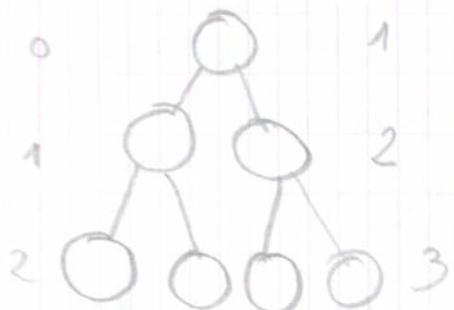


Heap

- complete binaire boom + heapvoorraadde
 - ↳ alle niveau's gevuld, behalve het laatste, maar dan liggen alle knopen helemaal links (zonder - er is dus hoogstens 1 knoop met 1 kind. gaten)

- door de regelmatige vorm van de heap is er een eenvoudig verband tussen n & ~~de hoogte~~ de hoogte.
 - op $i \Rightarrow 2^i$ knopen
 - een volledig opgevulde boom heeft dan

$$\sum_{i=0}^h 2^i = 1 + 2 + 4 + \dots + 2^{h-1} + 2^h = 2^{h+1} - 1$$



$$\begin{aligned}
 \sum_{i=0}^h 2^i &= 1 + 2 + 4 + \dots + 2^{h-1} + 2^h \\
 &= 1 + 2 + 4 = 7 \\
 &= 2^3 - 1 = 7
 \end{aligned}
 \quad \left| \begin{array}{l}
 d = 1 + 2 + \dots + 2^{h-1} + 2^h \\
 -2d = -2 + 4 - \dots - 2^{h-1} + 2^h \\
 \hline
 d = 1 + 2^{h+1} - 1 \\
 = 2^{h+1} - 1
 \end{array} \right.$$

- op een laagste niveau niet noodzakelijk gevuld, min 1 knoop/blad op het laagste niveau

$$\sum_{i=0}^{h-1} 2^i + 1 \leq 1 + 2 + 4 + \dots + 2^{h-2} + 2^{h-1} + 1 = \underbrace{2^h - 1}_{2^h} + 1$$

laatste niv. maar
1 knoop

m.a.w. $2^h \leq n \leq 2^{h+1} - 1$

$$2^h \leq n \leq 2^{h+1} \Rightarrow \lfloor \lg n \rfloor = h$$

- deze binaire structuur is ideaal om opgeslagen te worden in een tabel.

- ↳ - $2i \& 2i+1$ zijn de kinderen van knoop i
- $i/2$ is de ouder van de kinderen $2i \& 2i+1$

- de heapvoorraad: min of max-heap

- bewerkingen op heaps: - element toevoegen $O(\lg n)$

- wortel-element verwangen $O(\lg n)$
- wortel-element vervangen $O(\lg n)$
- element vervangen $O(\lg n)$

Constructie van heap

a) door toevoegen

b) door samenvoegen van deelheaps

a) door toevoegen

- eerste element is ok
- vervolgens $(n-1)$ overige elementen toevoegen
- stel alle niveaus vol (bovengrens)

$$T(n) \leq 2 \cdot 1 + 4 \cdot 2 + 8 \cdot 3 + \dots + 2^h \cdot h$$

- $2T(n) \leq 4 \cdot 1 + 8 \cdot 2 + 16 \cdot 3 + \dots + 2^{h+1} \cdot (h+1)$

$$T(n) \leq -2 - 4 - 8 - \dots - 2^h + 2^{h+1} \cdot h$$

$$T(n) \leq \underbrace{1 - (1 + 2 + \dots + 2^h)}_{2^{h+1}-1} + 2^{h+1} \cdot h$$

$$\leq 1 - 2^{h+1} + 1 + 2^{h+1} \cdot h$$

$$\leq 2 + 2^{h+1}(h-1)$$

$$\leq 2 + 2n \lg n - n$$

(aangezien $2^h \leq n < 2^{h+1}$)

$\Rightarrow O(n \lg n)$ in het slechteste geval

b) Heap opbouwen door samenvoegen van deelheaps

- opbouwen van onder naar boven
- een knoop, die 2 deelheaps onder zich heeft, tot wortel maken van de nieuwe heap
- conflicten hogerop worden genegeerd
- één enkele knoop is steeds een heap (starten op voorlaatste niveau, bij de knopen met kinderen).
- volgorde waarin deelheaps worden uitgebreid is onbelangrijk maar het is een voudig van rechts naar links starten bij $\frac{n}{2}$
- op het eerste zicht lijkt de constructie $O(n \lg n)$
- maar het werk wordt begrensd door de hoogte v/d deelheaps & dat gebeurd $\frac{h}{2}$ keer
 \downarrow
 $O(\lg n)$
- opnieuw een bovenlimiet, max werk

$$T(n) \leq 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 4(h-2) + 2(h-1) + 1 \cdot h$$

$$- 2T(n) \leq 2^h + 2^{h-1} \cdot 2 + 8(h-2) + 4(h-1) + 2h$$

$$T(n) \leq 2^h + 2^{h-1} + \dots + \cancel{4(h-2) + 2(h-1)} + 4 + 2 - h$$

$$\leq \underbrace{-1 + (1 + 2 + \dots + 2^{h-1} + 2^h)}_{2^{h+1}-1} - h$$

$$\leq 2^{h+1} - 2 - h$$

$$\leq 2n - 2 - \lfloor \lg n \rfloor$$

$$T(n) = O(n)$$

- heapsort

efficiënte constructie $\Rightarrow O(n)$

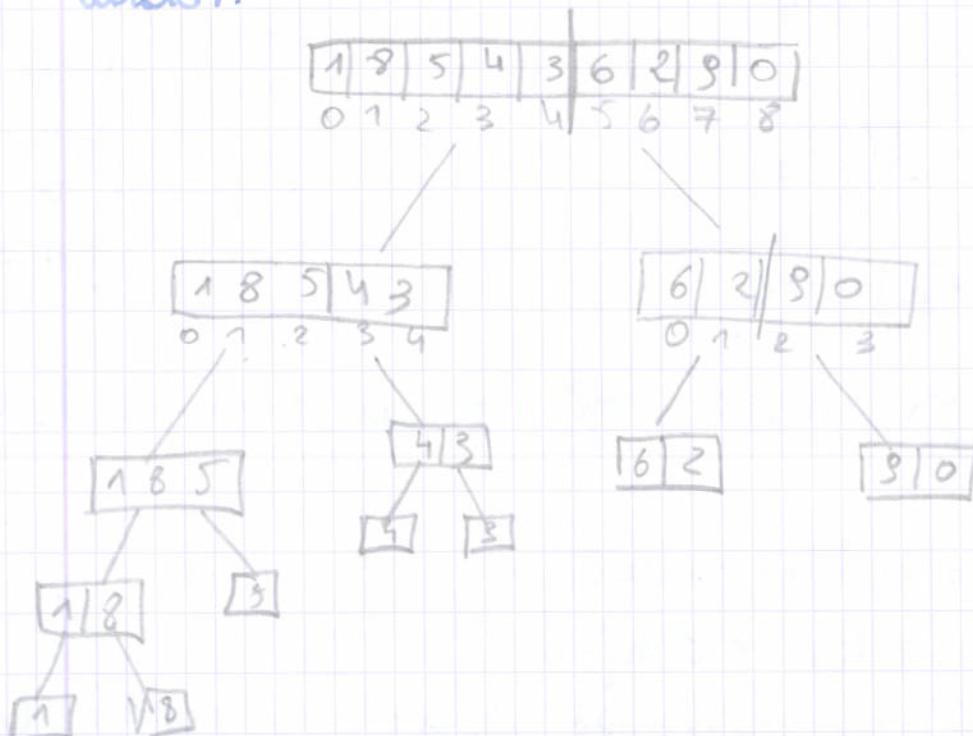
$n-1$ keer wortel verwijderen $O((n-i) \lg n)$ } $O(n \lg n)$

Rangschikken door (inwendig) samenvoegen

- verdeel & heers, opdelen in deelprobleem
- onafhankelijke problemen, uiteindelijk is het deelprobleem zo triviaal dat het (eenvoudig) opgelost kan worden.
- heersen: combineert ~~het~~ de deelproblemen tot de oplossing van een groter probleem.

D Merge sort

- een tabel met 1 element is gesorteerd, en wordt dus opgedeeld tot er slechts 1 element in die is.
- samenvoegen van 2 gesorteerde tabellen, gebeurd zeer eenvoudig (telkens de kleinste elementen vergelijken)
- wanneer het einde v. 1 van de tabellen bereikt wordt kan eenvoudig de overige helft v. de andere tabel gekopieerd worden.



} hulptabel
- kleinste deel-tabel moet gekopieerd worden.
(slechts 1 hulptabel nodig)?

- kan statisch maken: door steeds eerst links toe te voegen

Performantie

- eenvoud, veronderstellen $n = 2^k$, dan kan de tabel telkens in perfect zedelen worden mogelijk.

$$T(n) = 2T(n/2) + Cn$$

aantal operaties om samen te voegen

- om de recursie (recurrentie vgl.) op te lossen, delen we eerst door n

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + C$$

- passen toe op elke halve deeltabel

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + C$$

:

$$\frac{T(4)}{4} = \frac{T(2)}{2} + C$$

$$\frac{T(2)}{2} = T(1) + C$$

} k , kan opdelen
 $n = 2^k$

\downarrow

$\lg n = k$

- we tellen nu alle vgl'en op, zodat de meeste termen wegvalt

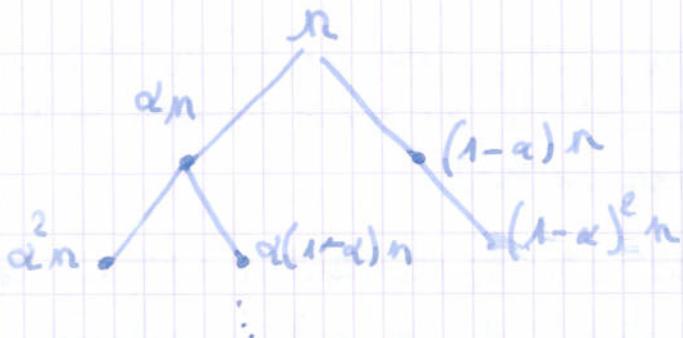
$$T(n) = nT(1) + Cn \cdot k$$

$$= nT(1) + Cn\lg n$$

Omdat $T(1)$ een constante is, is de performantie $O(n\lg n)$

- zelfde performantie komt men ook al uit n , geen macht van 2.

