FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Gevorderde algoritmen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

UNIVERSITEIT
GENT

Inhoudsopgave

Deel I

Theorie

Hoofdstuk 1

Inleiding

Het vak gevorderde algoritmen behandelt vier luiken:

1. **Gegevensstructuren**. Dit onderdeel behandelt meer efficiënte zoekbomen zoals Rood-zwarte bomen, Splay trees, B-trees en meerdimensionale gegevensstructuren zoals Quadtrees en k-d trees. Er wordt ook bekeken hoe zeer grote datastructuren (die niet vollledig in het geheugen passen) behandelt worden.

2. **Grafen**. Een uitbreiding op de grafentheorie. Onderwerpen zoals stroomnetwerken, _ToDo: aanvullen wanneer stof gezien is

3. **Strings**. Dit hoofdstuk gaat dieper in op stringfuncties. Een aantal onderwerpen zijn efficiënte zoekmethoden, de theorie achter reguliere expressies en hun grammatica en het samenvatten van teksten.

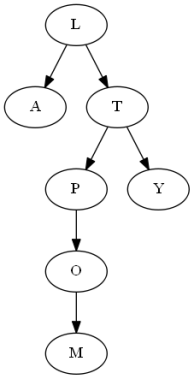
4. **Complexe problemen**. Dit onderdeel behandelt de NP-problemen. Wat zijn ze? Hoe kunnen we NP-problemen transformeren naar een eenvoudiger probleem? Hoe kan men benaderde oplossingen gebruiken?

Hoofdstuk 2

Efficiënte zoekbomen

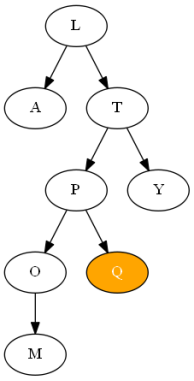
2.1 Herhaling binaire zoekbomen

Een binaire boom is een belangrijke boomstructuur waarin drie operaties belangrijk zijn: *zoeken, toevoegen en verwijderen*. Deze drie operaties worden kort herhaald op figuur 2.1



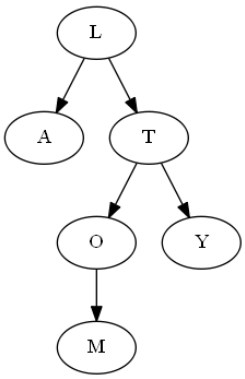
Figuur 2.1: Een binaire zoekboom.

- **Zoeken** naar een knoop komt overeen met binair zoeken. Stel dat we knoop P zoeken dan weten we dat dit groter is dan L dus gaan we naar rechts, en kleiner is dan T dus gaan we naar links.
- **Toevoegen** van een knoop komt overeen met zoeken naar die knoop, en dan de knoop op die positie toe te voegen. Stel dat we een knoop Q willen toevoegen, dan zal het binair zoeken leiden dat deze knoop in het rechterkind van P moet komen, zoals te zien op figuur 2.2
- **Verwijderen** van een knoop onderscheidt drie gevallen:
    1. *De knoop heeft geen kinderen*: dan kan de knoop eenvoudig verwijderd worden. Indien knoop Q verwijderdt wordt krijgen we terug de originele boom.



Figuur 2.2: De knoop Q is het rechterkind van knoop P.

2. *De knoop heeft één kind*: In dit geval moet de ouder die naar de te verwijderen knoop wijst, nu wijzen naar het enige kind van de te verwijderen knoop. Indien we knoop P verwijderen, moet het linkerkind van knoop T nu wijzen naar knoop O. Op figuur 2.3 wordt dit gedemonstreerd.

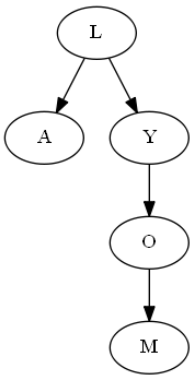


Figuur 2.3: Verwijderen van de knoop P die slechts één kind heeft.

