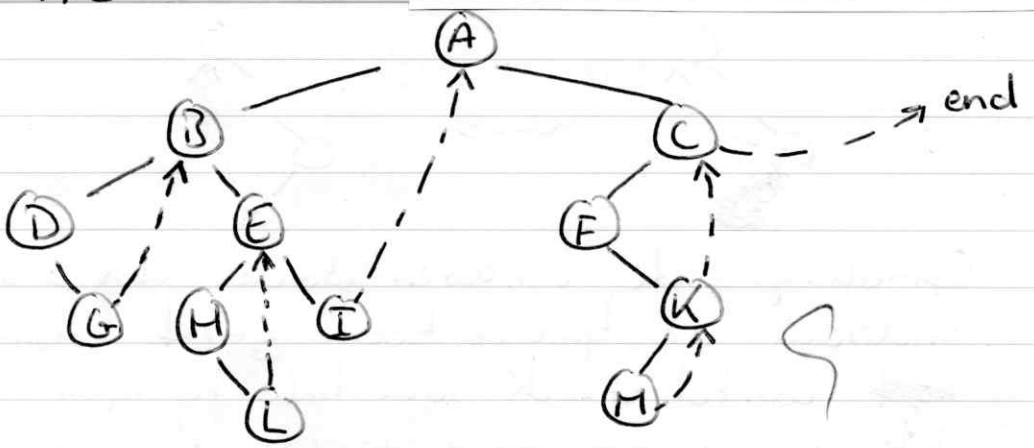
Course: Datastructuren
Name:
Date: 13-jan-2022
Study Programme:
Teacher: Moogeboom
Student ID number:

Blad 1/2

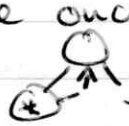
(1) a)



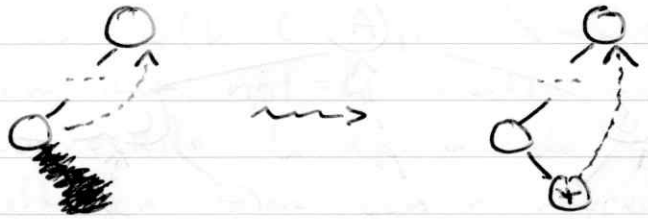
b)
```
// loop naar eerste knoop
cur := Root
while cur.left ≠ nil {
    cur := cur.left
}
while cur ≠ end {
    visit(cur)
    if cur.isThread {
        cur := cur.Right
    } else {
        cur := cur.Right
        while cur.left ≠ nil {
            cur := cur.left
        }
    }
}
```

✓ Volg BZB wandeling

c) Er zijn twee mogelijkheden: nieuwe knoop is rechter- of linkerkind. Geef nieuwe knoop aan met (*)
* linkerkind: volgend in inorde wandeling is de ouder, dus een draad wordt toegevoegd van (*) naar de ouder. Andere draden / kanten blijven hetzelfde.



