Lab2: Recursivitate

09 March 2023 16:26

3 tipuri recursivitate → stivă
→ coadă
→ arborescentă

① REC. PE STIVĂ

→ păstrează informații pe stivă pe parcursul avansului în recursivitate

→ Construiește rezultatul la revenirea din recursivitate

Ex:

```
(define (list-sum L)
  (if (null? L)
      0
      (+ (car L) (list-sum (cdr L)))))
```

list-sum '(2 4 5)

2 + list-sum '(4 5)

4 + list-sum '(5)

5 + list-sum '()

11

2+9

4+5

5+0

Complexitate — temporală : $O(n)$, n = lungime L
— spațială : $O(n)$

⇒ ineficientă spațial

⇒ poate genera stack-overflow

② REC. PE COADĂ

→ rezultatul se construiește pe parcursul avansului în recursivitate

TAIL-CALL OPTIMIZATION = apelul recursiv curent de pe stivă se înlocuiește cu următorul

→ rezultatul se construiește într-un parametru — acc (în general)

Ex:

```
(define (list-sum-helper L acc)
  (if (null? L)
      acc
      (list-sum-helper (cdr L) (+ acc (car L)))))

(define (list-sum-tail L)
  (list-sum-helper L 0))
```

list-sum '(2 4 5) 0

list-sum '(4 5) 2

list-sum '(5) 6

list-sum '() 11

⇓

11

Complexitate — temporală : $O(n)$, n = length(L)
— spațială : $O(1)$

STIVĂ vs COADĂ

Ex: înmulțiți elementele unei liste cu 10 (reverse acc) (append acc (list (car L)))

Ex: Înmulțiți elementele unei liste cu 10

(reverse acc)

(append acc (list (car L)))

```
(define (mult10 L)
  (if (null? L)
      '()
      (cons (* 10 (car L)) (mult10 (cdr L)))))
```

```
(define (mult10-tail-helper L acc)
  (if (null? L)
      acc
      (mult10-tail-helper (cdr L) (cons (* 10 (car L)) acc))))
(define (mult10-tail L)
  (mult10-tail-helper L '()))
```

(mult10 '(1 2 3)) => '(10 20 30)

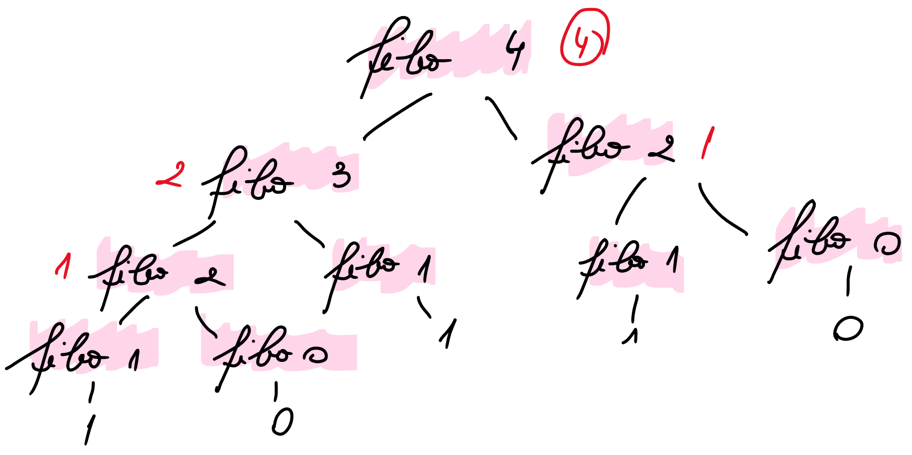
mult10-tail '(1 2 3) '()
↓
mult10-tail '(2 3) '(10)
↓
mult10-tail '(3) '(20 10)
↓
mult10-tail '() '(30 20 10)
↓
'(30 20 10) → inversat

Soluții: 1) ineficientă: append
2) eficientă: cons + inversez rezultatul

③ REC. ARBORESCENTĂ

→ cel puțin 2 sau mai multe apeluri rec. independente

Ex:
fibo 0 = 0
fibo 1 = 1
fibo n = fibo(n-1) + fibo(n-2)



EXERCIȚII

① REVERSE

→ Mină:

rev [] = []

rev [x : L] = rev(L) ++ [x]

→ Coadă:

rev [] acc = acc

rev [x:L] acc = rev L (x : acc)

② LESSER

→ rec. pe stivă

Vezi - ex mult10

③ GREATER

→ rec. pe coadă

Vezi - ex. mult10-tail

④ REMOVE - DUPLICATES - LEFT

'(1 2 3 2 4 3) → '(1 2 3 4)

→ rec. pe stivă

L: '(1 2 3 2 4 3) → '(2 3 2 4 3) → '(3 2 4 3) → '(2 4 3) → '(4 3) → '(3) → '()

'(1 2 4 3) ← '(2 4 3) ← '(2 4 3) ← '(2 4 3) ← '(4 3) ← '(3) '()

greşit

→ rec. pe coadă

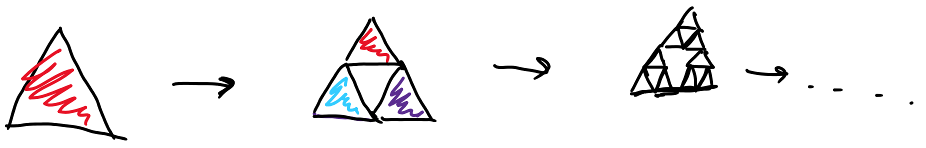
L: '(1 2 3 2 4 3) → '(2 3 2 4 3) → '(3 2 4 3) → '(2 4 3) '(4 3) '(3) → '()

acc = '() ; '(1) '(1 2) '(1 2 3) '(1 2 3) → '(1 2 3 4) → '(1 2 3 4)

⑤ REMOVE - DUPLICATES - RIGHT

Vezi ④ → stivă

⑥ SIERPINSKI



[1 🟥, 2 🟦, 3 🟪, 4 🟩, 5 🟦]

(sierpinski n L colors)

nr. iterații   lung. latură   lista de culori