- Deze performance wordt gehaald omdat de tabel telkens in 2 delen wordt opgedeeld en omdat het samenvoegen van de delen in lineaire tijd gaat.
 - Maar wat als de tabel niet in 2 gelijke delen wordt gesplitst? Men kan aantonen dat een constante grootte verhouding tussen de delen voldoet.
 - bv. het grootste deel is steeds een fractie α met $0,5 \leq \alpha < 1$. Na k verdeleningen is de grootte van het grootste deel nog slechts $n\alpha^k$ bij $\alpha^k = \frac{1}{2}$.
 - We stoppen met onderverdelen als er slechts nog 1 element over is, want dat is het triviale geval.
- m.a.w. $n\alpha^k = 1 \Leftrightarrow \alpha^k = \frac{1}{n} \quad \lg \frac{1}{n} = \overbrace{\lg 1 - \lg n}^{\uparrow}$
- $$k \lg \alpha = \lg \frac{1}{n}$$
- $$k = \frac{\lg n}{\lg \alpha} \Leftrightarrow k = O(\lg n)$$



- elementen op elke niveau, want het aantal elementen kleiner dan n .
- hoogte boom $O(\lg n)$, ensamenvoegen $O(n)$

$$\Downarrow$$

dus $O(n \lg n)$ & ook $\Omega(n \lg n)$

- iets sneller dan heapsort met een goede implementatie $\Theta(n \lg n)$

- implementaties

- o het oproepen v/d recursieve procedures zou voor enige vertraging kunnen zorgen, zodat het soms gunstig zal zijn om de recursieve oproep te vermijden. Met een goede compiler is het \neq miniem.
- o afwisselend verplaatsen, opv teltens kopiëren nr hulptabel
 - a) insertion-sort, wanneer de deeltabellen vrij blijven zijn (efficiënt & ook stabiel)
 - b) we kunnen ook direct onderaan starten (bottom-up)
 - alle paren geen volgende elementen samen voegen, tot deeltabellen met lengte 2., vervolgens de 2 deeltabellen ramen voegen tot een deeltabel met lengte 4.
 - deeltabellen zijn \neq & ramenvoegen ook \neq als bij de top-down methode
- o - ook hier dezelfde verbeteringen, door 2 even grote tabellen \times afwisselend kopiëren \times beginnen met 2 kleine deeltabellen, na inzien sort
 \rightarrow ook $\Theta(n \lg n)$
- een belangrijke eigenschap van de beide versies is dat de elementen in sequentiële volgorde worden behandeld.
(Gundrig v/d processor caches)
- Negeert ook ideaal als de elementen niet in tabellen, maar in gehechte lijsten of sequentiële bestanden zijn opgeslagen
- voordeel: stabiel
- nadeel: niet ter plaatse

Rangsleiken door onderverdelen

- verdeel-en-heus methode : twee deeltabellen, afhangelijk gerangscher. met dezelfde methode.
- Mergesort: onderverdelen makkelijk, samenvoegen moeilijk
Quicksort: onderverdelen meeste werk, samenvoegen onmiddellijk
- Een pivot verdeelt de tabel in twee (een uniek element)
 - o linkse deel, elementen kleiner dan of gelijk aan de pivot.
 - o rechtse deel, elementen groter of gelijk aan de pivot.
- ? Hoe groot zijn de deeltabellen? Is op voorhand niet geweten.
- Verwisselen van de elementen, zowel linker als rechter zijde tot ze elkaar ontmoeten, hierna is de tabel in twee deeltabellen onderverdeeld.
- Opmerkelijk is maar 1 voorwaarde. (dit lukt door de eerste swap)
- de burende herhalingen zijn zo eenvoudig, waardoor het algoritme zeer snel is.
- tabel wordt gesplitst door de index, j
- ook rekening houden met gelijke element (oneven wichtige partities)

Performantie

- o beste geval: dan zijn beide delen even groot (= meestal)

$$T(n) = 2T(n/2) + cn$$

$$T(n) = O(n \lg n)$$

- zeer optimistisch & onrealistisch

- o slechtste geval: één van de twee delen bevat slechts 1 element, en als dat op elke niveau gebaard krijgen we $O(n^2)$

$$T(n) = cn + T(1) + T(n-1)$$

en

$$T(n-1) = ((n-1) + T(1) + T(n-2))$$

$$T(n-2) = (n-2) + T(1) + T(n-3)$$

$$+ \quad T(2) = c2 + T(1) + T(1)$$

$$T(n) = \underline{\underline{n^2}} + nT(1)$$

in het slechtste geval dus $O(n^2)$

- o gemiddelde geval: de waarschijnlijkheid dat een willekeurig element, de pivot is, is gelijk (= een waarschijnlijkheid met als gevolg de grootte van de deeltabellen is even, waarschijnlijk).

de gemiddelde uitvoeringsijd:

$$\frac{1}{n} \sum_{i=1}^n T(n,i) = T(n) = cn + \frac{1}{n} \sum_{i=1}^n T(i-1) + T(n-i)$$

\uparrow
werk per
partitie

$\rightarrow i \text{ is de pivot } \Leftrightarrow i \text{ is een waarschijnlijkheid}$

$\frac{1}{n}$

$\overbrace{\qquad\qquad\qquad}^{\text{dubbele elementen}}$

$T(2) + T(n-2)$

$T(n-2) + T(2)$

alle vorige T's

$$T(n) = cn + \frac{2}{n} \sum_{i=0}^{n-1} T(i)$$

- de som maakt het moeilijk, kunnen we een volledig vermijden door de "full-history" bewerking.

Full-h $\left\{ \begin{array}{l} x \cdot n \\ (\text{sub}) \quad n \rightarrow n-1 \\ - 2 \text{ delen voor } \alpha \text{ wa} \end{array} \right.$

$$nT(n) = cn^2 + 2 \sum_{i=0}^{n-1} T(i)$$

$$\underline{- (n-1)T(n-1) = c(n-1)^2 + 2 \sum_{i=0}^{n-2} T(i)}$$

$$\begin{aligned} nT(n) - (n-1)T(n-1) &= cn^2 - c(n-1)^2 + 2T(n-1) \\ &= cn^2 - cn^2 + 2cn - c + 2T(n-1) \end{aligned}$$

$$nT(n) - (n-1)T(n-1) = c(2n-1) + 2T(n-1)$$

$$\begin{aligned} nT(n) &= (n-1)T(n-1) + 2T(n-1) + c(2n-1) \\ &= (n-1+2)T(n-1) + 2cn \quad \cancel{+ c(2n-1)} \text{ (vermaakbare)} \end{aligned}$$

$$nT(n) = (n+1)T(n-1) + 2cn$$

~~reductie~~ $\left\{ \text{delen door } n(n+1) \right.$

$$\frac{T(n)}{(n+1)} = \frac{\cancel{T(n-1)}}{n} + \frac{2c}{(n+1)}$$

$$\frac{\cancel{T(n-1)}}{(n)} = \frac{\cancel{T(n-2)}}{n-1} + \frac{2c}{n}$$

alles optellen

$$\frac{\cancel{T(1)}}{2} = \frac{T(1)}{2} + \frac{2c}{3}$$

$$\Rightarrow \frac{T(n)}{(n+1)} = \frac{T(1)}{2} + 2c \sum_{i=2}^n \frac{1}{i+1} < 2c \sum_{i=1}^n \frac{1}{i} \rightarrow n^{\text{e}} \text{ harmonische getal}$$

We weten v/h n^{e} harmonische getal H_n , dat deze als waarde $\Theta(n + O(1))$ heeft

maar: $T(n) = (n+1) \ln n$

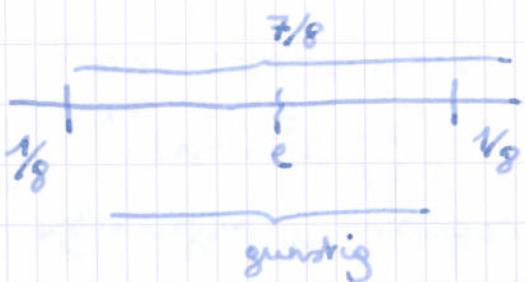
gemiddeld dus de performance net als het beste geval $O(n \lg n)$

Wanneer is elke grootte v/d deeltabel een waarschijnlijk of v/d pivot met dezelfde waarschijnlijkheid elk moment?

- elke permutatie v/d invoertabel een waarschijnlijk
 - ↳ elk element met dezelfde waarschijnlijkheid op elke plaats
 - o bekomen door de invoertabel, waaraf random te permuteren met een random generator
(ook voor de deeltabellen)
- eenvoudiger, random plaats bepalen met een random generator.

Random pivot (Randomized Quicksort)

- Een random element als pivot kiezen, met een grote waarschijnlijkheid $O(n \lg n)$, onafhankelijk v/d invoervolgorde
- aan tonen met een partitieboom (zie merge sort), op elk niveau $O(n)$ gegevens, werk om te partitioneren $O(n)$. Nu zal blijken dat de hoogte v/d boom met een grote waarschijnlijkheid $O(\log n)$ is.
- de hoogte v/d boom wordt bepaald door de langste reeks opeenvolgende partities. Trachten aan te tonen dat de waarschijnlijkheid dat een element "e" betrokken is bij meer dan $O(\log n)$ partities zeer klein is. $O(n^{-6})$



de kans dat de deeltabellen een grootteverhouding $\frac{7}{8}$ hebben is kleiner dan of gelijk aan $\frac{1}{4}$

$$\left(\frac{1}{8} + \frac{1}{8} \right) = \frac{2}{8} = \frac{1}{4}$$

- als de verhouding kleiner is dan $\frac{7}{8}$ dan noemen we de partitie **gunstig** voor e.
- nu na k gunstige partities is de grootte v/k partitiedelen dat e bevat zeker kleiner dan $n\left(\frac{7}{8}\right)^k$
- aangezien de partitie stoppt bij 1 element, kan e maximaal $n\left(\frac{7}{8}\right)^k = 1$ gunstige partities meenemen.

~~$E[k \log n \cdot \frac{7}{8}] = 0$~~

$$\frac{7^k}{8} = \frac{1}{n}$$

~~$E[k \log n + \log \frac{7}{8}] = 0$~~

$$E[\log \frac{7}{8}] = -\log \frac{7}{8}$$

~~$E[k \log n + k \log \frac{7}{8}]$~~

$$k = \frac{\log n}{-\log(\frac{7}{8})} = \frac{\log n}{\log \frac{8}{7}} \approx 1 \quad \left\{ \begin{array}{l} \log n \\ \log \frac{8}{7} \end{array} \right.$$

\hookrightarrow basis $\frac{8}{7}$, (vs willekeurig)

- We gaan nu de kans bepalen dat er tot 20 log n partities behoort waarvan er meer dan 18 log n ongunstig zijn.
- Een partitie met random pivot, kan beschouwd worden als een toevals experiment met 2 uitkomsten: gunstig of ongunstig.
- De opvolgende partities, met telkens een random pivot, vormen een reeks onafhankelijk toevalsexperimenten
- het aantal ongunstige partities, in een reeks, is een binomiale waarschijnlijkheidsverdeling.



- kans a = p

- kans b = 1-p

Wat is de kans kxa?

$$\binom{n}{k} p^k (1-p)^{n-k}$$

- dus de kans dat er hoogstens j ongunstige partities zijn op 20 log n is dan

$$\sum_{j=0}^{20 \log n} \binom{20 \log n}{j} \left(\frac{1}{4}\right)^j \left(\frac{3}{4}\right)^{20 \log n - j}$$

- de kans dat er dan meer dan 18 log n ongunstige partities zijn is dan

$$\sum_{j>18 \log n} \binom{20 \log n}{j} \left(\frac{1}{4}\right)^j \left(\frac{3}{4}\right)^{20 \log n - j}$$

- we zoeken een bovengrond door vereenvoudigen / groter maken

$\frac{3}{4} \Rightarrow 1$ & ook de combinatie vereenvoudigen met stirling

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdots (n-k+1)}{k!} < \frac{n^k}{k!} < \left(\frac{ne}{k}\right)^k$$

$$\text{stirling: } k! = \left(\frac{k}{e}\right)^k$$

- hieruit volgt:

zoeken

$$\sum_{j > 18 \log n} \left(\frac{5 \log n}{j} \right)^j \left(\frac{1}{18} \right)^j$$

sommative wordt groter,
dus wordt de sommer
groter \Rightarrow resultaat blijft
wali nooit kleiner
groter resultaat
is bovenlimiet

- als we j vervangen door $18 \log n$ wordt het resultaat nog groter

zoeken

$$\sum_{j > 18 \log n} \left(\frac{5 \log n}{18 \log n} \right)^j = \sum_{j > 18 \log n} \left(\frac{5e}{18} \right)^j$$

\rightarrow meetkundige reeks

\rightarrow (van afgesloten)

$$\times \frac{5e}{18}$$

en -

$$d = \left(\frac{5e}{18} \right)^{18 \log n} + \dots + \left(\frac{5e}{18} \right)^{20 \log n}$$

$$-\frac{5e}{18} d = \left(\frac{5e}{18} \right)^{18 \log n+1} + \dots + \left(\frac{5e}{18} \right)^{20 \log n+1}$$

$$\left(\frac{5e}{18} - 1 \right) d = - \left(\frac{5e}{18} \right)^{18 \log n} + \left(\frac{5e}{18} \right)^{20 \log n+1}$$

maar $0 < \frac{5e}{18} < 1$

$$d = \frac{\left(\frac{5e}{18} \right)^{18 \log n}}{1 - \left(\frac{5e}{18} \right)}$$

- De hoogte v/d boom wordt bepaald door het element dat het grootste aantal partities meemaakt. We hebben dus de kans nodig dat een v/d n te rangschikken elementen meer dan $18 \log n$ verschillende partities meemaakt.