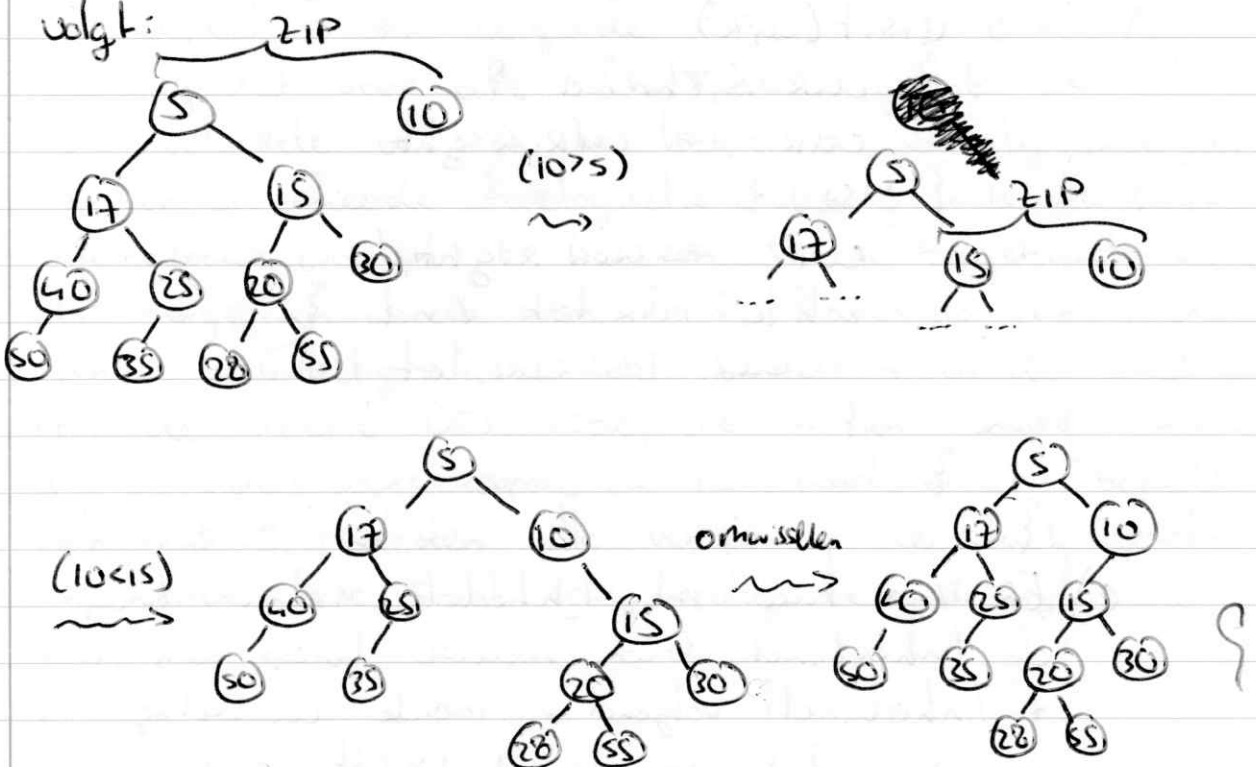
* RechterKind: draad van ouder wordt veranderd in een tak naar het nieuwe kind en er komt een draad van het nieuwe kind naar de plek waar de draad van de ouder eerst heen wees:



Opmerking: de kinderen worden dus altijd onderaan de binaire boom gezet door vanaf de wortel steeds naar links te lopen als het kind kleiner is en naar rechts te lopen als het kind groter is.

(2) a) De operatie DecreaseKey wordt gebruikt om een element een hogere prioriteit te geven in een min-queue. Deze wordt niet altijd bij de specificatie van de ADT Priority Queue meegenomen. ✓ lokale nodig

b) (i) We ritsen de boom samen met (10). Als volgt:

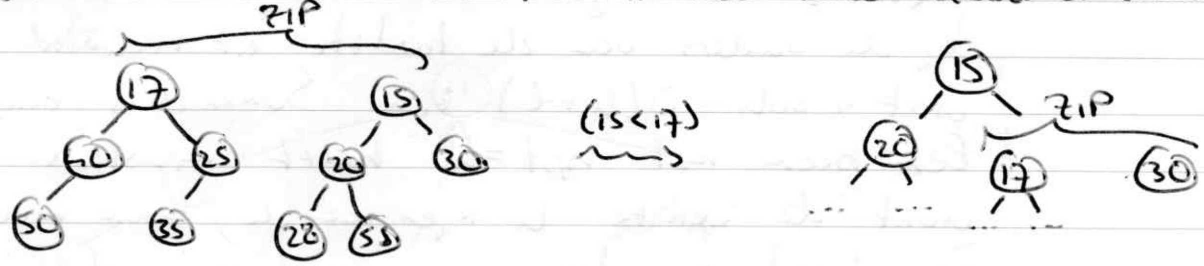


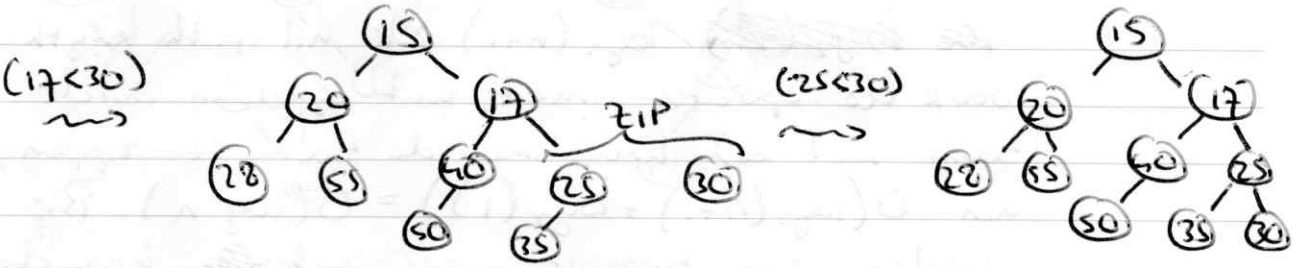
npl is rechts overal kleiner.

