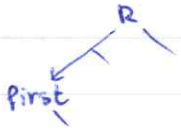
Universiteit Leiden
Leiden Institute of Advanced Computer Science (LIACS)

Course: Datastructuren
Date: 7-1-2021
Teacher: Hoogeboom

Name:
Study Programme:
Student ID number:

1 a)
- node = R
  while (node → left != nil){
      node = node → left
  }
  ofwel: ga zo ver mogelijk naar links



- if (curr → right !sthread){
      opvolger = curr → right
  }
  else {
      opvolger = curr → right
      while (opvolger → left != nil){
          opvolger = opvolger → left
      }
  }
  ofwel: als rechterkind draad, volg dan draad
  als rechterkind geen draad, dan meest linker knoop van rechtersubboom

b) - node = R ofwel: begin bij root
- if (curr → left != nil){
      opvolger = curr → left
  }
  else if (curr → right != nil){
      opvolger = curr → right
  }

```
else {
  while (curr → right isthread) {
    ~~opvolger~~ curr = curr → right        \\ is thread
  }
  ~~opvolger = opvolger → right~~
  opvolger = curr → right
}
```

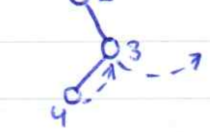
ofwel: als curr kind heeft,
volg kind (eerst links, dan rechts)
als geen kind: volg draden en ga naar
rechterkind ~~([illegible])~~

c) - node = R

```
while (node → left != nil) {
    node = node → left
}
while (node → right != nil) {
    node = node → right
}
```

ofwel: ga eerst zover mogelijk naar links
en dan zo ver mogelijk naar rechts

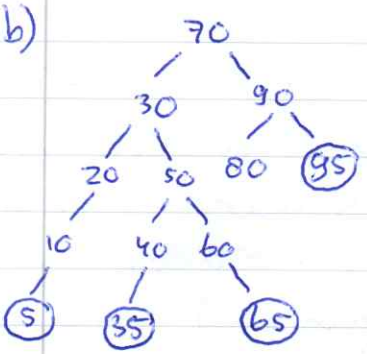
- Bekijk de volgende boom met draden:



De post order volgorde is:
4, 3, 2, 1
Maar van knoop 3 kan je nooit met alleen takken en draden naar knoop 2, want de draden lopen alleen naar rechts

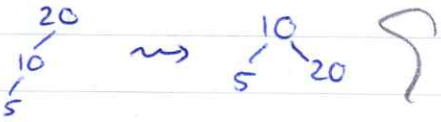
2 a) Een AVL-boom is een binaire zoekboom (dus knopen in linkersubboom zijn kleiner en knopen in rechtersubboom zijn groter voor elke knoop)
waarvoor voor elke knoop geldt:
De hoogtes van de linker- en rechter-subboom verschillen maximaal 1 van elkaar

b)



knoop 5

onbalans bij knoop 20 (LL)
dus enkele rotatie om 20 naar rechts



knoop 35

onbalans bij 70 (LR)
dus dubbele rotatie:
(eerst naar links om 30, dan naar rechts om 70)

knoop 65

onbalans bij 70 (LR), dus dubbele rotatie net als hiervoor

knoop 95

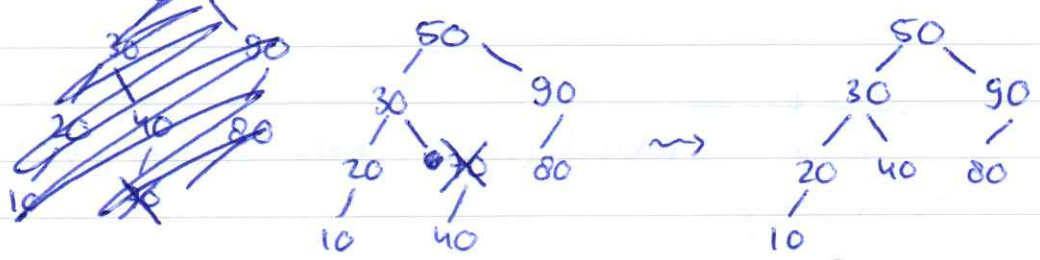
geen onbalans, dus geen rotaties nodig

c) • (binair) toevoegen kan direct door naar de goede plek te lopen (links als kleiner, rechts als groter), maar bij verwijderen moet, als de te verwijderen knoop geen blad is, de knoop eerst met zijn voorganger (of opvolger) verwisseld worden* ~~totdat omdat we alleen bladeren hij een~~ omdat we alleen bladeren willen verwijderen

* tot hij een blad is

• (AVL) bij toevoegen in een AVL-boom kan de boom uit balans raken, maar dat is altijd in 1 keer op te lossen met een (dubbele) rotatie
Bij verwijderen kan het echter voorkomen dat de boom na rotatie nog steeds uit balans is, en moet er dus nog meer geroteerd worden
vb: fibonacci boom