3. *De knoop heeft twee kinderen*: In dit geval worden twee kinderen ouderloos. De te verwijderen knoop bevat echter zijn opvolger in de deelboom waarvan zijn rechterkind wortel van is. De opvolger is namelijk het kleinste element in deze deelboom. In het geval dat we knoop T verwijderen zal Y de ouder worden van O aangezien dit het kleinste (en ook enigste) element is in de deelboom, zoals op figuur 2.4 weergegeven.

Een binaire boom is niet altijd efficiënt. Indien de sleutels ingelezen worden in volgorde, zal de binaire boom een gelinkte lijst voorstellen, waardoor de operaties $O(n)$ worden i.p.v. $O(\lg n)$. Dit probleem werd opgelost door bomen die zichzelf zo evenwichtig mogelijk trachten te houden. Er zijn drie soorten van deze bomen:

1. **Rood-zwarte bomen**. Dit soort bomen tracht elke operatie steeds efficiënt te maken door ervoor te zorgen dat de structuur van de boom nagenoeg perfect blijft. Dit is de meest robuste methode.



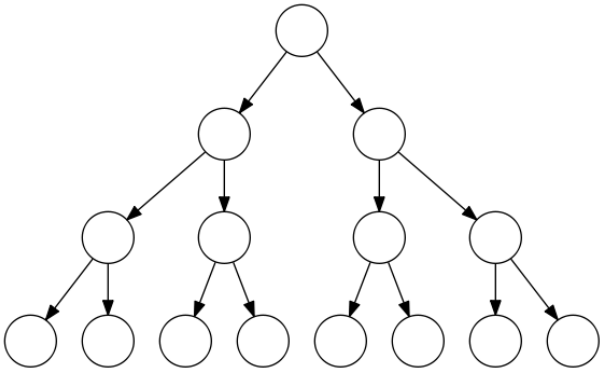
Figuur 2.4: Verwijderen van de knoop T die twee kinderen heeft.

2. **Splay trees**. Deze soort wordt de vorm van de boom meermaals aangepast, zodat de structuur van de boom nooit slecht is. Bij deze bomen is het gemiddelde geval steeds efficiënt (geamortiseerd), maar een individuele operatie kan soms slecht uitvallen.

3. **Randomized search trees** (treap). Deze boomstructuur zorgt ervoor dat de boom zo willekeurig mogelijk blijft, ongeacht de volgorde van toevoegen of verwijderen. Dit zorgt ervoor dat de verwachtingswaarde van elke operatie $O(\lg n)$ is.

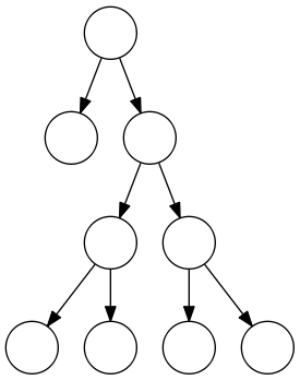
2.2 Rood-zwarte bomen

Gegeven Figuur 2.5, een complete binaire boom. Indien we telkens elementen zouden verwijderen uit de linkerdeelboom van de wortel, zal deze binaire boom niet meer in evenwicht zijn zoals te zien op figuur 2.6.



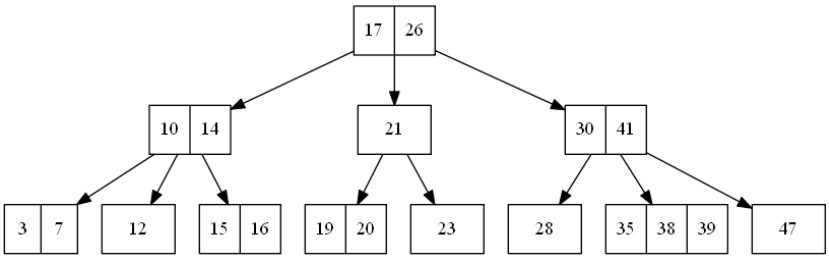
Figuur 2.5: Een complete binaire boom.

