

Gevorderde algoritmen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 24 juni 2019

Inhoudsopgave

I	Gegevensstructuren II	2
1	Efficiënte zoekbomen	3
1.1	Inleiding	3
1.2	Rood-zwarte bomen	4
1.2.1	Definitie en eigenschappen	4
1.2.2	Zoeken	5
1.2.3	Toevoegen en verwijderen	5
1.2.4	Rotaties	5
1.2.5	Bottom-up rood-zwarte bomen	6
1.2.6	Top-down rood-zwarte bomen	6
1.3	Splaybomen	6
1.4	Gerandomiseerde zoekbomen	6
2	Toepassingen van dynamisch programmeren	7
2.1	Optimale binaire zoekbomen	7
2.2	Langste gemeenschappelijke deelsequentie	10

Deel I

Gegevensstructuren II

Hoofdstuk 1

Efficiënte zoekbomen

1.1 Inleiding

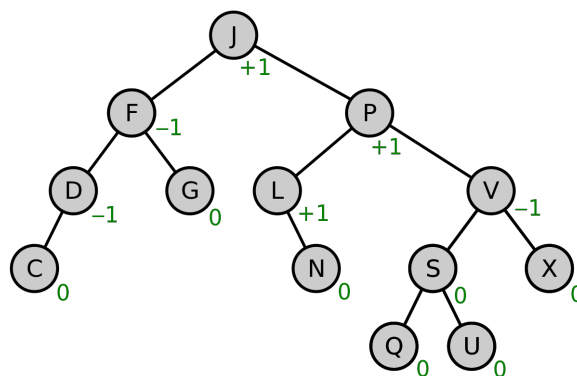
- Uitvoeringstijd van operaties op een binaire zoekboom met hoogte h is $O(h)$.
- De hoogte h is afhankelijk van de toevoegvolgorde:
 - In het slechtste geval bekommt men een gelinkte lijst, zodat $h = O(n)$.
 - Als elke toevoegvolgorde even waarschijnlijk is, dan is de verwachtingswaarde voor de hoogte $h = O(\lg n)$ met n het aantal gegevens.

! Geen realistische veronderstelling.
- Drie manieren om de efficiëntie van zoekbomen te verbeteren:

1. **Elke operatie steeds efficiënt maken.**

(a) AVL-bomen.

- Hoogteverschil Δh van de twee deelbomen van elke knoop is kleiner of gelijk aan 1.
- Δh wordt opgeslagen in de knoop zelf.



Figuur 1.1: Een AVL-boom. De groene cijfers stellen de hoogteverschillen voor van de twee deelbomen voor elke knoop.

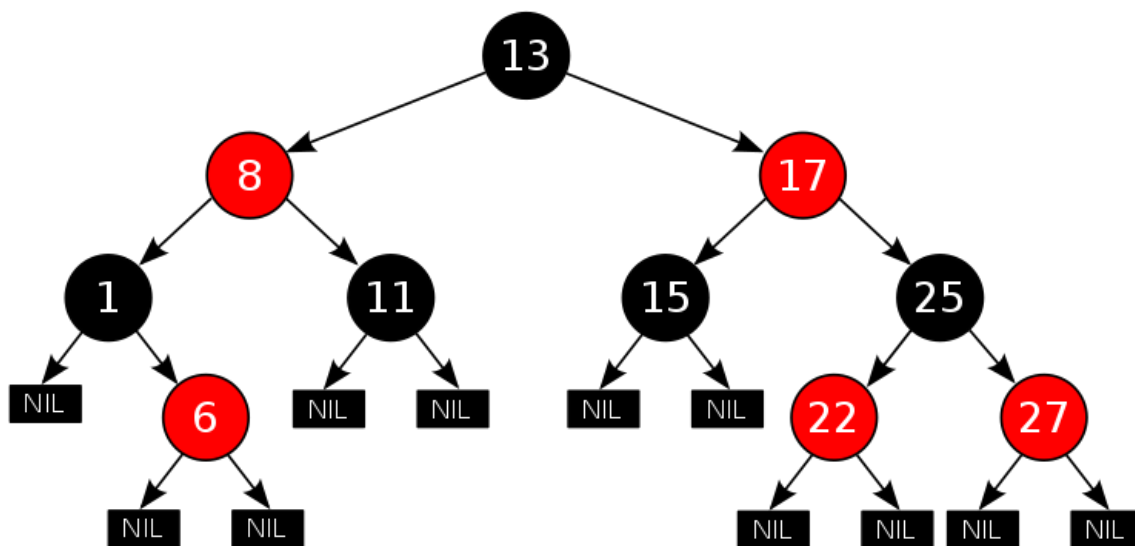
(b) 2-3-bomen.