Merk op: De subboom met wortel is leeg hier

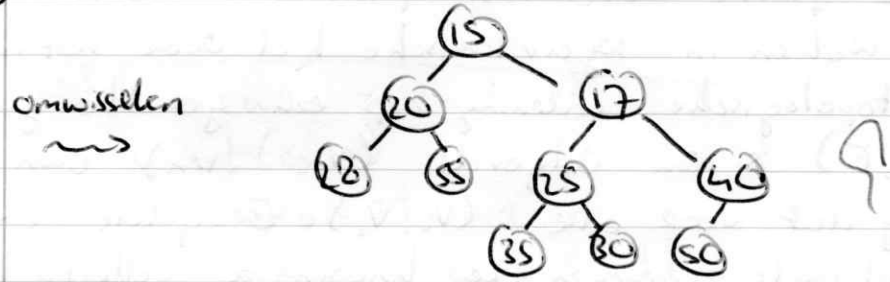
eigenlijk gezipt met een lege boom, wat gewoon dezelfde boom geeft. (klopt – zou misschien in dikraat wel toegevoegd moeten).

(ii). Minimum is de wortel. ZIP de twee subbomen samen:





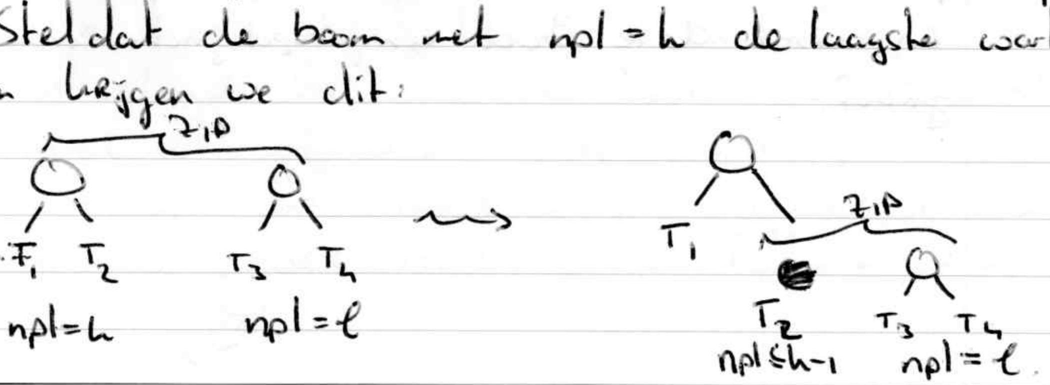
Ook hier werd (30) gezipt met een lege boom op het eind



npl is nu rechts overal kleiner. (of gelijk)

c) ~~[illegible]~~

Leftist trees hebben in iedere knoop links een minstens zo lange nil path length als rechts. ~~[illegible]~~ Stel we hebben twee bomen van nil path lengths h en ℓ. Deze zippen gaat in stappen van O(1). er hoeven alleen pointers veranderd te worden. ~~[illegible]~~ Stel dat de boom met npl = h de laagste wortel heeft dan krijgen we dit:



Op deze manier wordt er na iedere stap één van h of ℓ afgehaald. Dus het aantal zip-~~stappen~~ stappen is $O(h+\ell)$. Vervolgens herordenen kan door ~~de~~ alleen bij de ouders van de laatste zip recursief te verwissen, wat er ook $O(h+\ell)$ zijn. Samen dus complexiteit $O(h+\ell)$
Een boom met npl = h heeft minstens 2^{h-1} knopen, want de eerste h rijen van de boom moeten gevuld zijn. Dan heeft een boom van n knopen dus ook hoogstens ~~log₂(n)~~ $\log_2(n+1)$ als npl. Dit geldt voor de operatie insert, wat gedaan wordt door een boom met een knoop met de boom te zippen, een complexiteit van $O(\log_2(n+1) + \log_2(1)) = O(\log n)$. Bij DeleteMin worden dan twee bomen met ~~n~~ hoogstens n knopen samengevoegd, dus ook complexiteit $O(\log_2(n+1) + \log_2(n+1)) = O(\log n)$.
Dus de gnene standaard operaties DeleteMin en Insert werken in logaritmische tijd voor leftist trees.