Een verbetering hierop is een 2-3-4 boom. Dit is een boom waarbij een knoop twee, drie of vier kinderen kan hebben waarbij alle bladeren even diep zitten. Een knoop kan één, twee of drie dataelementen bijhouden. Een voorbeeld van een dergelijke boom is te zien op figuur 2.7. De toevoeg

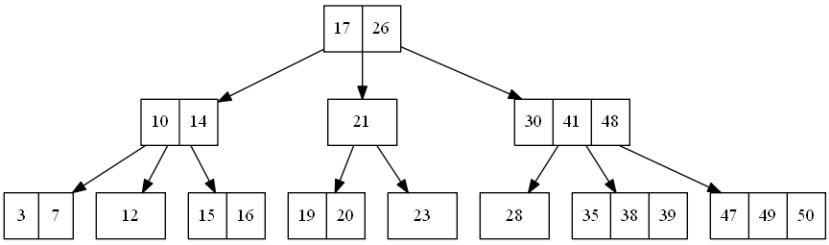


Figuur 2.6: Een onevenwichtige binaire boom.

operatie van de set {48, 49, 50} gebeurd als volgt: eerst wordt 48 toegevoegd aan het tweede data-element van de uiterst rechtse knoop. Het getal 49 kan geplaatst worden op het derde data-element. Op het moment dat 50 toegevoegd moet worden, kan er hiervoor geen nieuw kind aangemaakt worden aangezien alle bladeren op dezelfde diepte moet zitten. Daarom wordt de middelste knoop (48) naar de ouderknoop geplaatst, 49 op het tweede data-element en dan kan 50 uiteindelijk op het derde data-element komen. Figuur 2.8 toont de nieuwe 2-3-4 boom.

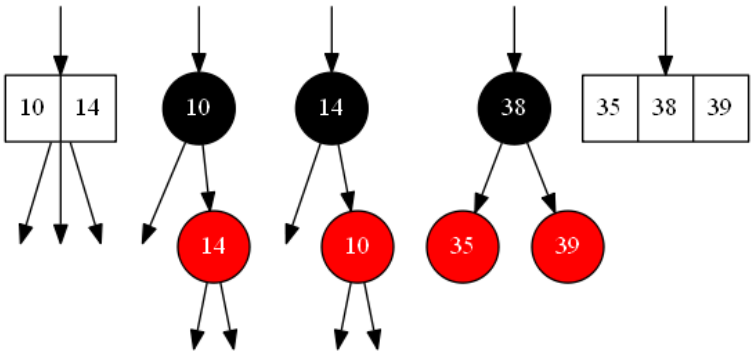


Figuur 2.7: Een 2-3-4 boom.



Figuur 2.8: Een 2-3-4 boom waaraan de elementen 48, 49 en 50 zijn toegevoegd.

Een rood-zwarte boom simuleert een 2-3-4 boom. Figuur 2.9 toont de conversie van 2 knopen van de 2-3-4 boom op figuur 2.7.



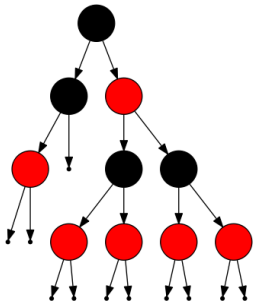
Figuur 2.9: Conversie 2-3-4 knopen naar rood-zwarte knopen.

2.2.1 Definitie rood-zwarte boom

Formeel kan een rood-zwarte zoekboom als volg gedefinieerd worden:

- Het is een binaire zoekboom.
- De knopen zijn rood of zwart (deze extra informatie kan bijgehouden worden met slechts één bit, aangezien er slechts twee kleuren zijn).
- Een rode knoop heeft *nooit* een rood kind.
- Elke virtuele knoop is zwart. Een virtuele knoop is een kind van een blad-element waarbij de virtuele knoop geen gegevens bevat.
- De zwarte hoogte z van een knoop k is hetzelfde voor elk pad naar een virtuele knoop startend vanaf k, met k niet meegerekend. De zwarte hoogte is het aantal zwarte knopen op dit pad.

