FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Gevorderde algoritmen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 29 september 2019

UNIVERSITEIT
GENT

Inhoudsopgave

Deel I

Gegevensstructuren II

Hoofdstuk 1

Efficiënte zoekbomen

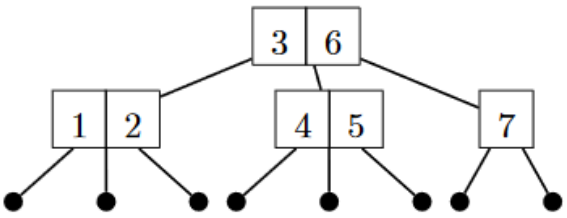
1.1 Inleiding

- Uitvoeringstijd van operaties (zoeken, toevoegen, verwijderen) op een binaire zoekboom met hoogte h is $O(h)$.
- De hoogte h is afhankelijk van de toevoegvolgorde van de n elementen:
  - In het slechtste geval bekomt men een gelinkte lijst, zodat $h = O(n)$.
  - Als elke toevoegvolgorde even waarschijnlijk is, dan is de verwachtingswaarde voor de hoogte $h = O(\lg n)$.
    - ! Geen realistische veronderstelling.
- Drie manieren om de efficiëntie van zoekbomen te verbeteren:
  1. **Elke operatie steeds efficiënt maken.** (Hoogte klein houden)
     (a) AVL-bomen.
     - Hoogteverschil van de tweede deelbomen van elke knoop wordt gedefinieerd als:

$$\Delta h <= 1$$

     - Δh wordt opgeslagen in de knoop zelf.

     (b) 2-3-bomen (figuur 1.1).



Figuur 1.1: Een 2-3-boom.

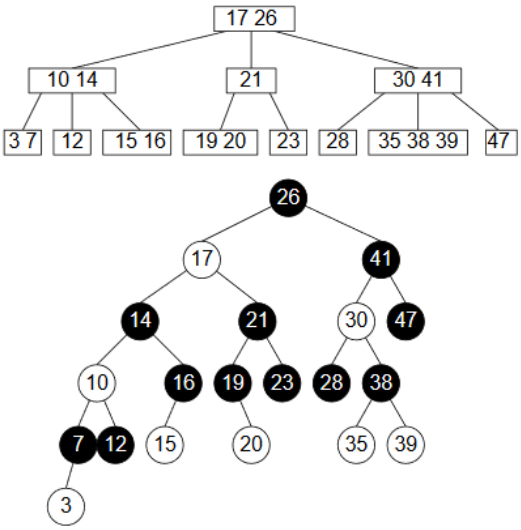
- Elke knoop heeft 2 of 3 kinderen en dus 1 of 2 sleutels.
- Elk blad heeft dezelfde diepte.
- Bij toevoegen of verwijderen wordt het ideale evenwicht behouden door het aantal kinderen van de knopen te manipuleren.

(c) 2-3-4-bomen.
- Analoog aan een 2-3-boom, maar elke knoop heeft 2, 3 of 4 kinderen.
- ! Per knoop moet er plaats voorzien zijn voor 3 sleutels, wat onnodig veel geheugen vraagt.

(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 performantie 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



Figuur 1.2: Een 2-3-4-boom en equivalent rood-zwarte boom (wit stelt hier rood voor).

- Simuleert een 2-3-4-boom (fig 1.2).
  - Een knoop in een 2-3-4 boom worden 1, 2 of 3 knopen in een rood-zwarte boom.
  - Een 2-knoop wordt een zwarte knoop.
  - Een 3-knoop wordt een zwarte knoop met een rood kind.
  - Een 4-knoop wordt een zwarte knoop met twee rode kinderen.
- Een rood-zwarte boom is gemakkelijker te definieren als er afgestapt wordt van het 2-3-4-boom concept.

1.2.1 Definitie en eigenschappen

- **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 elke virtuele knoop bevat evenveel zwarte knopen. Dit aantal zwarte knoepn wordt de *zwarte hoogte* genoemd
  - De wortel is zwart.
- **Eigenschappen:**
  - Een deelboom met zwarte hoogte z heeft tenminste $2^z - 1$ inwendige knopen. Dit is de deelboom waarvan elke knoop zwart is.
  - De hoogte h van een rood-zwarte boom met n knopen is steeds $O(\lg n)$, want:
    - 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:

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

1.2.2 Zoeken

- De kleur speelt geen rol, zodat de rood-zwarte boom een gewone binaire zoekboom wordt.
- De hoogte van een rood-zwarte boom is wel gegerandeerd $O(\lg n)$.
- 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.