(3) a) Een topologische ordening op een gerichte graaf $G = (V, E)$ is een volgorde $(v_1, \ldots, v_n)$ van V zodanig dat voor alle $(v_i, v_j) \in E$ geldt $i < j$.

b) De volgorde waarin de knopen x volledig zijn afgehandeld, is een omgekeerde topologische ordening, mits alle knopen ook bezocht worden. Er kan een stack bijgehouden worden die meegegeven wordt aan DFS. Binnen de functie na "knoop x volledig afgehandeld" wordt x op de stack gezet. Als de functie DFS voor alle knopen zonder inkomende takken wordt aangeroepen, en het markeren als "bezocht" permanent is, worden alle knopen op de stack gezet en vormt deze stack een topologische ordening op de graaf.

Course: Datastructuren Name:

Date: 13-jan-2022 Study Programme:

Teacher: Hoogeboom Student ID number:

c) Eerst de topologische ordening, deze is bijv.
(G, A, D, H, B, E, C, K, F).
We geven iedere knoop aan het begin waarde 0.
Vervolgens gaan we te opvolgende af en wordt de nieuwe waarde het maximum van waarde(i) + lengte(i,j) voor alle inkomende takken (i,j). Dan krijgen we:
G: waarde = 0 (geen inkomende takken)
A: waarde = max {0+5} = 5
D: waarde = 0 (geen inkomende takken)
H: waarde = max {0+4, 0+5} = 5
B: waarde = max {5+4, 0+8} = 9
E: waarde = max {5+2} = 7
C: waarde = max {8+1, E+4} = max {9+1, 7+4} = 11
K: waarde = max {8+3, E+4, H+4} = max {9+3, 7+4, 5+4} = 12
F: waarde = max {C+3, K+3} = max {11+3, 12+3} = 15.
Dus de projecten kunnen op zijn vroegst op tijdstip 15 afgerond zijn.

(4) a) (i) Naief heeft als worst-case complexiteit $O(m \cdot n)$, neem bijvoorbeeld het patroon aa...ab van lengte m met tekst aa...ab van lengte n. Dan moeten er m letters in het patroon worden afgegaan, en dat $n-m+1$ keer voor iedere positie in de tekst.
(ii) Het opstellen van failure links is $O(m)$, dus linear in de lengte van het patroon. Dit is omdat bij het algoritme uit (b) telkens één stap meer kan worden gedaan dan de huidige index i min het aantal stappen dat ervoor is gedaan (voor i-1), omdat deze link gevolgd wordt bij het bepalen van de link voor i. Het patroon zoeken heeft complexiteit $O(n)$, want bij

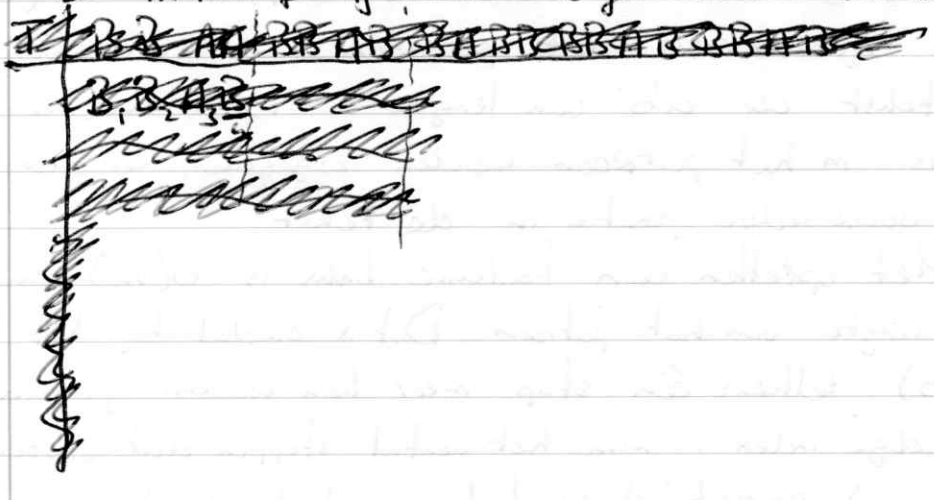
het vooruit lopen in de tekst wordt telkens één toegevoegd aan de huidige index in het patroon, wat weer ruimte geeft om (hoogstens) één keer stil te staan in de tekst en terug te bewegen in het patroon.
Dit geeft dus hoogstens 2·n check, of wel complexiteit O(n).
Omdat een patroon met hogere lengte dan de tekst geen zin heeft, komt de totale complexiteit ook uit op O(n).