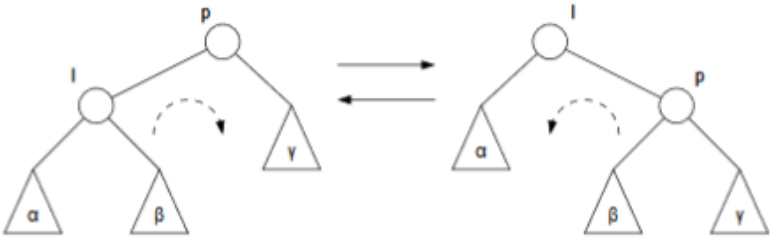
Door deze voorwaarden is het gegarandeerd dat de kleinste deelboom minstens half zo diep is als de grootste deelboom. Bekijk figuur 2.10, waarop een geldige rood-zwarte boom te zien is. De zwarte hoogte van de wortel is 2, want voor elk pad naar een virtuele knoop is het aantal zwarte knopen 2. Een deelboom met wortel w en zwarte hoogte z zal minstens $2^z - 1$ inwendige knopen bevatten.



Figuur 2.10: Een rood-zwarte boom.

2.2.2 Rotaties

Bij het toevoegen of verwijderen van een element kan een rood-zwarte boom niet meer aan de voorwaarden voldoen. Rotaties zal de vorm van de boom aanpassen, maar zal de inorder volgorde behouden. Een rotatie is O(1).

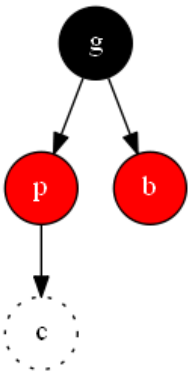


Figuur 2.11: Rotaties.

2.2.3 Bottom-up rood-zwarte bomen

- **Toevoegen.** Stel dat we een nieuwe knoop c willen toevoegen. Een nieuwe knoop krijgt altijd een rode kleur want de zwarte diepte herstellen is een ingewikkelder proces. Indien de ouderknoop p van c een zwarte kleur heeft, is er geen probleem. Is de kleur van p rood, dan zijn er zes gevallen mgoelijk, waarvan 3 het spiegelbeeld zijn van elkaar (afhankelijk of dat c het linkerkind of rechterkind is van p.

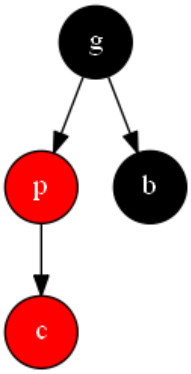
  1. *De broer b van p is rood.* In dit geval kan g rood gemaakt worden. De knopen p en b



Figuur 2.12: Rode broer.

  kunnen op hun beurt dan zwart gemaakt worden. Indien g een zwarte ouder heeft is de toevoegoperatie gelukt. Is de kleur van de ouder van g rood, dan wordt het probleem opgeschoven waarbij g nu de rol neemt van c.

  2. *De broer b van p is zwart.* In dit geval wordt er nog een onderscheidt gemaakt tussen de positie van c.



Figuur 2.13: Zwarte broer.

(a) *Knoop c is een linkerkind van p.* Roteer p en g naar rechts, maak p zwart en g rood. Het probleem is opgelost.

(b) *Knoop c is een rechterkind van p* Roteer p en c naar links, zodat vorige situatie bekomen wordt.