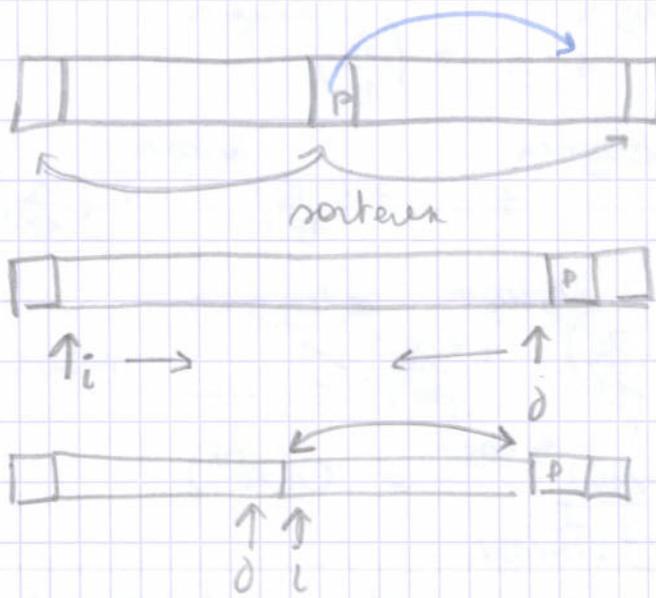
de som van

- De unie van gebeurtenissen niet groter dan \checkmark hun waarschijnlijkheden $\sum_{i=1}^n \frac{1}{i} = O(n^{-1}) = O(n^{-6})$

- Dus is de kans dat Randomized quicksort $O(n \log n)$ bedraagt $= 1 - O(n^{-6})$

Ø een andere toepassing de mediaan van drie

- een streekgroef, willekeurige elementen de mediaan te bepalen
- linsie, middelste \Rightarrow redusie
- kans streekgroef : $\approx 1/3$ gelijk aan min of max en dat op ell niveau $\Rightarrow O(n^2)$ en die kans is zeer klein.
- de andere elementen als standaardwaarden, daarom rangschikt men deze ter plaatse
- spil element tijdelijk oplossen (vooraanstaande) element zodat we het erin op de juiste plaats kunnen terug zetten.
- de volgende partities bevratten de pivot niet meer.
- performantie is ongewijzigd ze lijkt 5% beter!



- implementaties

- deelabel : insertionsort

- recursie stoppen bij kleine deeltabellen + op einde insertion sort

Ø keiplaats

Ø een shaker : gemiddeld $\Omega(\lg n)$ slechtst $O(n)$

↳ stapelgrootte beperkt worden door de staaktreursie te verwijderen $\Rightarrow O(\lg n)$

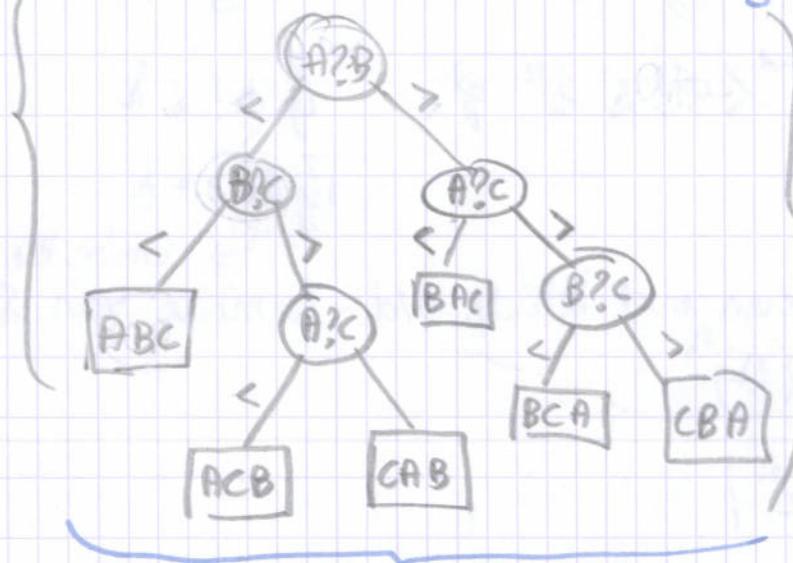
Ø niet stabiel

Ondergrens voor de efficiëntie van rangschikken.

$$\sim O(n^2)$$

- de efficiënte methoden (behalve quicksort) voeren $O(n \lg n)$ operaties uit om n gegevens te rangschikken
- snellere methoden dan $O(n \lg n)$? → neen
- aantallen, zonder rekening te houden met de ≠ uitkijken. Maar wel met een gemeenschappelijke eigenschap: sleutels vullen
- (nog andere factoren, maar worden buiten beschouwing gelaten)
- men kan al de vullen & mogelijke resultaten (permuaties) voorstellen door een full-binary tree (elke o of 2 kinderen).
- de enigedige knoop stelt een sleuteltyp voor & een eind-knoop een willekeurige volgorde (beslisningsboom)
- je kan voor elk sorteer algoritmen een beslisningsboom opstellen. b.v. voor insertion sort $[A \ B \ C]$
- we veronderstellen ook dat een sleutelvulg $O(1)$ is (niet echt realistisch)

minimale hoogte



hoe sleutels verplaat worden speelt geen rol

$n!$ permuaties (bladeren)

- aantallen rangschikken: $\Omega(n! \lg n)$ zowel gemiddeld als in het slechste geval.

Slechtste geval: de langste mogelijke weg v/d wortel naar een blad. \Rightarrow de hoogte v/d boom.

- we willen een ondergrens dus de minimale hoogte! van een volle bin. boom met $n!$ blaaderen.

Een volle bin. boom met hoogte h , niet meer dan 2^h blaaderen
inductie: $\rightarrow h=0 \quad 2^0 = 1$ blad, ok.

$\rightarrow h>0$ veronderstellen we dat ook geldt voor elke kleinere hoogte.

een boom met hoogte h , heeft 2 deelbomen die niet hoger zijn dan $h-1$, zodat ze elk max 2^{h-1} blaaderen bevatten

$$\hookrightarrow h = 2 \cdot 2^{h-1} = 2^h$$

- nu kunnen we deze eigenschap gebruiken: we weten echter dat een ~~is~~ beslissingsboom, die n sleutels rangschikt, $n!$ blaaderen heeft \rightarrow wat is de hoogte?

$$2^{h-1} < n! \leq 2^h \text{ of } \lg n! \leq h$$

$$\lceil \lg n! \rceil = h$$

\rightarrow ambetant

$n!$ benaderen met behulp v/d formule van Stirling

$$\hookrightarrow n! = \left(\frac{n}{e}\right)^n$$

$$h = \lceil \lg \frac{n^n}{e} \rceil$$

$$\Rightarrow n \lg n - n \lg e$$

$$h \geq \Omega(n \lg n)$$

(informatietheoretische ondergrens)

$$\begin{aligned} \text{bij} h \\ n = 2^h \\ \lg n = h \end{aligned}$$

Gemiddeld geval: de gemiddelde weglength v/d wortel naar een blad, in een volle bin. boom met $n!$ bladeren.
 → ondergrens: de minimale gemiddelde weglength

Veronderstelling: elke permutatie v/d (verschillende) sleutels is even waarschijnlijk. De verwachtingswaarde van die weglength wordt dan het rekenkundig gemiddelde van alle mogelijke wegen.

De som dan alle wegen (wortel tot een blad) \Rightarrow uitwendige weglength.

De uitwendige weglength van een volle bin. boom is minstens $b \lg b$ met b het aantal bladeren

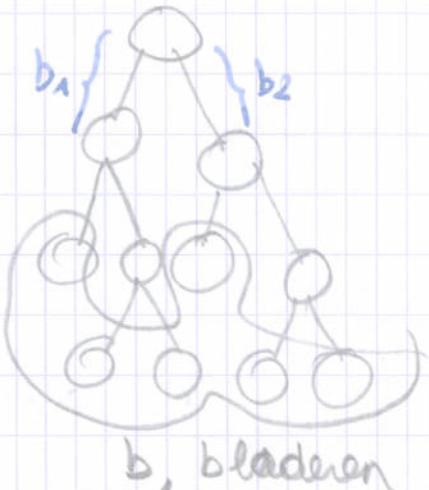
→ aan tonen via induktie

- $h=0$, de wortel enigste (1 blad)

$$= 1 \cdot \lg 1 = 0$$

$$2^0 = 1$$

$$0 = \lg 1$$



b , bladeren

- $h > 0$, de voor alle kleinere hoogten dan h .

- wortel 2 delbomen met b_1 & b_2 bladeren

$$\hookrightarrow \text{eigenschap } b \lg b = b_1 \lg b_1 + b_2 \lg b_2 + b_1 + b_2$$

- nu met $b = b_1 + b_2$ wordt de kleinste mogelijke waarde behaald met $b_1 = b_2 = \frac{b}{2}$

$$b \lg b = \underbrace{\frac{b}{2} \lg \frac{b}{2}}_b + \underbrace{\frac{b}{2} \lg \frac{b}{2}}_b + \underbrace{\frac{b}{2} + \frac{b}{2}}_b$$

$$= b \lg \frac{b}{2} + b$$

$$= b \left(\lg \frac{b}{2} + 1 \right)$$

$$= b \left(\lg \frac{b}{2} + \lg 2 \right)$$

$$= b \left(\lg \frac{b \cdot 2}{2} \right) = b \lg b$$

de gemiddelde weglength met $n!$ bladeren is dus $\xrightarrow{\text{rekenkundig}} \text{gemiddelde}$

$$\frac{1}{n!} n! \lg n!$$

$$= \Omega(n \lg n)$$

Bucket sort (stabiel \Rightarrow insertion sort, niet te plasseren)

- reële getallen
- uniforme waarschijnlijkheidsverdeling
- beperkt interval
- interval opgedeeld in m gelijke buckets.
- nagaan tot welke bucket een sleutel behoort $\Rightarrow O(1)$
- sleutels verdeelt over de m buckets, door de waarschijnlijkheidsverdeling, gemiddeld evenveel sleutels
- door m voldoende groot te kiezen ($m = O(n)$), zorgt men ervoor dat elk deelinterval gemiddeld slechts een klein aantal sleutels, onafhankelijk van n .
- elk deelinterval met insertion sort.
- aantal elementen in elke bucket is onbekend, dus voorstellen door een gelinkte lijst of dynamische tabel.

Performantie

- $O(n)$ deelintervallen visualiseren
- sleutels over deelintervallen verdelen
- dan de intervallen afzonderlijk rangschikken
- terugplaatsen in de juiste volgorde \rightarrow oorspronkelijke tabel

$$T(n) = \Theta(n) + \sum_{i=1}^m O(n_i)^2$$

Slechtstegeval: alle ~~sleutels~~ sleutels in 1 bucket $\Rightarrow O(n^2)$

- aangezien de kans p , dat een sleutel in een deelinterval valt gelijk is aan $1/m = O(1/n)$, en onafhankelijk is van waar de andere sleutels terecht komen is die kans $O(1/n^2) \Rightarrow$ bijzonder klein!

