

Examen Algoritmen & Datastructuren I

Academiejaar 2017–2018 — 1^{ste} zittijd

Maandag 29 januari 2018

Het examen duurt 3,5 uur en is een gesloten boek examen. Alles behalve je schrijfgerief (zakken, jassen, GSM, ...) hoort thuis aan de zijkant van de aula. Het gebruik van elektronische toestellen is ten strengste verboden. Zorg dat je GSM uitstaat wanneer het examen begint. Leg je studentenkaart op je tafel.

Los elke vraag op op een apart blad en geef duidelijk aan welke vraag je aan het beantwoorden bent. Vergeet niet om op elk blad je naam, rolnummer en richting te vermelden.

Schrijf nooit enkel je einduitkomsten op maar ook de redeneringen, tussenstappen en eventuele formules die je gebruikt hebt.

Veel succes!

Vraag 1

- a) Bewijs dat een heap met n elementen $\lceil \frac{n}{2} \rceil$ leafs heeft.
- b) Voor welk algoritme uit de cursus is deze eigenschap van belang?

Vraag 2

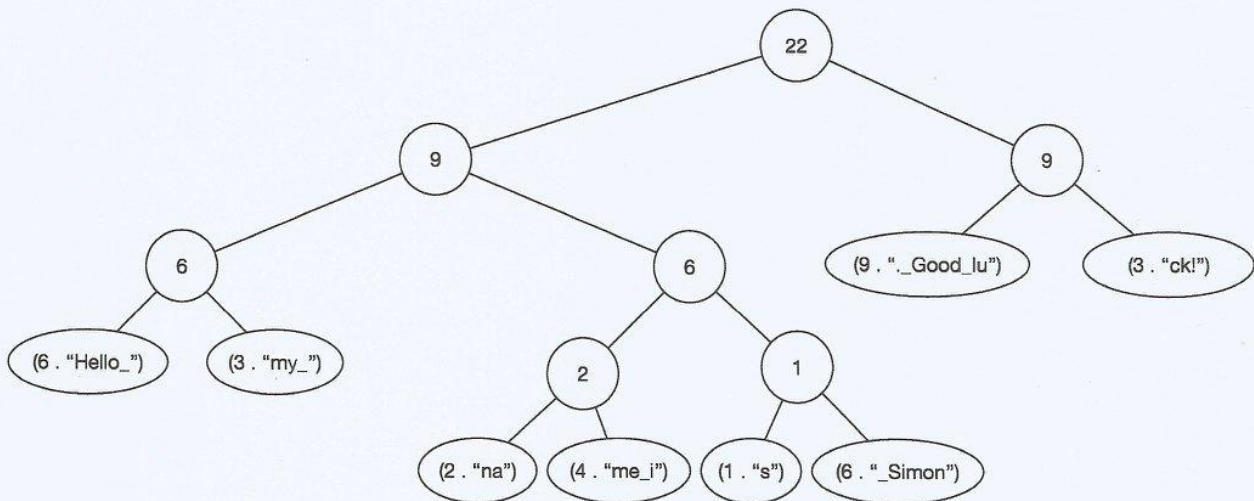
Implementeer **insertion sort** voor gewone Scheme-lijsten op een functionele manier. D.w.z. dat je algoritme een nieuwe gesorteerde lijst moet teruggeven i.p.v. de originele lijst destructief te veranderen bij het sorteren. Let erop dat je algoritme generisch is (dus dat je lijsten van alle mogelijke types kan sorteren)!

Is je algoritme stabiel? Leg uit.

Vraag 3

Instanties van de string datastructuur die we hebben besproken in de cursus zijn niet aanpasbaar. Voor sommige doeleinden is het echter wel nodig om strings aan te passen. Hierbij denken we bijvoorbeeld aan tekstverwerkers zoals Word of Notepad of programmeeromgevingen zoals Dr-Racket. In zulke programma's worden grote stukken tekst aangepast. Om zo'n programma's op een performante manier te schrijven kan men beter gebruik maken van een datastructuur zoals een **rope** om tekst voor te stellen.

Een rope is een binaire boom die gebruikt wordt om een tekst voor te stellen. Onderstaande figuur toont een voorbeeld van een instantie van een rope.



Deze datastructuur is interessant omdat de operaties om tekst toe te voegen, te verwijderen en aan te passen kunnen plaatsvinden met een goede performantie-karakteristiek. Een rope heeft de volgende kenmerken:

- De yield (t.t.z. alle bladeren van links naar rechts) komt overeen met de opgeslagen tekst. In ons voorbeeld is dat "Hello_my_name_is_Simon._Good_luck!". Elke leaf-node bevat een stuk van de tekst. Deze stukjes tekst zijn gewoon traditionele strings zoals we die hebben besproken in de cursus. Daarnaast bevat de leaf-node een getal om de lengte aan te geven van het stuk tekst dat wordt opgeslagen in de leaf. De waarde van een leaf node is een cons-cel waarbij de car de numerieke waarde bevat (i.e. de lengte van het stuk tekst) en de cdr de eigenlijke string.
- Alle andere nodes bevatten een getal dat gelijk is aan de som van de lengte van alle strings die worden opgeslagen in hun **linkerdeelboom**.