   ! Er zijn ouderwijzers of een stapel nodig om naar boven in de boom te gaan.

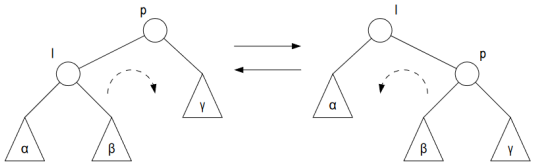
   ! Multithreading is niet mogelijk. Alle threads die een bottom-up rood-zwarte boom gebruiken moeten gelocked worden bij een toevoeg-of verwijderoperatie.
2. **Top-down:**
   - Pas de boom aan langs de dalende zoekweg.

! Als de ouder van de toe te voegen knoop reeds zwart is, dan moet er niets aan de boom aangepast worden (want elke nieuwe knoop is altijd rood). Top-down houdt hier geen rekening mee en heeft toch reeds de boom aangepast tegen dat de knoop toegevoegd wordt.

✓ Geen ouderwijzers of stapel nodig.

✓ Multithreading wel mogelijk. Bij het afdalen naar elke knoop zijn enkel nog de deelbomen van die knoop nodig om de boom te herstellen.

1.2.4 Rotaties



Figuur 1.3: Rotaties

- Een rotatie wijzigt de vorm van de boom, maar behouden de inorder volgorde van de sleutels.
- Er moeten enkel pointers aangepast worden, 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.

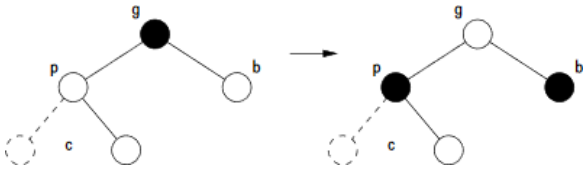
1.2.5 Bottom-up rood-zwarte bomen

Toevoegen

- De knoop wordt eerst op de gewone manier toegevoegd.
- Welke kleur geven we die knoop?
  - **Zwart:** dit kan de zwarte hoogte van veel knopen ontregelen.
  - **Rood:** dit mag enkel als de ouder zwart is.
  - Kies voor rood omdat zwarte hoogte moeilijker te herstellen valt.
- Als de ouder zwart is, dan is toevoegen gelukt.
- Als de ouder rood is wordt deze storing verwijderd door rotaties en kleurwijzigingen door te voeren.

- Vaststellingen:
  - De ouder p van de nieuwe knoop c is rood.
  - De grootouder g van c is zwart want p is rood.
- Er zijn zes mogelijke gevallen, die twee groepen van drie vormen, naar gelang dat p een linker- of rechterkind is van g.
- We onderstellen dat p een linkerkind is van g.

1. **De broer b van p is rood.**

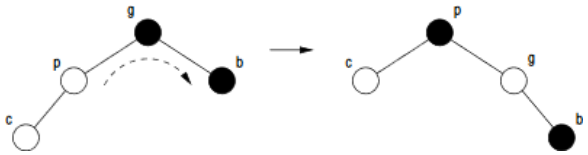


Figuur 1.4: Rode broer.

   - Maak p en b zwart.
   - Maak g rood.
   - Als g een zwarte ouder heeft, is het probleem opgelost.
   - Als g een rode ouder heeft, zijn er opnieuw twee opeenvolgende rode knopen.
   - Het probleem wordt opgeschoven in de richting van de wortel.

2. **De broer p van p is zwart.**

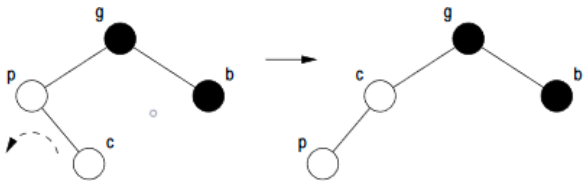
   (a) **Knoop c is een linkerkind van p.**



Figuur 1.5: Rode broer.

   - Roteer p en g naar rechts.
   - Maak p zwart.
   - Maak g rood.

   (b) **Knoop c is een rechterkind van p.**



Figuur 1.6: Rode broer.

   - Roteer p en c naar links.
   - We krijgen nu het vorige geval.

- Hoogstens 2 rotaties om de boom te herstellen, voorafgegaan door eventueel $O(\lg n)$ opschuivingen.
- Roteren en opschuiven is $O(1)$, en afdalen is $O(\lg n)$ zodat toevoegen $O(\lg n)$ is.