- maar ook in minder slechte gevallen kan de methode slecht performen, als ~~alle~~ enkele vd deelintervallen te groot worden

Gemiddeld geval: $\Theta(n)$

→ de waarde die gemiddeld zal worden aangenomen
de gemiddelde waarde van $T(n)$ is:
verwachtingswaarde

$$E[T(n)] = \Theta(n) + \sum_{i=1}^m E[\alpha(n_i)]$$

- lineairiteit: boven grens bevat een constant, dus kunnen we deze uitschrijven.

$$E\left[\sum_{i=1}^m c_i x_i\right] = \sum_{i=1}^m c_i E[x_i]$$

$$E[cX] = c E[X]$$

waarmee volgt

$$E[T(n)] = \Theta(n) + \sum_{i=1}^m \Theta(E[n_i^2]) \quad \left| \begin{array}{l} = \Theta(n) + \sum_{i=1}^m \Theta(1/n_i) = \Theta(n) \\ (\text{voldoende, sleutel in } O(1) \\ \text{en rangschikken van alle} \\ \text{deelintervallen gemiddeld}) \end{array} \right.$$

→ of een sleutel in deelinterval i terechtkomt of niet, is een toevalsexperiment met 2 resultaten. De experimenten zijn onafhankelijk, en worden n keer herhaald.

→ n_i is dus een toevalsvariable, het aantal keer dat het experiment juistig was, heeft dan een binomiale waarschijnlijkhedsverdeling.

$$\pi = 1/m$$



de kans $a = p$

$R \times a?$

$$b = 1-p$$

$$P(X=n_i=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

gekend $\left\{ \begin{array}{l} E[n_i] = n \cdot p \\ V[n_i] = n \cdot p \cdot (1-p) \end{array} \right.$

$$E[n_i^2] = \boxed{\square} \rightarrow \text{afgeleid worden uit } V[n_i]$$

$$V[X] = E[X^2] - E[X]^2$$

$$\stackrel{\text{↑}}{E[n_i^2]} = V[n_i] + E[n_i]^2$$

met $p = \Theta(1/n)$

$$E[n_i^2] = np(1-p) + np^2 = np(1-p + p) = (1 - \frac{1}{n} + 1) = \Theta(1 - \frac{1}{n})$$

De selectie operatie

- belangrijke toepassing veruant met rangschikken, is het zoeken naar de mediaan.
- belangrijk in de statistiek
- mediaan zoeken is een speciaal geval v/d k^e kleinste
- geen enkel algoritme kan garanderen dat een element de k^e kleinste is zonder de $k-1$ kleinere of $n-k$ grotere.
↳ de k^e kleinste elementen (niet noodzakelijk in volgorde)

Aantal mogelijkheden, afhankelijk v/d waarde van k

1) zeer kleine k : - k maal het kleinste element

of

zeer grote k : - k maal grootste element

- $k=1$ of $k=n$ ($n-1$) sleutel vullen

- tegelijkertijd mind max: $\lceil \frac{3n}{2} \rceil - 2$, paren nemen
of vullen
met min & max

- 2 grootste elementen: $n-2 + \lceil \lg n \rceil$, collectie kandidaten
bij te houden voor 2^e grootste

2) als wat grotere k

a) k keer de eerste stappen van heapsort (dalingende heap)

$O(n + k \lg n)$

b) een stijgende heap bij houden met de voorlopige k kleinsten. Geinitialiseerd met de eerste k elementen vervolgens volgende elementen ($n-k$). Vullen met wortel en als kleiner wortel vervangen

$O(k + (n-k) \lg k)$

3) voor nog grotere k : een methode gebaseerd op de partitie van quicksort.

- de partitie verdeelt de tabel in elementen kleiner dan de pivot, en elementen groter dan de pivot

Om het k^{e} blauwe te vinden:

↳ implementatie die de pivot op positie i plaatst

als $k = i$ gevonden

als $k < i$ dan zoeken in linker deel

als $k > i$ dan zoeken in rechter deel naast $(k-i)^{\text{e}}$ element.

- zoeken in deeltabellen gebundeld recursief (kan bewijderd worden).

- net zoals bij quicksort hangt de performance af van het resultaat v/d partities, maar ook van de positie k ten opzichte van het splitelement, in alle partities.

o beste geval: partitie telkens in 2, een voud $n=2^k$

$$T(n) = 2T(n/2) + cn$$

- de halvering stelselmatig, en k pas op het einde gelijk aan i , getallen

$$T(n/2) = 2T(n/4) + c(n/2)$$

:

$$T(2) = 2T(1) + 2c$$

$$\Rightarrow T(n) = T(1) + \underbrace{(2+4+8+\dots+n)}_{2^{n-2}} c$$

$$T(n) = O(n)$$

o in het slechste geval

$$T(n) = cn + T(n-1)$$

$$T(n-1) = c(n-1) + T(n-2)$$

$$\vdots$$

$$T(2) = 2c + T(1)$$

$$T(n) = \underbrace{(2+3+4+\dots+(n-1)+n)}_{O(n^2)} c + T(1)$$

$O(n^2) \Rightarrow$ vermijden zo gevuld spilement kiezen

o het gemiddelde

- voor de eneouwst random pivot \Rightarrow geen veranderd stelling over de invoorstabel
- elke i even waarschijnlijk, $\frac{1}{n}$

$$T(n, k) = \frac{1}{n} \sum_{i=1}^n T(n, k, i) \quad \begin{cases} i=k, 0 \\ i < k, T(n-i, k-i) \\ i > k, T(i-1, k) \end{cases}$$

$$T(n, k, i) = cn + \sum_{l=1}^{k-1} T(n-i, k-l) + 0 + \sum_{i=k+1}^n T(i-1, k)$$

$$T(n, k) = cn + \frac{1}{n} \left(\sum_{i=1}^{k-1} T(n-i, k-i) + \sum_{i=k+1}^n T(i-1, k) \right)$$

o we innen de max k, een bovengrens (sterker), & ook indices aanpassen

$$T(n, k) \leq cn + \frac{1}{n} \max_k \left\{ \sum_{i=n-k+1}^{n-1} T(i) + \sum_{i=k}^{n-1} T(i) \right\}$$

- de recursieve beweering wegwerken, substitutiemethode
- bewijzen via induktie

stel $T(n) \leq a n$ met a een constante

- voor $n=1$, ok voldaan

- voor $n>1$, we stellen voldaan voor alle kleinere n

$$T(n, k) \leq cn + \frac{1}{n} \max_k \left\{ \sum_{i=n-k+1}^{n-1} ia + \sum_{i=k}^{n-1} ia \right\}$$

$$T(n, k) \leq cn + \frac{a}{n} \max_k \left\{ \sum_{i=n-km}^{n-1} i + \sum_{i=k}^{n-1} i \right\}$$

$\left(\frac{n+1}{2} \right) \mid \left(\frac{1}{2} \right)$

als $k = \frac{(n+a)}{2}$ blijft max!

we zoeken de k waarvoor
de som maximaal wordt!

$$\sum_{i=n-\frac{(n+1)}{2}+1}^{n-1} i + \sum_{i=\frac{n+1}{2}}^{n-1} i \Rightarrow \sum_{i=\frac{n+1}{2}}^{n-1} i$$

$\frac{2n - n - 1 + a}{2} = \frac{(n+1)}{2}$

rekentijdige reeks

$$(a) \quad 1 + 2 + 3 + 4 + \dots + (n-2) + (n-1) \quad (a)$$

$$(b) \quad \text{(nu onderste deel aftrekken)} - \frac{1 + 2 + 3 + 4 + \dots + \frac{(n-1)}{2}}{2} \quad (b)$$

1 term $\frac{(n+1)-1}{2}$ (want $\frac{n+1}{2}$)
haat een nog bij!

$$(a) = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

$$(b) = \frac{\frac{(n-1)}{2} \cdot \frac{(n+1)}{2}}{2} = \frac{(n^2-n+n-1)}{8}$$

$$a-b = \frac{4n^2-4n-n^2+1}{8} = \frac{3n^2-4n+1}{8}$$

$$\begin{aligned} & 3n^2-4n+1 \\ & \Downarrow \\ & D = b^2-4ac \\ & = 16-4 \cdot 3 \cdot 1 \\ & = 4 \end{aligned}$$

$$x_1 = \frac{4 \pm 2}{6} \Rightarrow 1 \text{ of } \frac{1}{3}$$

$$T(n, k) \leq cn + \frac{a}{n} \cdot \frac{(3n-1)(n-1)}{4}$$

$$\leq cn + \frac{a}{n} \cdot \frac{3n^2}{4}$$

$$\leq cn + \frac{3na}{4}$$

Uitwendig rangschikken

- er zijn toepassingen (individuele oper. of in functie v. b.v. een db) die zoveel gegevens moeten rangschikken, dat het uitwendig geheugen ontbereikend is, en dus moet beroep moet doen op het uitwendig geheugen.
- Probleem: toegang tot uitwendig geheugen is veel kostlijker / trager dan tot het uitwendig geheugen. Een dergelijke schijfoperatie is equivalent met duizenden berekeningen, dus is het gunstig om zo weinig mogelijk schijfoperaties te doen.
- de standaard methode ("uitwendig rangschikken") noemt men uitwendig samenvoegen, en ze bestaat uit twee fases:
 - a) inwendig rangschikken
 - b) uitwendig samenvoegen
- De gegevens worden opgedeeld in stukken, die men inwendig rangschikt, hierna voegt men deze stukken (of reeksen = runs) uitwendig samen (\Rightarrow mergesort)

inwendig rangschikken

Twee methodes:
a) load - sort - store
b) replacement - selection

a) load - sort - store

- elke reeks is even groot
- zoveel mogelijk gegevens inlezen
- sorteren met b.v. Quicksort $O(n \lg n)$
- en terug uitschrijven (tijdens rangschikken gebruiken er geen invoer/uitvoeroperaties)

b) replacement sort

- inwendig geheugen wordt gebruikt als een filter, waarbij continu gegevens worden ingelezen. Kleinere sleutels worden doorgelaten en in volgorde uitgeschreven, en gegevens met grotere sleutels worden tegengehouden.
- Elk uitgeschreven element is het kleinste v/h filter, het is minstens zo groot als het vorige uitgeschreven element.
- Grootte reeksen variërend: gemiddeld 2maal zo groot als bij **load - sort - store**
- Rangschikken gelijkheden met插入/uitvoer operaties.
- De laatste uitgeschreven waarde opgeslagen, een nieuw element wordt met deze waarde vergelijkt en wordt een besluit genomen of doorgelaten of tegengehouden (= als kleiner)
 - ↳ behoort tot volgende reeks
- Een reeks wordt afgesloten wanneer alle sleutels kleiner zijn dan de laatste uitgeschreven sleutel => start van de volgende reeks.
- Replacement selection kan het uitschrijven van sleutel zeer lang uitstellen, maar moet verder vooruit plaatsen dan de grootte v/h inwendig geheugen (n)
 - o Wanneer elke sleutel, wordt voorafgegaan door minder dan n grotere sleutels => 1 reeks
 - als de tabel dus al gerangschikt o ook in minder gunstige gevallen wordt goed gebruik gemaakt van de gedecideerde ordening.