Voor deze opdracht moet je drie operaties implementeren. Doe dit zo efficiënt mogelijk.

- a) (rope-length rope): deze procedure krijgt als enige parameter een rope mee. De procedure geeft de lengte van de tekst terug die wordt gerepresenteerd door de rope.

```
> (rope-length testrope)
34
```

- b) (`rope-ref rope index`): deze procedure verwacht als eerste argument een rope. Als tweede argument verwacht de procedure een index. De procedure geeft het karakter terug dat op positie `index` staat in de tekst die door de rope gerepresenteerd wordt. Een voorbeeld:

```
> (rope-ref testrope 7)
#\y
```

- c) (`delete-if rope index proc`): deze procedure krijgt drie parameters mee: een rope, een index en een procedure. Het procedureel type van `proc` is als volgt: `char -> boolean`. De procedure verwijdert het karakter op de positie `index`, maar alleen wanneer de oproep van `proc` met dat karakter resulteert in `#t`. Wanneer het resultaat van de procedure-oproep `#f` is, gebeurt er niets. Let erop dat de waardes van de nodes in de binaire boom ook correct worden geüpdatet.

In de bijlage vind je de code voor de in de cursus geziene implementatie van het `binary-tree` ADT, het skelet voor de implementatie van het rope ADT alsook de voorbeeldcode die de rope creëert zoals die is voorgesteld in de figuur.

```

#!r6rs
(library
 (rope)
 (export new-rope? rope-length rope-ref delete-if)
 (import (rnrs base)
         (rnrs io simple)
         (prefix (a-d tree binary-tree) bt:)
         (sfri :9)))

(define-record-type rope
  (new tree)
  rope?
  (tree tree tree!))

(define (rope-length rope)
  ;Defining 3a
  )

(define (rope-ref rope index)
  ;Defining 3b
  )

(define (delete-if rope index proc)
  ;Defining 3c
  )

(define testrope
  (new (bt:new
        22
        (bt:new
          9
          (bt:new
            6
            (bt:new (cons 6 "Hello_") bt:null-tree bt:null-tree)
            (bt:new (cons 3 "my_") bt:null-tree bt:null-tree))
          (bt:new
            6
            (bt:new
              2
              (bt:new (cons 2 "na") bt:null-tree bt:null-tree)
              (bt:new (cons 4 "me_i") bt:null-tree bt:null-tree))
            (bt:new
              1
              (bt:new (cons 1 "s") bt:null-tree bt:null-tree)
              (bt:new (cons 6 "_Simon") bt:null-tree bt:null-tree))))
        (bt:new
          9
          (bt:new (cons 9 "._Good_lu") bt:null-tree bt:null-tree)
          (bt:new (cons 3 "ck!") bt:null-tree bt:null-tree))))))

```

```
#!r6rs
```

```
(library
  (binary-tree)
  (export null-tree new null-tree? value value! left left! right right!)
  (import (rnrs base)
           (srfi :9)
           (rnrs mutable-pairs)))

(define-record-type tree
  (new v l r)
  tree?
  (v value value!)
  (l left left!)
  (r right right!))

(define null-tree ())

(define (null-tree? node)
  (eq? node null-tree))
```

Vraag 4

Veronderstel dat we een hash-tabel hebben van grootte 16 (i.e. $M = 16$) en als hashfunctie $h(k) = 4 * k$.

- a) Verdeel de elementen **21, 97, 420, 61, 179, 151, 101** over de hash-tabel in deze volgorde met behulp van de onderstaande strategieën. Toon aan of leg uit wat er gebeurt in het geval van collisions. Verwijder vervolgens elementen 21 en 179 in deze volgorde. Toon ook hier aan wat er gebeurt.
- 1) External Chaining.
 - 2) Linear Probing waar $c = 4$. Voor het verwijderen van de elementen mag je ervan uitgaan dat we gebruik maken van de tombstone-methode.
 - 3) Double rehashing waar $h_2(k) = 11 - (k \bmod 11)$.
- b) Welke in de cursus bestudeerde problemen zie je opduiken? Indien je een probleem vaststelt, benoem het en leg uit. Hoe zou men dit kunnen oplossen? Geef voor elk probleem dat je hebt vastgesteld duidelijk aan bij welke strategie(ën) dat probleem opduikt.