d) manier 1: verwissel met 50 en ~~verwijder met 50~~ schuif daarna 40 omhoog

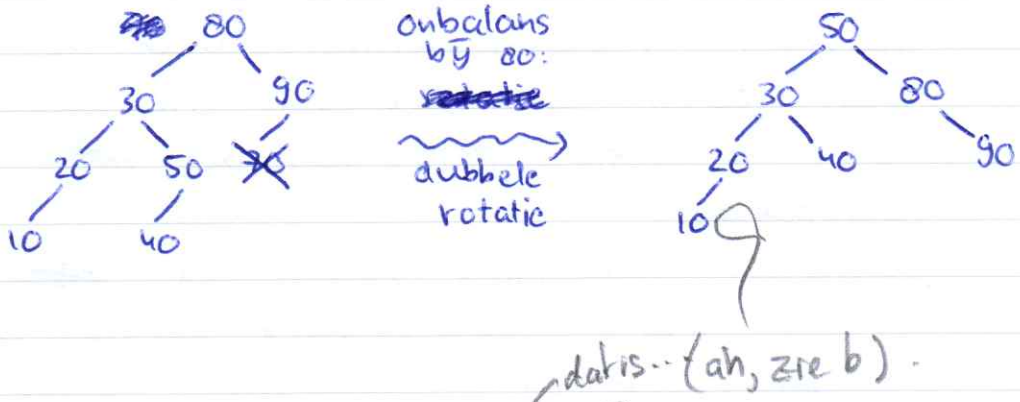


boom is nog in balans

Universiteit Leiden
Leiden Institute of Advanced Computer Science (LIACS)

Course: Datastructuren
Name:
Date: 7-1-2021
Study Programme:
Teacher: Hoogeboom
Student ID number:

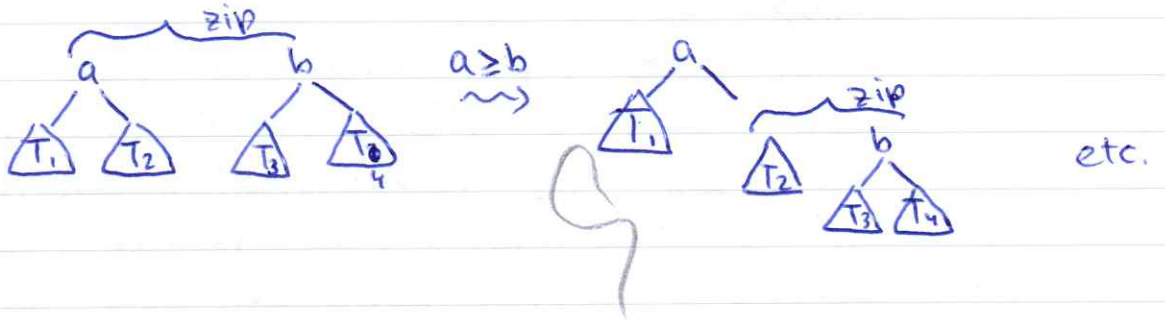
2 d) manier 2: verwissel met 80



dat is.. (ah, zie b).

3 a) Leftist tree = een heap waarbij geldt dat de null-path-length (= kortste lengte tot een extern blad) van linkerkind groter dan of gelijk aan de npl van rechterkind

Bij zippen worden twee (sub)bomen recursief in elkaar geritst: kies steeds de grootste wortel, laat de linkersubboom daar aan zitten en rits de rechtersubboom met de overgebleven boom (dus die met de kleinere wortel)



b) In een PQ hebben alle elementen een prioriteit
Stop de elementen in de Leftist tree zdd alle knopen een lagere prioriteit hebben dan hun ouders
De operaties van de PQ werken nu als volgt: ~~Init: initialiseer lege~~
getmax: return wortel van de boom
add: voeg een element toe door dit ~~(herhaaldelijk)~~ met de boom te zippen
~~delete~~:
delete: verwijder knoop en zip vervolgens de twee subbomen (links en rechts) met elkaar
(Init: initialiseer lege boom. isEmpty: check of boom leeg is)

c) (i) die lengte is precies k
(ii) als de wortel npl k heeft, bevat dus het meest rechter pad k knopen (zie i)
en voor elke knoop geldt dat het linkerkind een minstens even grote npl moet hebben (want leftist) als het rechterkind
~~En ook de linkersubboom moet dat~~
Dus de npl via elk pad vanaf de wortel moet minstens k zijn
Dus het minimale aantal knopen is als de boom helemaal gevuld is en dit zijn voor een npl van k precies $2^k - 1$ knopen

(iii) het meest rechter pad moet lengte k hebben, maar de linkersubboom kan oneindig groot worden en dan gelden nog steeds de leftist tree eigenschappen

4 a) Als het patroon is: $\underbrace{AAA\ldots AAB}_{m}$
en de tekst is:
$\underbrace{AAA\ldots AAAB}_{n}$

waarbij natuurlijk $n \geq m$, dan presteert het naieve zoekalgoritme slecht, omdat hij steeds één letter opschuift en dan weer het hele patroon moet langsgaan tot het bij de laatste letter fout gaat
Dit heeft dus complexiteit $\mathcal{O}(m \cdot n)$

b)
```
FLink(1) = 0
for (pos is 2 to n){
    Fail = FLink(pos - 1)
    while (FLink(Fail) > 0 && P(Fail) != P(pos)){
        Fail = FLink(Fail)
    }
    FLink(pos) = Fail + 1
}
```

ofwel: volg failure links van 1 positie ervoor tot dezelfde letter wordt gevonden en tel er dan nog 1 bij op

4 c)

	1	2	3	4	5	6	7	8
	A	B	C	A	B	A	B	C
Flink	0	1	1	1	2	3	2	3

d) T = A B C A B C A B A A B C A B A B B A B C

A B C A B $\underline{A_6}$

FL(6)=3 C_3 A B A $\underline{B_{67}}$

FL(7) = 2 $\underline{B_2}$

FL(2) = 1 A_1 B C A B A B $\underline{C_8}$

FL(8) = 3 $\underline{C_3}$

FL(3) = 1 $\underline{A_1}$

FL(1) = 0 A B C

Foute letters staan onderstreept. Er wordt dan op de volgende regel volgens de failure links verdergegaan met het vergelijken van het patroon met de tekst