extra

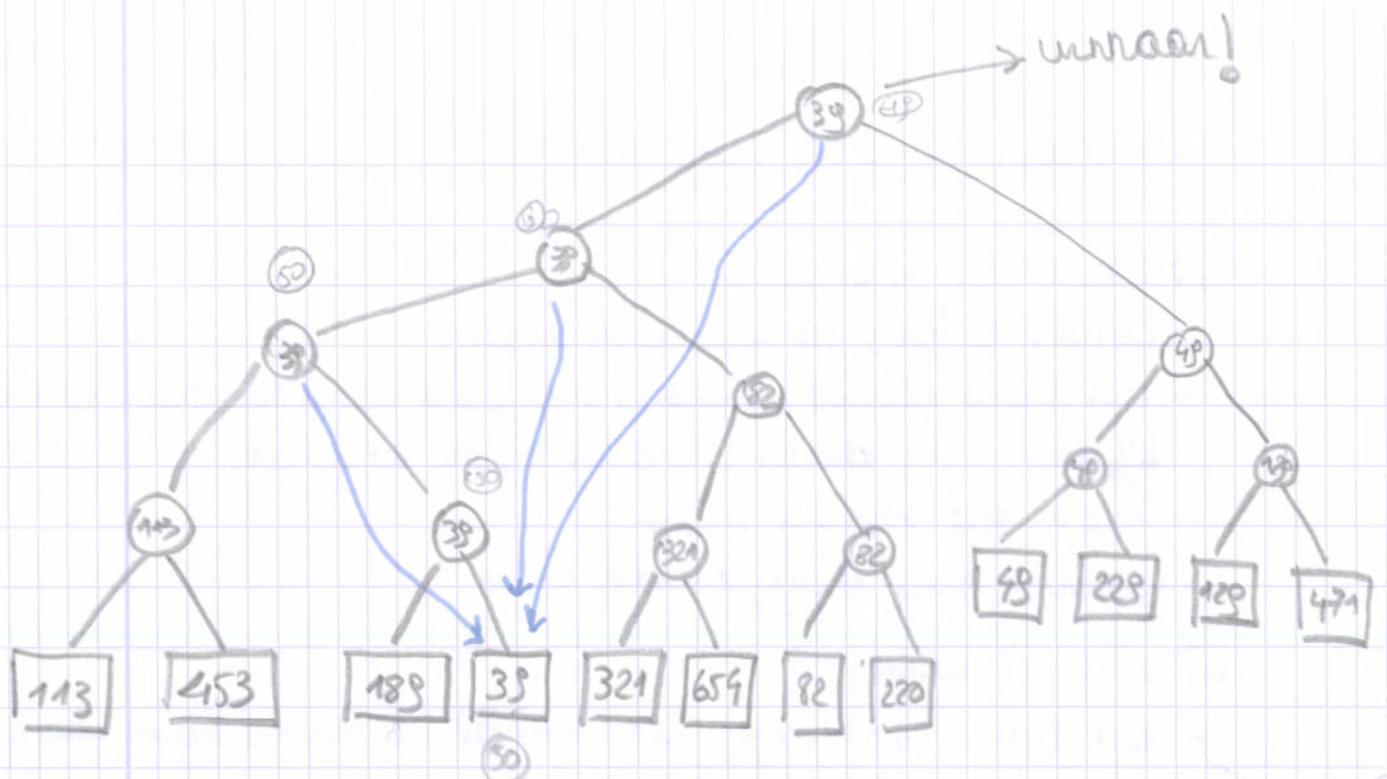
Structuren inwendig rangschikken

a) Heap (min) \Rightarrow binaire

- omdat telkens het minimum nodig
- een nieuw element (misschien even groot, als uitgeschreven) hoort bij de huidige reeks, anders bij de volgende.
- heap moet kleiner maken
- kunnen heap leeg, nieuwe reeks

b) selectieboom

- net zoals een heap, een complete binaire boom maar ook een volle! (0 of 2 kinderen)
- inwendige knopen \neq inwendige knopen (bladeren)
- elke inwendige knop, minimum van zijn deelboom met een pointer er naar toe. (blad)
- de gegevens zitten bij de bladeren.
- n gegevens, $n-1$ inwendige en n bladeren
- terminologie : turnooien & winnaars (de kleinste sleutel)
 \downarrow turnooi winnaar staat bij de wortel.
- gegevens van de huidige reeks & die v/d volgende in dezelfde boom. (een samen gestelde sleutel : reeks en sleutel)
 \downarrow lexico grafisch vergelijken (eerst reeks dan sleutel).
- boom initialiseren met fictieve gegevens v/c muide reeks. ook op het einde.
- als gegeven (wortel) wordt uitgeschreven, nieuw element op de plaats v/d blad waarnaar de pointer v/d wortel verwijst
- grootte v/d boom in constante



- bewegen langs een weg van onder naar boven, voor het aanpassen vd wortel. (bij een heap van bomen m beneden).
- slechts worden getest, die kunnen wedstrijd verloren tegen de oude wortel.

↳ hiervoor een alternatieve boom : selectieboom v. verliezen

- ook terug een ↗ weg van onder naar boven.
- nu slechts een blad staan in een inwendige knoop, je kan slechts 1 maal verliezen
(elke knoop bevat de grootste van de kleinste uit elke deelboom)

Implementatie replacement-selectie

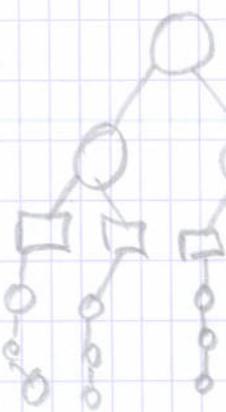
- probeert om de selectieboom zo klein mogelijk te maken
- een stijgende weg volgen in een grote selectieboom zorgt voor veel cache fouten, en er is geen verband tussen op een volgende weg. Knopen dicht bij de wortel bij de wortel komen op veel wegen voor, maar dit is niet zo voor lagere gelegen knopen:

Enkele voorstellen:

- a) gegevens vld knopen worden in een aparte geheugenzone opgeslagen. (bij deel koepassagen ligt deze grootte vast)
 - probleem met gegevens van ≠ groottes. (een weggewreven element kan niet vervangen worden door een nieuw element omdat er te weinig ruimte is.)
 - selecteren & vervangen ontkoppelen: nieuwe gegevens worden toegevoegd zolang er plaats is, en uitgeschreven zolang elementen niet toegevoegd kan worden.

gevolg: aantal knopen vld selectieboom wordt variabel

- b) een minireeks aan elk blad toewijzen



- minireeksen worden vooraf gesorteerd met Quicksort
- een weggewreven element wordt door het volgende element v/e minireeks
- aanpassen boom gelijk
- nieuwe gegevens eerst verzameld in een minireeks, wanneer vld voor deze gesorteerde reeks een adres wordt toekendend.

- ? - aantal knopen variabel
- Men koppelt deze reeks aan een blad, en voegt lege element toe.
 - Een lege reeks wordt niet vervangen maar verwijderd uit de boom

beschikbare schijven

- één schijf : laad - work - stage , kan slechts 1 schijf tegelijkertijd gebruiken. Een schijf bevat in voorverstand en de rechtes
- twee of meer :
 - o replacement selection. Wanneer er meerdere schijven beschikt past men die schijving toe. Simuleren van 1 schijf met grotere blokgradiet.
 - o opvolgende blokken van dezelfde rads worden gelijktijdig op dezelfde cilinders van opvolgende schijven opgeslagen

uitwendig samenvoegen

to-do / nog nooit examenvraag over gesteld

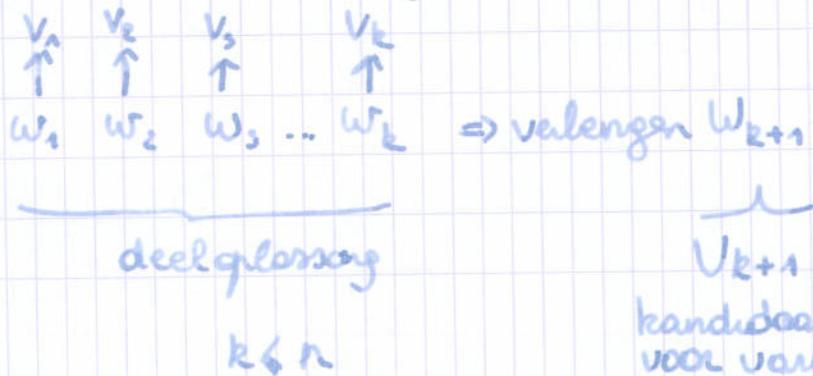
Backtracking

- er bestaan veel praktische problemen waarvoor nog geen efficiënt algoritme is gevonden (P-NP probleem) => lessen roeden opstellen
↳ (en wss ook nooit).
- alle mogelijke kandidaatoplossingen overlopen & uitproberen.
- enkel zinvol als de "afmetingen" van het probleem relatief klein zijn. (combinatorisch) => anders heuristische algoritmen
- alle kandidaatoplossingen uitzetten, 1maal! incrementeel construeren. (achtereenvolgens componenten toevoegen, tot een oplossing gekomen is.) - waarden aan variabelen toekennen

Voortetting

- toekennen van waarden aan variabelen, rekening houdend met bepaalde voorwaarden (SAT, simulated annealing)
- een keuze voor de volgorde v/d variabelen, en een volgorde voor de mogelijke waarde van elke variabele.
- achtereenvolgens waarde toekennen aan de variabelen
- wanneer geen waarden meer, keert men terug in de loop.
- een oplossing: als alle variabelen een geldige waarde hebben toegewezen.
- geen oplossing: als geen enkele waarde voor de eerste variabele voldoet.

(op stappen terugkeren, alle kandidaat oplossingen worden precies eenmaal gegenereerd & als de oplossing bestaat, zal deze zeker gevonden worden)



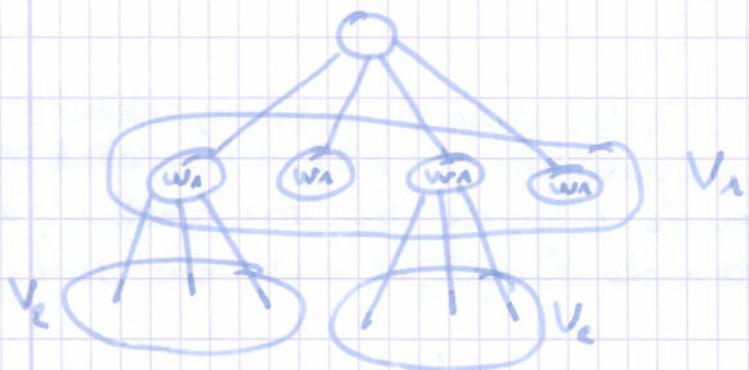
$\overbrace{\quad\quad\quad}^{V_{k+1}}$

kandidaat waarden berekenen voor variabele $k+1$

- als de verzameling V_{k+1} leeg is keren we terug op onze stappen. \rightarrow volgende waarde voor w_k (\rightarrow kan nieuwe verzameling voor V_{k+1} geven)
- backtracking, stopt als er geen nieuwe keuze meer gemaakt kan worden vi. de eerste variabelen.