- **Verwijderen.**

2.2.4 Top-down rood-zwarte bomen

2.3 Splay trees

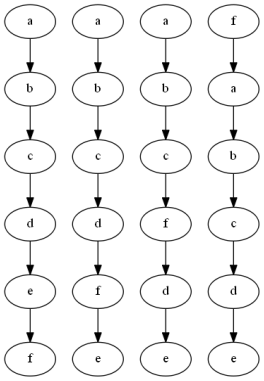
Splay trees zullen ervoor zorgen dat een reeks van operaties efficiënt is. Het kan dus voorkomen dat een individuele operatie traag uitvalt, maar dat de geamortiseerde performantie goed is. Een splay tree is eigenlijk een normale binaire boom waarbij één extra functie wordt toegevoegd, de splay-functie. Indien een knoop gezocht en gevonden wordt, zal de splay-functie ervoor zorgen dat deze knoop de wortel wordt van de boom. Dit kan een boom die de vorm van een gelinkte lijst heeft stelselmatig ombouwen tot een goede binaire boom. Een naïeve methode is echter om telkens de knoop te roteren met zijn ouder. Figuur 2.14 toont het resultaat van deze operatie. In de linkse boom wordt de knoop F gezocht. Na het zoeken wordt de splay operatie uitgevoerd die altijd de knoop F zal roteren met zijn ouder. Het eindresultaat is terug een gelinkte lijst. Een beter methode zou zijn om eerste E met D te roteren, dan F met E. De volgende stap roteert C met B, gevolgd door de rotatie F met C.

2.3.1 Bottom-up splay trees

2.3.2 Top-down splay trees

2.3.3 Performantie van splay trees

Omdat de vorm van de boom voortdurend verandert, wordt in het bewijs gebruik gemaakt van een potentiaalfunctie Φ. Elke vorm van de boom krijgt een getal dat het potentiaal voorstelt. Een goede operatie zal het potentiaal verhogen terwijl een slechte operatie deze zal laten dalen. Stel a_i de



Figuur 2.14: Naïeve splay-functie.

geamortiseerde tijd van een operatie, t_i de werkelijke tijd van de operatie, Φ_i het potentiaal na de operatie en Φ_{i-1} het potentiaal voor de operatie. a_i kan beschreven worden als:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Een reeks van operaties is de som van alle a_i:

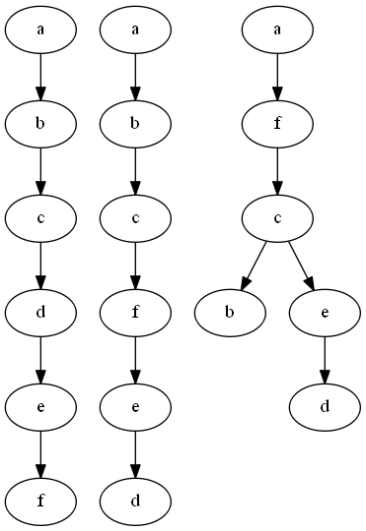
$$\begin{aligned}\sum_{i=1}^{m} a_i &= \sum_{i=1}^{m} (t_i + \Phi_i - \Phi_{i-1}) \\ &= \sum_{i=1}^{m} (t_i + \Phi_m - \Phi_0)\end{aligned}$$

hierbij is Φ_0 de beginpotentiaal. De potentiaalfunctie moet zo gekozen worden zodat het eindpotentiaal niet kleiner is als het beginpotentiaal. Doorgaans wordt de potentiaalfunctie

$$\Phi = \sum_{i=1}^{m} \lg s_i$$

genomen. Hier is s_i het aantal knopen in de deelboom waar knoop i wortel van is. $\lg s_i$ wordt dan de rang van knoop i genoemd, of r_i. Om de geamortiseerde tijd te bewijzen wordt een knoop c beschouwd met als rang r_c in de originele boom, die op een bottom-up manier wordt toegevoegd. De analyse is analoog voor een top-down methode.

Indien c de wortel is, dan moet er niet afgedaald worden en is de werkelijke tijd O(1), wat niets verandert aan het potentiaal.



Figuur 2.15: Goede splay functie.