- Elke knoop heeft 2 of 3 kinderen.

- Elk blad heeft dezelfde diepte.
 - Bij toevoegen of verwijderen wordt het ideale evenwicht behouden door het aantal kinderen van de knopen te manipuleren.
- (c) 2-3-4-bomen.
- Eenvoudiger dan 2-3-bomen om te implementeren.
- (d) Rood-zwarte bomen (sectie 1.2.1).
2. **Elke reeks operaties steeds efficiënt maken.**
- (a) Splaybomen (sectie 1.3).
- De vorm van de boom wordt meermaals aangepast.
 - Elke reeks opeenvolgende operaties is gegarandeerd efficiënt.
 - Een individuele operatie kan wel traag uitvallen.
 - *Geamortiseerd* is de prestatie per operatie goed.
3. **De gemiddelde efficiëntie onafhankelijk maken van de operatievolgorde.**
- (a) Gerandomiseerde zoekbomen (sectie 1.4).
- Gebruik van een random generator.
 - De boom is random, onafhankelijk van de toevoeg- en verwijdervolgorde.
 - Verwachtingswaarde van de hoogte h wordt dan $O(\lg n)$.

1.2 Rood-zwarte bomen

1.2.1 Definitie en eigenschappen



Figuur 1.2: Een rood-zwarte boom. De NIL knopen stellen virtuele knopen voor.

• **Definitie:**

- Een binaire zoekboom.
- Elke knoop is rood of zwart gekleurd.
- Elke virtuele knoop is zwart. Een virtuele knoop is een knoop die geen waarde heeft, maar wel een kleur. Deze vervangen de nullwijzers.

- Een rode knoop heeft steeds twee zwarte kinderen
- Elke mogelijke weg vanuit een knoop naar een virtuele knoop bevat evenveel zwarte knopen. Dit aantal wordt de *zwarte hoogte* genoemd
- De wortel is zwart.

- **Eigenschappen:**

- Een deelboom met wortel w en zwarte hoogte z heeft tenminste $2^z - 1$ inwendige knopen.
- De hoogte van een rood-zwarte boom met n knopen is steeds $O(\lg n)$.
 - ◊ Er zijn nooit twee opeenvolgende rode knopen op elke weg vanuit een knoop naar een virtuele knoop $\rightarrow z \geq h/2$.
 - ◊ Substitutie in de eerste eigenschap geeft:

$$\begin{aligned} n &\geq 2^z - 1 \geq 2^{h/2} - 1 \\ \rightarrow h &\leq 2 \lg(n + 1) \end{aligned}$$

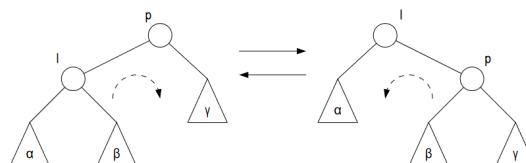
1.2.2 Zoeken

- De kleur speelt geen rol, zodat de rood-zwarte boom een gewone binaire zoekboom wordt.
- De hoogte is wel geëgreandeerd $O(\lg n)$.
- Zoeken naar een willekeurige sleutel is dus $O(\lg n)$.

1.2.3 Toevoegen en verwijderen

- Element toevoegen of verwijderen, zonder rekening te houden met de kleur, is ook $O(\lg n)$.
- ! Geen garantie dat deze gewijzigde boom nog rood-zwart zal zijn.
- Twee manieren om toe te voegen:
 1. **Bottom-up:**
 - Voeg knoop toe zonder rekening te houden met de kleur.
 - Herstel de rood-zwarte boom, te beginnen bij de nieuwe knoop, en desnoods tot bij de wortel.
 2. **Top-down:**
 - Pas de boom aan langs de dalende zoekweg.
 - ✓ Efficiënter dan bottom-down aangezien er geen ouderwijzers of een stapel nodig is.