Vertelling

- deeloplossingen als knopen van zoekboom
- een kind van knoop is dan een deeloplossing die uit de knoop wordt afgeleid (verlengen om een waarde).
- wortel begin van backtracking proces



- diepte bepaald door het aantal variabelen
- backtracking recursief afdalen via boom (o.d. level onder, hogere gehangen vertakken)

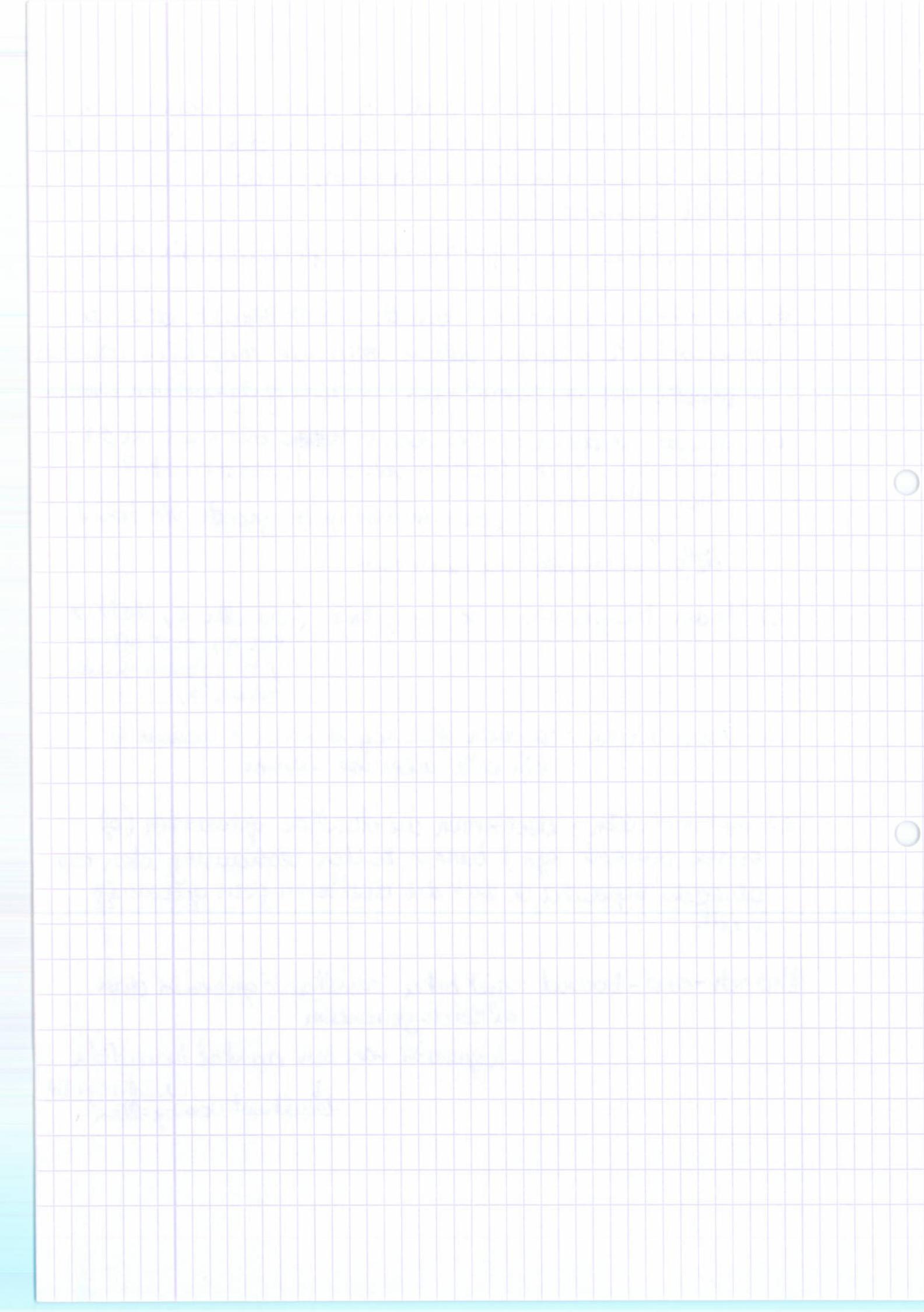
Nadeel voor niet al te grote n , kan de boom zeer groot worden. \Rightarrow Performantie verbeteren door het aantal knopen zoveel mogelijk te beperken, en zo weinig mogelijk te onderzoeken.

- Snoeien/pruning (spectaculaire performantieverb.)

- a) variabelen ordenen : grootte vaardt beperkt, door de variabelen te ordenen volgens steigende mogelijke waarden
 - grootte v/d verzamelingen : statisch of dynamisch bepalen.
- b) waarden ordenen : interessant ~~is~~ als men slechts 1 oplossing zoekt. Ordenen volgens dalend aantal mogelijkheden.
 - (geen invloed op de grootte v/d boom)
↓
de "gemakkelijkste" waarden eerst
- c) Verder terugkeren: x, y, \dots (als elke z , slechts 1 voor x , niet naar y terugkeren maar naar x)
- d) Vooruitkijken : na paar of er nog minder 1 waarde is voor alle volgende waarden
- e) Symmetrieën : deelbomen die dezelfde oplossingen (of meer recent zijn) kunnen buiten beschouwing laten als alreeds bepaald is dat die deelboom geen oplossing biedt.

Branch-and-bound : niet beter resultaat opleveren dan alreeds gevonden

- toepassen van een aantal heuristische
 \downarrow (ruutregels)
resultaat voorspellen



Systematisch overlopen

O gericht

- recursief bij bomen, diepte eerst (alle knopen overlopen)
- levelorder bij bomen, breedte eerst. \rightarrow en alle verbindingen

o bomen: 1 weg v/d wortel nr een blad, bij graaf niet (er zijn \neq wegen mogelijk) \Rightarrow bijhouden bezocht

- buren ips kinderen
- bijhouden v. kleuren wit, grijs, zwart

o boomtak
o terugverbinding \Rightarrow met kleur (grijs \rightarrow wit)
o geen verbinding (naar grijs) \rightarrow zwart (grijs \rightarrow zwart) \rightarrow pre-
o duurs verbinding (grijs \rightarrow zwart) \rightarrow oide

- terugverbinding: een huis (knopen v/e huis behoren tot dezelfde boom)

O ongericht

- elke verbinding 2 maal \rightarrow terugverbinding & geen verbinding bestaan niet.
- een verbinding (kind \rightarrow ouder) geen terugverbinding

- o ontdekt
- a) de volgorde van ~~ouder~~ voorouders
 - b) de volgorde van afgeleide potraten

- o andere operaties op knopen, overlopen v/d buren elkaar eraf etwissen.

Efficiëntie

- initiatie $\Theta(1)$ operaties
- zoeken in nieuwe startknop loopt gekondaan $\Theta(n)$
- elke knop 1 maal ontdekt $\Theta(n)$
- elke buur van elke knop evalueren $\Theta(n)$ \rightarrow gelukkig by $\Theta(n^2) \Rightarrow$ matrix

$$\Theta(n+m) \text{ of } \Theta(n+n^2) \Rightarrow \Theta(n^2)$$

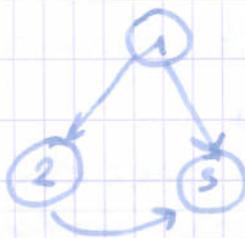
o breedte-eenr zoeken

- wachtrij met de nog niet ontdekte knopen
- level-order \Rightarrow bomen
- zwart, wit, grijs

o - boomtak

- terugverbinding (terugverbinding niet bij opeendende)
- dwarsverbinding (ongerelateerde knopen uit dezelfde boom op hetzelfde niveau of laag)
- gericht:
 - dezelfde niveau hoger
 - ell hoger
 - 1 laag
- geen verbindingen niet mogelijk
- boomtakken: een overspannende boom
- first in - first out
- breedte eerste \Rightarrow lussen in ongerichte graaf
 \Rightarrow minimaal aantal lussen

zelfde als diepte eerst



1
X 25
X 3

Zoeken in een gerangschikte tabel

- enkel de moeite als frequent gezocht zal worden
- o sequentieel zoeken:
 - sleutel niet in tabel, gemiddeld moeier stoppen (een sleutel die past)
 - binair zoeken: telkens de helft elimineren

$$\Theta(\lg n) \quad 1000 \Rightarrow 10$$

$$1000.000 \Rightarrow 20$$

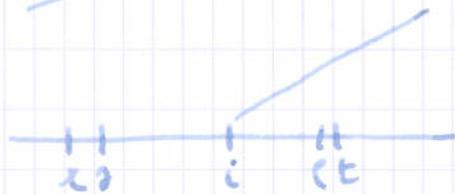
verschillende varianties

enkele varianten

- a) cyclisch gerangschikt

$$x_1, x_2, x_3, \dots, x_i, \dots, x_{n-1}, x_n$$

$\leftarrow l \text{ min}$ zoek } i ?



- door de 2 waarden goed te kiezen kan men telkens de helft elimineren

- b) gerangschikt onbekende lengte

x_1, x_2, \dots (elementen berekend)

- 1) zoek selecteer $l < r$
nu is $l < r$ dus kan i er niet liggen, dan is $l = i$ of wel later
dus kijk in interval r, l

- 2) set
nu is $s > t$ dus ligt C erin

- nu berekenen vi duur \Rightarrow zo weinig mogelijk rekenen

- zoek element y

- zoek waarde niet kleiner dan y

↳ aantal $x_1, x_2, x_3, x_4, x_5, \dots$ tot $y \leq x_j$

$\Theta(\lg j)$ om index y te vinden als $j \geq 1$

lager \Leftrightarrow in $\frac{j}{2} \leq y < j$ nog keer binair zoeken

$\Theta(\lg j)$

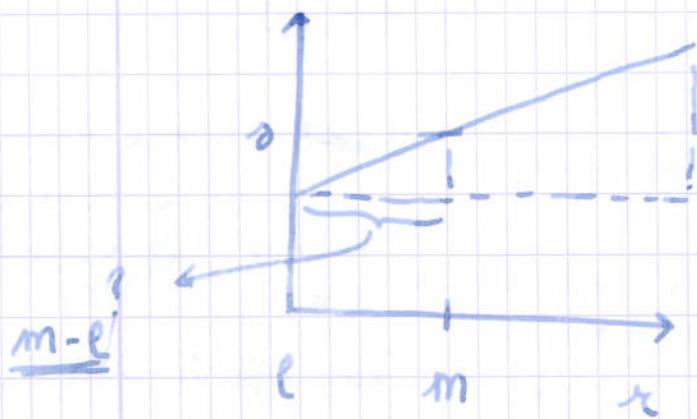
interpolerend zoeken (zoeken in een woordenboek)

Bst. zoeken geen rekening met waarsch. heidswend. Interpol. zoeken tent. tellens op de plaats waar eerst sleutel s , verwacht wordt. (een voudig bij uniform verdeelde sleutels).

uit m te berekenen $\alpha + \frac{(x-\alpha)}{2}$ vervangen we $\frac{1}{2}$
door: $\frac{(s-t[e])}{(t[r]-t[e])}$ } binne de tabel 1 geheel

Gemiddeld $O(\lg \lg n)$, zowel o.a.m.s.g als aanvraag sleutel

- alleen gunstig zeer grote tabellen (berekening iets complexer)
- gebruikt bij uitwendig zoeken, zo weinig mogelijk schijf operaties,



$$\frac{m-l}{s-t[e]} = \frac{x-l}{t[r]-t[e]} \Leftrightarrow \frac{s-t[e]}{t[r]-t[e]}$$

- slechtste geval $O(n)$

↑
laatig! juo $\frac{1}{2}$

Hash tabellen

- toepassingen: zoeken, toevoegen en verwijderen onderstrepen
- symbol table: namen v. procedures en variabelen
- efficiënte implementatie en veralgemening v/d rechtstreeks adresseerbare tabel
- rechtstreeks addit. tabel onbruikbaar $\#$ mog. sl. > plaats
- compromis tussen plaats en tijd
- geen eenduidig verband \Rightarrow botsingen, collisions
- plaats berekent met een hashfunctie \Rightarrow primaire index
- prijs: enkel gemiddelde performance (uitstekend (niet steeds))
- perfecte hashing (slechts op voorhand gekend), niet zo als toevoegen & verwijderen (dynamisch)
- tevreden met bijna perfecte hashfuncties \Rightarrow gering aantal collis.

a) opvangen van conflicten

Chaining

1) Separate chaining (open hashing)

- tabel van lijsten
- slechtste geval alle elem. in 1 lijst \Rightarrow zoeken $O(n)$
- gemiddeld geval: hangt af v/d kwaliteit v/d hashfunctie, na te streven: elke sleutel met dezelfde waarschijnlijkheid bij elke lijst \times onafhankelijk v. elkaar \Rightarrow binomiale verdeeling $\Rightarrow \frac{n}{m}$ (gemiddelde lengte v. elke lijst) $\Rightarrow \alpha$ loadfactor

$$1 \geq \alpha < 1$$

- zoeken m. o.f. gemiddeld sneller als gesloten schikt maar lijsten zijn dat dus speelt geen rol of gesloten schikt we kunnen met dezelfde kans bij elke lijst a elke lijst gemiddeld α $\Rightarrow \Theta(1+\alpha)$
- zoeken aanwezig: uigenaardiger, hoe meer sleutels een lijst bevat hoe meer kans dat hij in de lijst zit. Toch ook $\Theta(1+\alpha)$