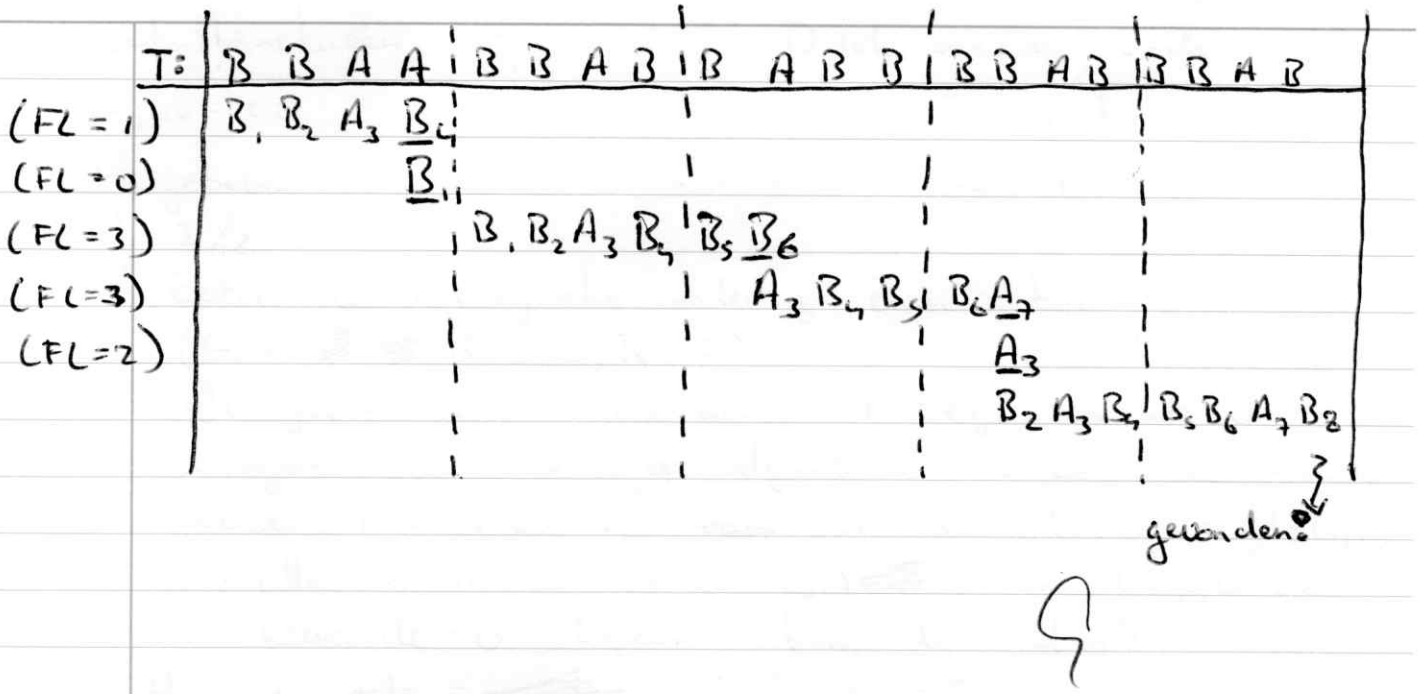
b)
```
FLink[1] := 0
FOR i FROM 2 TO n {
    Fail := FLink[i-1]
    WHILE Fail > 0 AND P[Fail] ≠ P[i-1] {
        Fail := FLink[Fail]
    }
    FLink[i] := Fail + 1
}
```

c)

index	1	2	3	4	5	6	7	8
P	B	B	A	B	B	B	A	B
FLink	0	1	2	1	2	3	3	4

d) Onderstreept als er geen match is. Bij geen match en FLink[i] = 0 schuiven we één naar rechts, anders verder met patroon zoeken door i := FLink[i] te doen. Onder de letters staan de indices in het patroon P. Letters in iedere Rij worden vergeleken met de bovenste rij.




T: B B A A B B A B B A B B B B A B B B A B
(FL = 1) B1 B2 A3 B4
(FL = 0) B1
(FL = 3) B1 B2 A3 B4 B5 B6
(FL = 3) A3 B4 B5 B6 A7
(FL = 2) A3
B2 A3 B4 B5 B6 A7 B8
gefunden!