1.2.4 Rotaties



Figuur 1.3: Rotaties

- Wijzigen de vorm van de boom, maar behouden de inorder volgorde van de sleutels.
- Moet enkel pointers aanpassen, en is dus $O(1)$.
- **Rechtste rotatie** van een ouder p en zijn linkerkind l :
 - Het rechterkind van l wordt het linkerkind van p .
 - De ouder van p wordt de ouder van l .
 - p wordt het rechterkind van l .
- **Linkse rotatie** van een ouder p en zijn rechterkind r :
 - Het linkerkind van r wordt het rechterkind van p .
 - De ouder van r wordt de ouder van p .
 - p wordt het linkerkind van l .

1.2.5 Bottom-up rood-zwarte bomen

~~ToDo:~~ vanaf hier

1.2.6 Top-down rood-zwarte bomen

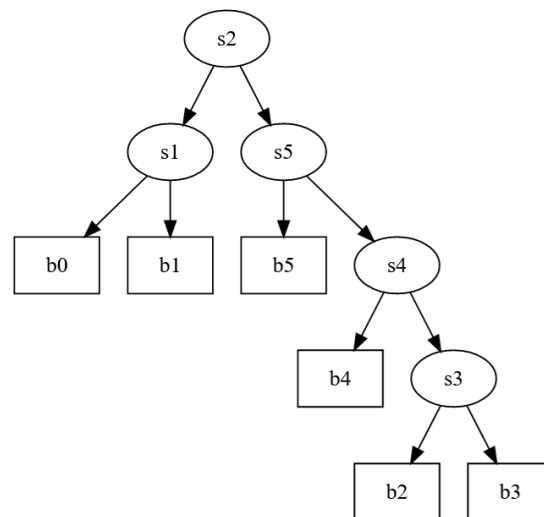
1.3 Splaybomen

1.4 Gerandomiseerde zoekbomen

Hoofdstuk 2

Toepassingen van dynamisch programmeren

2.1 Optimale binaire zoekbomen



Figuur 2.1: Een optimale binaire zoekboom met bijhorende kansentabel.

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

- Veronderstel dat de gegevens die in een binaire zoekboom moeten opgeslaan worden op voorhand gekend zijn.
- Veronderstel ook dat de waarschijnlijkheid gekend is waarmee de gegevens gezocht zullen worden.
- Tracht de boom zo te schikken zodat de verwachtingswaarde van de zoektijd minimaal wordt.

- De zoektijd wordt bepaald door de lengte van de zoekweg.
- De gerangschikte sleutels van de n gegevens zijn s_1, \dots, s_n .
- De $n + 1$ bladeren zijn b_0, \dots, b_n .
 - Elk blad staat voor een afwezig gegeven die in een deelboom op die plaats had moeten zitten.
 - Het blad b_0 staat voor alle sleutels kleiner dan s_1 .
 - Het blad b_n staat voor alle sleutels groter dan s_n .
 - Het blad b_i staat voor alle sleutels groter dan s_i en kleiner dan s_{i+1} , met $1 \leq i < n$
- De waarschijnlijkheid om de i -de sleutel s_i te zoeken is p_i .
- De waarschijnlijkheid om alle afwezige sleutels, voorgesteld door een blad b_i , te zoeken is q_i .
- De som van alle waarschijnlijkheden moet 1 zijn:

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$$

- Als de zoektijd gelijk is aan het aantal knopen op de zoekweg (de diepte plus één), dan is de verwachtingswaarde van de zoektijd

$$\sum_{i=1}^n p_i(\text{diepte}(s_i) + 1) + \sum_{i=0}^n q_i(\text{diepte}(b_i) + 1)$$

- Deze verwachtingswaarde moet geminimaliseerd worden.
 - Boom met minimale hoogte is niet voldoende.
 - Alle mogelijke zoekbomen onderzoeken is ook geen optie, hun aantal is

$$\begin{aligned} \frac{1}{n+1} \binom{2n}{n} &\sim \frac{1}{n+1} \cdot \frac{2^{2n}}{\sqrt{\pi n}} \\ &\sim \frac{1}{n+1} \cdot \frac{4^n}{\sqrt{\pi n}} \\ &\sim \Omega\left(\frac{4^n}{n\sqrt{n}}\right) \end{aligned}$$

✓ Dynamisch programmeren biedt een uitkomst.