2) coalesced chaining

- een tabel v. lijstknopen : elke knoop een begin zijn v. lijst maar ook gebruikt om een andere lijst te verlengen. De lijsten fusioneren
 - knopen niet dynamisch aanmaken, voordeel, maar ook nadadel als de aantal gewenste slakels niet gekend
 - aantal varianten : met of zonder cellar
 - o adreszone & cellar
 - uitsluitend prim. indexen berekenen in de adreszone
 - bij collis. neem knoop uit de berging & lijst verlengd
 - als berging vol, gebruik knopen uit de adreszone.
 - Blz. adreszone 86% zeer goed.
 - o standaard
 - zonder berging lijsten voegen fusioneren
 - een berging toegewand voor alle conflicten \Rightarrow sep. chain
 - verwijderen maakt het iets moeilijker \Rightarrow lazy deletion
 - zoeken, obt alsof verkeerde slakel
 - toevoegen, alsof leeg
 - eenvoudig laatste index v/d lege knoop.
- Onderneem verwijderen beft separate chaining
- Experimenteel : $\alpha > 0,6$

Open addressering (closed hashing)

- net zoals coalesced chaining, alleen wordt plaats vrijgevuld voor tijd
- alternatieve plaatsen zijn conflicten op te lossen.
- de alternatieve plaatsen liggen vast aangezien alle plaatsen berek moet worden
- op voorhand een idee van aantal elementen
- α moet groter dan 1

zoeken

niet

- zoeken naar lege plaats, als verkeerde sleutel berken een nieuwe plaats tot een lege plaats

toevoegen

- een lege plaats zoeken (als tabel niet vol is.)
- kan een volledige tabel gevullen, toch razer slechte prestaties
naar het einde toe, daarom kiest men op de hash-tabel groot genoeg te houden
- voor hetzelfde aantal sleutels, meer plaats dan sep. hash. ~~doen~~
maar door tewallen van wijzers (tach bitten)

Voordeel: plaats bestaat al

Nadeel: niet efficiënt uitbreiden

verwijderen

- lazy deletion
- sleutel kan tot meerdere zoeksequenties behoren
- zoekijd hangt niet enkel af van α maar ook de latente aanwezige sleutels

bepalen vld zoeksequenties

- alternatieve plaatsen berekenen
- garanderen dat de volledige tabel doorzocht wordt.
- ideaal: elke permutatie even waarschijnlijk
↳ moeilijk te realiseren (uniforme hashing)
- slechts beperkte fractie vld $m!$ permutaties

a) linear probing

$$(h(s) + i) \bmod m \quad \text{met } i = 0, \dots, m-1$$

- telkens 1 plaats verder zoeken
- slechts m verschillende sequenties mogelijk
- primaire clustering
- eenvoudig
- kans clustering is reëel, $\frac{(i+m)}{m}$



b) kwadratische hashing

$$(h(s) + c_1 i + c_2 i^2) \bmod m$$

- $c_1 \neq c_2 \neq 0$
- om alle indexen te kunnen berekenen c_1, c_2 en m aan voorwaarden voldoen.
- nog steeds dezelfde zoeksequentie (n)
- slechts secundaire clustering (probleem in theorie)

c) double hashing

$$(h(s) + h'(s)i) \bmod m$$

- benadrukt uniforme hashing meer
- m^2 verschillende sequenties
- zowel h als h' hash functies ($\neq !$)
- waarde $h'(s)$ relatief priem met m
- als m priem: $0, h'(s) < m$
- als m even dan $h'(s)$ oneven

Performantie open adresseering

- gemakkelijker bij een uniforme hashfunctie (optimistisch)
- performantie wordt volledig bepaald door de bezettingsgraad α ($\text{root} > 1$)
- o hoeveel testen zijn er gemiddeld nodig om een afwezige sleutel te zoeken?

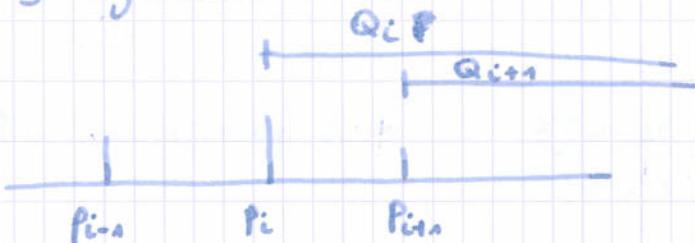
Als p_i de waarschijnlijkheid voorstelt dat er exact i testen moet gebeuren, dan wordt het gemiddeld aantal testen gegeven door:

$$\sum_{i=1}^{n+1} i p_i$$

? wtf?

met n het aantal bezette plaatsen in een hash tabel van m groot. + 1 extra test

- probleem: hoe kan je p_i berekenen.
- oplossing: afleiden uit $Q_i \Rightarrow$ de kans dat tenminste i testen.



nu is $p_i = Q_i - Q_{i+1}$ a gelukkig kunnen we Q_i wel berekenen.

de uitrekking wordt nu:

$$\sum_{i=1}^{n+1} i (Q_i - Q_{i+1}) \Rightarrow 1.Q_1 - 1.Q_2 + 2.Q_2 - 2.Q_3 + 3.Q_3 - 3.Q_4 \dots$$

$$\sum_{i=1}^{n+1} Q_i \quad \text{maar nu hoe bereken je } Q_i?$$

we beginnen bij tenminste 1 test

- de kans tenminste 1 test $\rightarrow Q_1 = 1$, want er moet altijd 1 test gebeuren!
- de kans tenminste 2 testen: de 1^e test was dus bezet, nu hebben we aangenomen dat er n beschikbare plaatsen zijn, en dat elke plaats even waarschijnlijk is $\Rightarrow \frac{1}{m}$, de kans dat er tenminste 2 testen is dus $Q_2 = \frac{1}{m}$
- nu Q_3 ? de kans tenminste 3 testen is op voorwaarde dat de 1 & 2 test bezet waren. Nu is de kans dat 1 of 2 bezet zijn
 $P\{1 \cap 2\} = P\{1\} \cdot P\{2|1\}$
- daarbij als een 3^e test gebeurd is één vld n plaatsen getest geweest en is er dus 1 plaats minder die getest moet worden dus is

$$Q_3 = \frac{n}{m} \cdot \frac{n-1}{m-1}$$

$$Q_i = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-i+2}{m-i+2} \quad \text{dus bovengrens alles } \frac{1}{m}$$

$\underbrace{\phantom{\frac{n}{m}}}_{\nwarrow}$

$$Q_i = \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}$$

terug steken in formule

$$\sum_{i=1}^{n+1} Q_i \leq \sum_{i=1}^{n+1} \alpha^{i-1} \leq \sum_{i=0}^{\infty} \alpha^i \quad \left. \begin{array}{l} \text{reeks} \\ = \frac{1}{1-\alpha} \end{array} \right\} \text{als } \alpha \approx 1 \text{ zeer veel testen}$$

o gemiddeld aantal testen bij toevoegen is dus herzelfde

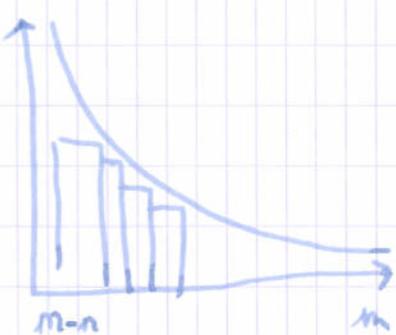
o hoeveel keren vereist zoeken naar een aanwezige sleutel?

- zoek sequentie zelfde als waardeer hij werd toegevoegd & dus nog afwezig was.
- Als een sleutel werd toegevoegd aan een tabel die alreeds i elementen bevatte, dan was het gemiddeld aantal testen

$$\frac{1}{1-i/m} = \frac{m}{m-i}$$

nu als elke sleutel met dezelfde waarschijnlijkheid gekozen wordt bijgaan we

$$\frac{1}{n} \sum_{i=0}^{i-1} \frac{m}{m-i} = \frac{1}{\alpha} \sum_{i=0}^{i-1} \frac{1}{m-i} = \underbrace{\frac{1}{\alpha} \sum_{k=n-n+1}^m \frac{1}{k}}_{\text{Riemann}} \quad \underbrace{\text{integraal nemen}}$$



wieker groter?

$$\begin{aligned} \frac{1}{\alpha} \sum_{k=n-n+1}^m \frac{1}{k} &< \frac{1}{\alpha} \int_{m-n}^m \frac{1}{k} dk \\ &= \frac{1}{\alpha} \left(\lg k \right) \Big|_{m-n}^m \\ &= \frac{1}{\alpha} (\lg m - \lg m-n) \\ &= \frac{1}{\alpha} \lg \frac{m}{m-n} = \frac{1}{\alpha} \lg \frac{1}{1-\alpha} \end{aligned}$$

o reductie: als α een bepaalde threshold overschrijdt

Hashfuncties

- goede hashfunctie moet geëvalueerd, conflicten minimaliseer
- deterministisch
- enkelvoudige uniforme hashing:
 - eldere tabel index even waarschijnlijk m. elke deutele
 - onafhankelijk vld indices voor de andere sleutels
- hashfunctie ontwerpen is moeilijk omdat de waarschijnlijkheidsverdeling niet gekend is. (heuristische methode)
- symbol table: max, maxy, maxz \Rightarrow Ver van elkaar
- een functie zoeken onafhankelijk van enig patroon in de gebruikte sleutels.

a) varste hashfuncties

- veronderstellen dat de sleutels in een procesor word kunnen, en dus als positieve gehele getallen voorgesteld.
- uitvoergetallen tonen aan & goede methoden
 - delen
 - vermenigvuldigen

1) delen: geheel getal om in een index

$$h(s) = s \bmod m$$

- een voudig efficiënt
- random sleutels, ook gelijmatig verdeeld.

de grootte m moet wel met enige voorzorg worden gekozen:

- als m een \rightarrow even sleutels, even indicies oren sleutels, oneven indicies
- als $m = 2^i \rightarrow$ index uit de laaste i bits vlt bitpatroon (ok, als laaste i bits van elke sleutel even waarschijnlijk is.) anders beter alle bits
- macht van 10 \rightarrow als de sleutels decimale getallen zijn
- goede waarden: priemgetallen niet te dicht bij machten van 2
- uitleggen hoe gelijmatig een hashfunctie sleutels verdeelt

o vermenigvuldigen

- in 2 stappen a) sleutel wordt vermenigvuldigd met een constante c , $0 \leq c < 1$
- maakt niet uit welke waarde m , maar gewoonlijk 2^i omdat dan zonder vlottende komma
kommagetal $\rightarrow 2^w$
 $c \times \boxed{} = \text{geheel getal}$

$$\lfloor (c - \lfloor c \rfloor).m \rfloor$$

m een macht van 2

$$m = 2^k$$

$$\lfloor c \cdot 2^w \rfloor \text{ (1 keer uitrekenen)}$$

$$\begin{array}{c} \downarrow \\ \frac{w}{2} \quad \boxed{} \quad \times 2^w = \\ \frac{2w}{2} \quad \boxed{} \quad \text{delen} \\ \frac{W+w}{2} \quad \frac{R}{2^w} \quad R \text{ breedt} \end{array}$$

geheel getal maar
 2^w te groot
dus delen door 2^w

- voor grote sleutels die niet in een processor woord passen
beperkte mogelijkheden

o ell lang bitpatroon, groot geheel getal: software matige
sommeert als meerdere woorden.
↳ inefficiënt

Inkring:

- een getal maken volgens bepaald halstelsel (best prien)

horner $C_0 + C_1 x_1 + C_2 x^2 + \dots$

modulo 2^w (niet te groot woord)

