8 53 1 28 6)) \rightarrow ((7 29 49 53) (27 29 49 53))
```

Il programma si basa sull'idea sviluppata nell'esercizio di laboratorio su questo tema. Il terzo argomento di lis-rec rappresenta un prefisso *rovesciato* (per ragioni di efficienza) di qualche soluzione.

La lista restituita da lis-rec deve quindi contenere liste di interi, tutte della stessa lunghezza. Data una lista con queste caratteristiche, il compito della procedura best è quindi di mettere insieme *tutte e sole* le liste che fanno parte della soluzione per gli argomenti s, t, p.

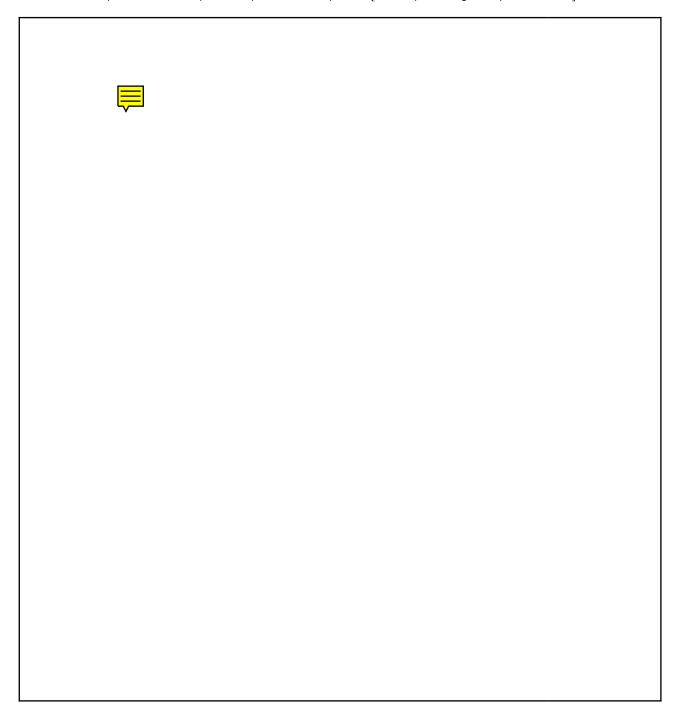
Scrivi un programma in Scheme per realizzare la procedura best, tenendo conto del contesto in cui è applicata.

2. Programmazione in Scheme

Considera il programma discusso a lezione per determinare la *rappresentazione ternaria bilanciata* (BTR) di un numero intero *n*, riportato qui sotto per comodità:

Scrivi un programma in Scheme, basato sulla procedura btr-list per calcolare la lista dei "contributi" delle cifre ternarie bilanciate alla composizione del valore n, ciascun contributo essendo pari al valore della cifra stessa moltiplicato per il peso (cioè la potenza di tre) corrispondente. Per esempio:

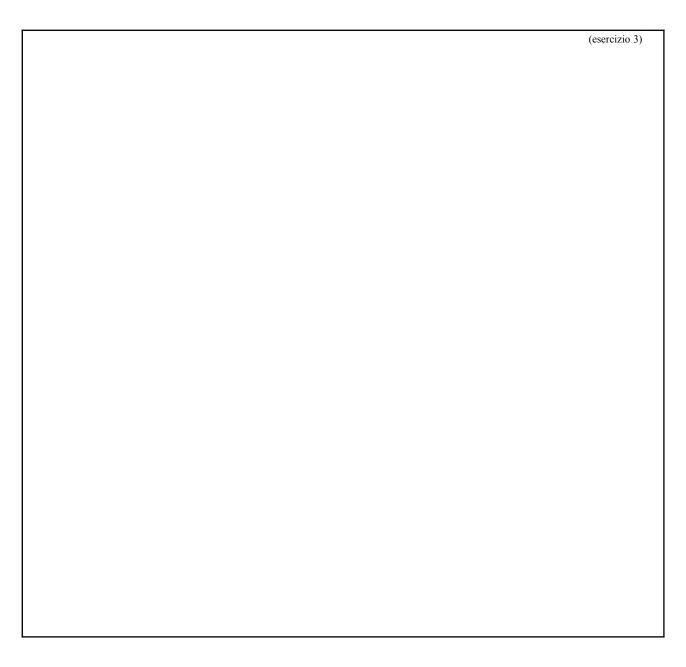
```
(btr-list -25) \rightarrow (-27 0 3 -1) [dove (btr-rep -25) \rightarrow "-.+-"]
```



3. Programmazione dinamica

Applica la tecnica *bottom-up* di *programmazione dinamica* per realizzare una versione più efficiente del metodo statico q riportato qui di seguito:

```
public static long q( int i, int j, int k ) { //i,j,k>=0
  long x = ( i < 2 ) ? i : q( i-2, j, k );
  long y = ( j < 2 ) ? j : q( i, j-2, k );
  long z = ( k < 2 ) ? k : q( i, j, k-2 );
  long m = x + y + z;
  return ( m == 0 ) ? 1 : m;
}</pre>
```



4. Ricorsione e iterazione

Il seguente programma ricorsivo in Java calcola l'elenco di tutte le permutazioni di una stringa u, rappresentandolo attraverso la struttura predefinita Vector<String>.

Completa la definizione del metodo statico permIter, riportato nel riquadro sottostante, che trasforma la struttura ricorsiva di permRec in una struttura iterativa basata sull'uso di *stack*.

```
public static Vector<String> permIter( String u ) {  // u non vuola

Vector<String> p = new Vector<String>();
int n = u.length(), k;

Stack<String> s = new Stack<String>();
Stack<Integer> t = new Stack<Integer>();
s.push( u ); t.push( 0 );
do {

if ( k == n-1 ) {
   p.add( u );
} else {

   hyhile (!s.empty());
return p;
}
```

5. Correttezza dei programmi e invarianti

Completa il programma riportato nel riquadro per calcolare la quarta potenza di un intero positivo n utilizzando solo somme e confronti. A tal fine, le espressioni introdotte negli spazi previsti devono garantire che le proprietà invarianti siano effettivamente soddisfatte.

```
public static int pow4( int n ) { // Pre: n > 0

int x = ______, y = 0;

int z = _____, u = ____;

while ( x < u ) { // Inv: (x ≤ n²) ∧ (y = x²) ∧ (z = 2x+1) ∧ (u = min(y+z,n²))

x = x + 1;

y = y + _____;

z = z + _____;

if ( ______) {

u = y + z;

}}

return y; // Post: y = n⁴

}
```