- Een optimalisatieprobleem komt in aanmerkingen voor een efficiënte oplossing via dynamisch programmeren als het:
 1. het een **optimale deelstructuur** heeft;
 2. de **deelproblemen onafhankelijk maar overlappend** zijn.
- Is dit hier van toepassing?
 - Is er een optimale deelstructuur?
 - ◊ Als een zoekboom optimaal is, moeten zijn deelbomen ook optimaal zijn.
 - ◊ Een optimale oplossing bestaat uit optimale oplossingen voor deelproblemen.
 - Zijn de deelproblemen onafhankelijk?
 - ◊ Ja want deelbomen hebben geen gemeenschappelijke knopen.

- Zijn de deelproblemen overlappend?
 - ◊ Elke deelboom bevat een reeks opeenvolgende sleutels s_i, \dots, s_j met bijhorende bladeren b_{i-1}, \dots, b_j .
 - ◊ Deze deelboom heeft een wortel s_w waarbij $(i \leq w \leq j)$.
 - ◊ De linkse deelboom bevat de sleutels s_i, \dots, s_{w-1} en bladeren b_{i-1}, \dots, b_{w-1} .
 - ◊ De rechtse deelboom bevat de sleutels s_{w+1}, \dots, s_j en bladeren b_w, \dots, b_j .
 - ◊ Voor een optimale deelboom met wortel s_w moeten deze beide deelbomen ook optimaal zijn.
 - ◊ Deze wordt gevonden door:
 1. achtereenvolgens elk van zijn sleutels s_i, \dots, s_j als wortel te kiezen;
 2. de zoektijd voor de boom te berekenen door gebruikte maken van de zoektijden van de optimale deelbomen;
 3. de wortel te kiezen die de kleinste zoektijd oplevert.
- We willen dus de kleinste verwachte zoektijd $z(i, j)$.
- Dit moet gebeuren voor alle i en j waarbij:
 - $1 \leq i \leq n+1$
 - $0 \leq j \leq n$
 - $j \geq i-1$
- De optimale boom heeft dus de kleinste verwachte zoektijd $z(1, n)$.
- Hoe $z_w(i, j)$ bepalen voor een deelboom met wortel s_w ?
 - Gebruik de kans om in de wortel te komen.
 - Gebruik de optimale zoektijden van zijn deelbomen, $z(i, w-1)$ en $z(w+1, j)$.
 - Gebruik ook de diepte van elke knoop, maar elke knoop staat nu op een niveau lager.
 - ◊ De bijdrage tot de zoektijd neemt toe met de som van de zoekwaarschijnlijkheden.

$$g(i, j) = \sum_{k=i}^j p_k + \sum_{k=i-1}^j q_k$$

- Hieruit volgt:

$$\begin{aligned} z_w(i, j) &= p_w + (z(i, w-1) + g(i, w-1)) + (z(w+1, j) + g(w+1, j)) \\ &= z(i, w-1) + z(w+1, j) + g(i, j) \end{aligned}$$

- Dit moet minimaal zijn \rightarrow achtereenvolgens elke sleutel van de deelboom tot wortel maken.
 - De index w doorloopt alle waarden tussen i en j .

$$\begin{aligned} z(i, j) &= \min_{i \leq w \leq j} \{z_w(i, j)\} \\ &= \min_{i \leq w \leq j} \{z(i, w-1) + z(w+1, j) + g(i, j)\} \end{aligned}$$

- Hoe wordt de optimale boom bijgehouden?
 - Hou enkel de index w bij van de wortel van elke optimale deelboom.
 - Voor de deelboom met sleutels s_i, \dots, s_j is de index $w = r(i, j)$.
- Implementatie:

- Een recursieve implementatie zou veel deeloplossingen opnieuw berekenen.
- Daarom **bottom-up** implementeren. Maakt gebruik van drie tabellen:
 1. Tweedimensionale tabel $z[1..n+1, 0..n]$ voor de waarden $z(i, j)$.
 2. Tweedimensionale tabel $g[1..n+1, 0..n]$ voor de waarden $g(i, j)$.
 3. Tweedimensionale tabel $r[1..n, 1..n]$ voor de indices $r(i, j)$.
- Algoritme:
 1. Initialiseer de waarden $z(i, i-1)$ en $g(i, i-1)$ op $q[i-1]$.
 2. Bepaal achtereenvolgens elementen op elke evenwijdige diagonaal, in de richting van de tabelhoek linksboven.
 - ◇ Voor $z(i, j)$ zijn de waarden $z(i, i-1), z(i, i), \dots, z(i, j-1)$ van de linkse deelboom nodig en de waarden $z(i+1, j), \dots, z(j, j), z(j+1, j)$ van de rechtse deelboom nodig.
 - ◇ Deze waarden staan op diagonalen onder deze van $z(i, j)$.
- Efficiëntie:
 - **Bovengrens:** drie verneste lussen $\rightarrow O(n^3)$.
 - **Ondergrens:**
 - ◇ Meeste werk bevindt zich in de binneste lus.
 - ◇ Een deelboom met sleutels s_i, \dots, s_j heeft $j-i+1$ mogelijke wortels.
 - ◇ Elke test is $O(1)$.
 - ◇ Dit werk is evenredig met

$$\begin{aligned}
 & \sum_{i=1}^n \sum_{j=i}^n (j-i+1) \\
 \Rightarrow & \sum_{i=1}^n i^2 \\
 \Rightarrow & \frac{n(n+1)(2n+1)}{6} \\
 \Rightarrow & \Omega(n^3)
 \end{aligned}$$

- **Algemeen:** $\Theta(n^3)$.
- Kan met een bijkomende eigenschap (zien we niet in de cursus) gereduceerd worden tot $\Theta(n^2)$.

2.2 Langste gemeenschappelijke deelsequentie

- Een deelsequentie van een string wordt bekomen door nul of meer stringelementen weg te laten.
- Elke deelstring is een deelsequentie, maar niet omgekeerd.
- Een langste gemeenschappelijke deelsequentie (LGD) van twee strings kan nagaan hoe goed deze twee strings op elkaar lijken.
- Geven twee strings:
 - $X = \langle x_0, x_1, \dots, x_{n-1} \rangle$
 - $Y = \langle y_0, y_1, \dots, y_{m-1} \rangle$
- Hoe LGD bepalen?

- Is er een optimale deelstructuur?
 - ◊ Een optimale oplossing maakt gebruik van optimale oplossingen voor deelproblemen.
 - ◊ De deelproblemen zijn paren prefixen van de twee strings.
 - ◊ Het prefix van X met lengte i is X_i .
 - ◊ Het prefix van Y met lengte j is Y_j .
 - ◊ De ledige prefix is X_0 en Y_0 .
- Zijn de deelproblemen onafhankelijk?
 - ◊ Stel $Z = \langle z_0, z_1, \dots, z_{k-1} \rangle$ de LGD van X en Y . Er zijn drie mogelijkheden:
 1. Als $n = 0$ of $m = 0$ dan is $k = 0$.
 2. Als $x_{n-1} = y_{m-1}$ dan is $z_{k-1} = x_{n-1} = y_{m-1}$ en is Z een LGD van X_{n-1} en Y_{m-1} .
 3. Als $x_{n-1} \neq y_{m-1}$ dan is Z een LGD van X_{n-1} en Y of een LGD van X en Y_{m-1} .
- Zijn de deelproblemen overlappend?
 - ◊ Om de LGD van X en Y te vinden is het nodig om de LGD van X en Y_{m-1} als van X_{n-1} en Y te vinden.
- LGD kan bepaald worden door recursieve vergelijking:

$$c[i, j] = \begin{cases} 0 & \text{als } i = 0 \text{ of } j = 0 \\ c[i-1, j-1] + 1 & \text{als } i > 0 \text{ en } j > 0 \text{ en } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{als } i > 0 \text{ en } j > 0 \text{ en } x_i \neq y_j \end{cases}$$

- De lengte van de LGD komt overeen met $c[n, m]$.
- De waarden $c[i, j]$ kunnen eenvoudig per rij van links naar rechts bepaald worden door de recursierelatie.
- Efficiëntie:
 - We beginnen de tabel in te vullen vanaf $c[1, 1]$ (als $i = 0$ of $j = 0$ zijn de waarden 0).
 - De tabel c wordt rij per rij, kolom per kolom ingevuld.
 - De vereiste plaats en totale performantie is beiden $\Theta(nm)$.