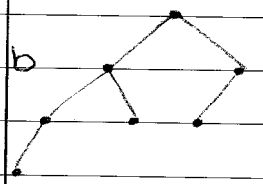
Postcode en
Woonplaats:

Studierichting:

Jaar van eerste inschrijving:

Tentamen: A2 D
Datum: 03-02-2004
Naam docent:

1a Een AVL is een binaire zoekboom waarbij voor elke interne knoop geldt dat het verschil tussen de hoogtes van zijn linker- en rechterkind hoogstens 1 is.

b 

Als een AVL-boom hoogte h heeft, heeft minstens de linker- of de rechtersubboom hoogte h-1 (de root van de originele boom wordt hierbij niet meegeteld).

Stel dat in ieder geval deze linkersubboom hoogte h-1 heeft, bij een AVL-boom betekent dit dat deze subboom tot een diepte van $h-3$ proper is en dus minstens 2^{h-3} knopen heeft.

Om zo'n zelfde reden heeft de rechtersubboom minstens 2^{h-4} knopen. (De hoogte is hier immers h-1 of h-2)

$$n \geq 2^{h-3} + 2^{h-4} \geq 2^{h-3}$$

$$\log n \geq h-3 \Rightarrow h \leq \log n \Rightarrow h = O(\log n)$$

Het bewijs voor het geval dat de rechtersubboom hoogte h-1 heeft en de linkersubboom niet gaat analoog hieraan.

Dus $h = O(\log n)$.

2 a Een ~~minimum~~ spanning tree is ~~de~~ een (deel-) graaf die alle punten bevat, de graaf is samenhangend en bevat geen cykels.

Een minimum spanning tree een spanning tree met ~~de~~ het laagst mogelijke totale gewicht.

b Algorithm MST(V, E)

Input: verzameling V van vertices van een graaf G, verzameling E van gewogen edges van G

Output: een minimum spanning tree van G

```
Stop de elementen van E in een priorityqueue Q
  met de gewichten in niet-dalende volgorde.
for all u ∈ V do
    maak cluster C_u
    stop u in C_u
while not Q.isEmpty() do
    e ← Q.removeMin();
    u en v eindpunten van e
    if C_u ≠ C_v then
        verenig C_u en C_v
        voeg e toe in vereniging van C_u en C_v
return een cluster C_u
```

Een cluster bevat hier zowel knopen als kanten. Ongebruikelijk, maar het kan.

Postcode en Woonplaats: | Jaar van eerste inschrijving: | Datum: Naam docent:

3a Algorithm MaxC(N)

Input : flow network N met source s, sink t, een capaciteitsfunctie c

Output : ~~pad~~ van ~~t naar t~~ maximale capaciteit

```
for all u ∈ N punten do
    D[u] ← -∞
i ← 0, D[s] ← ∞
// stop s in Li
while Li niet leeg do
    for all u ∈ Li do
        for all edges e incident to u do
            v endpoint of e ≠ u
            D[v] ← max(min(D[u], c(e)), D[v])
            if v ≠ t then
                stop v in Li+1
    i ← i+1
return D[t]
```

voor het gevraagde pad moet het algoritme iets aangepast worden

b - eerste for-lus is $O(n)$

- while-lus: ~~maximaal n keer uitgevoerd~~

elke edge wordt precies één keer bekeken $\Rightarrow O(m)$

$\Rightarrow O(n) + O(m) = O(n+m)$

daarnaast kost het ongeveer $O(\log n)$ tijd om de edges te vinden die incident zijn met een knoop

$\Rightarrow O((n+m)\log n)$

Postcode en Woonplaats:	Jaar van eerste inschrijving:	Datum: Naam docent:

4a Algorithm f(n, b)

Input: n de graad van het polynoom, b de ~~coëfficiënten~~ $[z_0, \dots, z_{n-1}]$

Output: de coëfficiënt van het unieke polynoom p(x)

if n = 0 then

return [1]

if n = 1 then

$FFT([(a - x_0)^{-1}, 0, 0, 0], [-z_0, 1, 0, 0])$

return

if n > 1 then

~~return [...]~~

y ← ~~FFT([...])~~ $f(n-1, [z_1, \dots, z_{n-1}])$

return [1, ...]

↑ elementen uit y

algoritme mbv. ~~$\sum_{i=0} a_i x^i = a_0 + \sum_{i=1} a_i x^i = a_0 + a_1 x + \sum_{i=2} a_i x^i = a_0 + x(a_1 + \sum_{i=2}$~~

$a_0 + a_1 x + a_2 x^2 + \dots$

$=$

$a_0 + x(a_1 + a_2 x + \dots)$

b FFT kost $\mathcal{O}(n \log n)$

$$T(n) = \begin{cases} c & n < 2 \\ T(n-1) + n\log n & n \geq 2 \end{cases}$$

~~als $T(n) = \begin{cases} c & n < 2 \\ T(n-1) + \log n & n \geq 2 \end{cases}$~~

~~dan~~ $n \log n = \Theta(n^{\log 1} \log^1 n) \Rightarrow T(n) = \Theta(n \log^2 n)$