- voor ell woord een hashfunctie

Universele hashing

- hashfunctie ligt vast
- nog steeds mogelijk, een hashfunctie allemaal dezelfde indices berekent
- plannen een random functie, bij gebruikname v/d tabel
- een randomized algoritme
- gekozen uit een zoig veelalige gekozen familie van functies
 - ↳ de kans dat een willekeurige gekozen ~~familie~~ functie van $2 \neq n$ reekels eenzelfde index berekent $\frac{1}{m}$

Binaire zoekbomen

- extra ordening
- zoeken $O(h)$
- mun $O(1)$
- volgorde (vandaar $O(n)$)
- volgen
 - $O(h)$
 - 1) rechte deelboom ~~leeg~~ niet leeg
 - 2) rechte deelboom leeg (ander wijzen, tot eerste keer rechts afslaan)
- toevoegen $O(h) \rightarrow$ zoeken + aanhangen
 - gelijke deurstels
 - a) gelijktijdige lijst
 - b) een logische waarde, verwijzen
 - c) random getal

Performance zoeken & Toevoegen

Path length v/d afgelegde weg bij zoeken & toevoegen is nooit langer dan de hoogte v/d boom

o slechste geval, gelijke lijst $O(n)$

o gemiddeld: Veronderstellen dat de boom random werd opgebouwd
elke toevoeg volgorde evenwaarschijnlijk

- Men kan aanhouden gemiddelde hoogte $O(\lg n) \Rightarrow$ bestig
- Een voordeel: de gemiddelde diepte v/d knoop (wortel)

Gemiddelde diept: \Rightarrow gemiddelde verwachte weg lengte

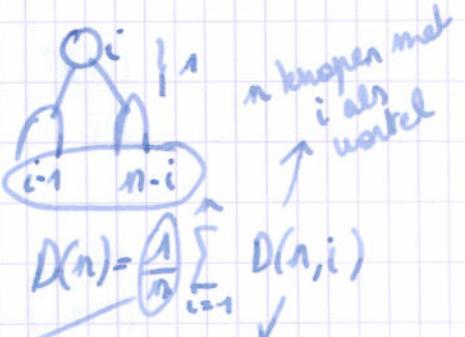
\rightarrow elke toevoeg volgorde even waarschijnlijk \Rightarrow elke element even waarschijnlijk om de wortel te zijn.

Stel wortel i dan



$$D(n) = n-1 + \frac{1}{n} \sum_{i=1}^{n-1} D(i-1) + D(n-i)$$

$$= n-1 + \frac{2}{n} \sum_{i=0}^{n-1} D(i)$$



elle knoop is evenwaarschijnlijk om de wortel te zijn

$$D(n,i) = D(i-1) + D(n-i)$$

$$+ i-1 + n-i$$

$$= D(i-1) + D(n-i)$$

$$+ n-1$$

$$D(n) = n-1 + 2 \sum_{i=0}^{n-1} D(i)$$

$$\text{or } D(n) = n^2 - n + 2 \sum_{i=0}^{n-1} D(i)$$

- } \| \downarrow n-1

$$(n-1) D(n-1) = (n-1)^2 - (n-1) + 2 \sum_{i=0}^{n-2} D(i)$$

$$\begin{aligned} n D(n) &= (n-1) D(n-1) = n^2 - n + 2 \sum_{i=0}^{n-1} D(i) - \left[(n^2 - 2n + 1) - (n-1) + 2 \right] \\ &= 2n - 2 + 2 D(n-1) \end{aligned}$$

$$n D(n) = (n-1) D(n-1) + 2n - 2 + 2 D(n-1)$$

$$n D(n) = (n+1) D(n-1) + 2n \cancel{- 2} \quad \text{constant weg}$$

- telescoping $\frac{1}{n(n+1)}$

$$\frac{D(n)}{(n+1)} = \cancel{\frac{D(n-1)}{n}} + \frac{2}{(n+1)}$$

$$\cancel{\frac{D(n-1)}{n}} = \cancel{\frac{D(n-2)}{n-1}} + \frac{2}{n}$$

:

$$\cancel{\frac{D(3)}{3}} = \cancel{\frac{D(1)}{2}} + \frac{2}{3}$$

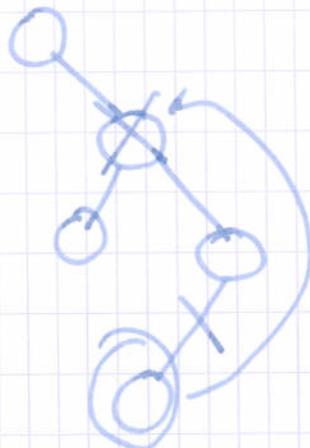
$$\frac{D(n)}{(n+1)} = 2(n+1) \cdot \underbrace{\left(\frac{1}{n+1} + \frac{1}{n} + \dots \right)}_{H_n = \lg n}$$

$\Theta(n \lg n) \Rightarrow Q(\lg) \text{ als elke sleutel met dezelfde sleutel doorzocht wordt}$

Vervijderen

- lazy deletion (interessant geblokkeerde lijst voor duplicaten)
- veel verwijderen, effectief verwijderen (moeilijk)

- 1) 0 kinderen,
- 2) 1 kind (bypass)
- 3) 2 kinderen (voorganger)



Threaded tree

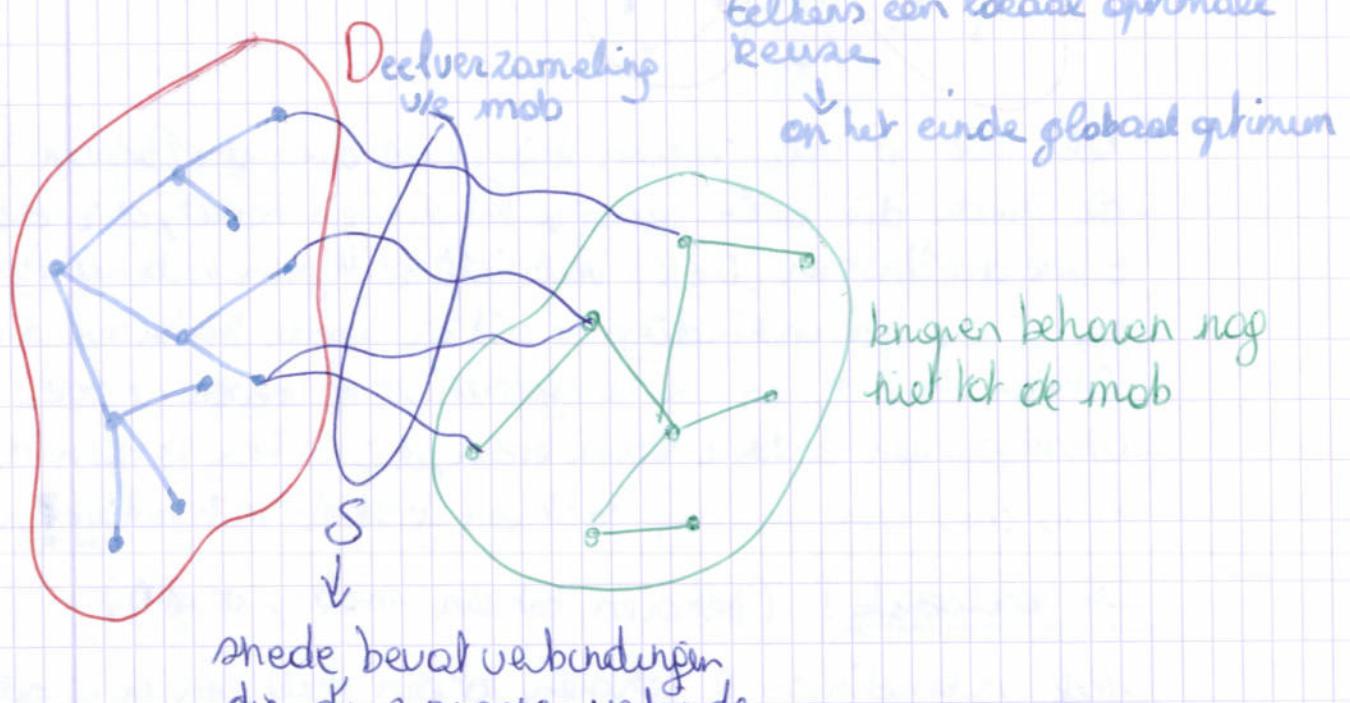
- $n+1$ verwijzgers, kan beter gebruikt worden

$$2n - (n-1) = n+1$$

- linker \rightarrow voorloper
- rechter \rightarrow volgeling
- thread verwijzgen steeds om hoger

Minimale overspannende boom (mob)

- een overspannende boom \rightarrow alle knopen vld graaf
 - \rightarrow slechts deel vd verbindingen
 - \hookrightarrow worden gegenereerd door breedte eerst & diepte eerst
 - \hookrightarrow niet uniek
- als een gewogen graaf: som vld gewichten vld takken
vld overspannende boom = gewicht
- een overspannende boom: met minimaal gewicht noemt
men dan ook een minimale overspannende boom.
 - \hookrightarrow ook niet uniek, wel als alle gewichten \neq zijn
- mob's behoeft tot veel toepassingen
- een mob wordt geconstrueerd tak per tak (greedy)



- vervolgens wordt uit deze collectie S , de lichtste verbinding
gepakt & gebruikt om de verzameling D , uit te breiden.

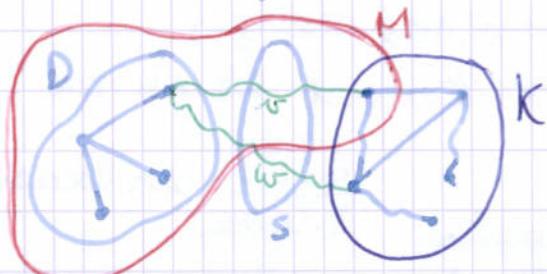
Gegeven een deelverzameling D van graafverbindingen die al behoren tot een mob, en een graafverbinding v , die niet tot D behoort.

Omdat v samen met D een mob zou vormen is nodig & voldoende dat er een snede S bestaat, die geen verbindingen van D bevat & waarin v de lichtste is.

Aantoonen eigenschap

Is nodig? (v is een snede nodig met v de lichtste)

- M is een mob, die verbindingen van D bevat & v



- Als we v verwijderen krijgen we twee deelbomen D & K
- de Snede die alle graafverbindingen bevat, die deze twee deelbomen (D & K) verbindt heeft geen verbindingen van D .
- Als er nu een verbinding w uit de snede halen en die is lichter dan v , en we verwijderen w & voegen w toe.
- We krijgen een lichtere bres. boom, die lichter is dan M .
- En is dus een snede nodig met een lichtste verbinding!

Is voldoende? (behoren tot een mob, v & D)

Stel verbindingen D behoren tot een mob, en er is ook een snede S met v de lichtste. Is dan v & D een mob?

- Stel neen! Kies dan andere Mob die D bevat (met w)



- als we nu ~~w~~ v terug toevoegen, krijgen we eenlus.
- dus moeten nog andere knopen die deel mogen. Verbonden
- als we w verwijderen krijgen we een overspannende boom die zeker niet saamgaan is want v was de lichtste

Prim

- eam mob qhannen, tak per tak (greedy): pakken wat je kan krijgen & nooit op je schapen terugkeren.



- bijhouden, elke knop die nog niet tot de mob behoort (zijn lichtste verbinding met de mob)
- behalen tot alle knopen in de mob ($n-1$) verbindingen

- prioriteitswachtrij: min hech - met als prioriteit gewicht

- knopen
- andere eind knop

- prioriteit aanspreken (gaat niet)

- meerdere keren verdeken (takken of nog niet al ingevuld)
 $\hookrightarrow O(n)$

- $O(n)$ critiekserie

- $O(m \lg n)$

- oorspronkelijk een label

Korte afstanden te gemakkelijk, goed uitgelegd in cursus

Floyd warshall zie cedricue.me uitleg