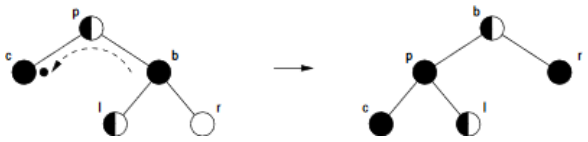
Verwijderen

- Verwijder eerst de knoop zoals bij een gewone zoekboom.
- Als de te verwijderen knoop rood is, is er geen gevolg voor de zwarte hoogte en is de operatie klaar.
- Als de te verwijderen knoop zwart is, zijn er twee mogelijkheden:
  1. **De knoop heeft minstens één rood kind.** Dit rood kind kan de zwarte kleur overnemen, zodat de zwarte hoogten intact blijven.
  2. **De knoop heeft twee zwarte kinderen (virtueel of echt).** De zwarte kleur wordt aan één van de kinderen gegeven, zodat die **dubbelzwart** wordt.
- In het eerste geval is de operatie klaar. In het tweede geval moet de boom, die nu een dubbelzwarte knoop bevat, herstelt worden.
- Als de dubbelzwarte knoop c de wortel is, kan deze extra zwarte kleur verdwijnen.
- Als c geen wortel is, en ouder p heeft, dan zijn er acht mogelijkheden die in groepen van twee uiteenvallen naargelang c een linker- of rechterkind van p is.
- We veronderstellen dat c een linkerkind is van p.
  1. **De broer b van c is zwart.** De kleur van p is willekeurig. Hier zijn er drie gevallen mogelijk, afhankelijk van de kleur van de kinderen van b.
     (a) **Broer b heeft twee zwarte kinderen.**

Figuur 1.7

- Knoop b kan rood worden.
- De extra zwarte kleur van c kan aan p gegeven worden.
  - Als p rood was, dan is de operatie gelukt.
  - Als p reeds zwart was, dan verschuift het probleem zich naar boven.

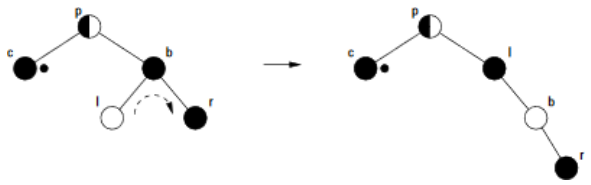
     (b) **Broer b heeft een rood rechterkind.** De kleur van het linkerkind l van b is willekeurig.



Figuur 1.8

- Roteer p en b naar links.
- Knoop p krijgt de extra zwarte kleur van c.
- Het rechterkind r van b wordt zwart.
- Knoop b krijgt de oorspronkelijke kleur van p.

     (c) **Broer b heeft een zwarte rechterkind en een rood linkerkind.**
- Roteer b en l naar rechts.



Figuur 1.9

- Maak b rood en l zwart.
- Dit is nu het vorige geval.

2. **De broer b van c is rood.**

Figuur 1.10

- Roteer p en b naar links.
- Maak b zwart en p rood.
- Dit is nu het eerste geval.

- Hoogstens 3 rotaties nodig om de boom te herstellen, voorafgegaan door eventueel $O(\lg n)$ opschuivingen.
- Roteren en opschuiven is $O(1)$, en afdalen is $O(\lg n)$ zodat verwijderen $O(\lg n)$ is.

1.2.6 Top-down rood-zwarte bomen

Toevoegen

- Ook hier worden nieuwe knopen rood gemaakt.
- Op de weg naar beneden mogen er geen rode broers zijn.
- Als we een **zwarte knoop met twee rode kinderen** tegenkomen, dan maken we die knoop rood en zijn kinderen zwart.
- Als zijn ouder rood is, kan dit met rotaties en kleurwijzigingen opgelost worden.
- Toevoegen daalt enkel in de boom en is $O(\lg n)$.

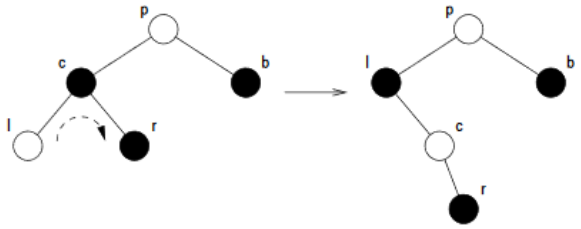
Verwijderen

- De zwarte hoogte van de fysisch te verwijderen knoop is één, omdat minstens één van zijn kinderen virtueel is.
- Om geen problemen te krijgen met de zwarte hoogte moet deze knoop rood zijn, maar dan moet zijn tweede kind ook virtueel zijn.
- De zoekknoop kan eender waar in de boom zitten, daarom wordt elke bolgende knoop op de zoekweg rood gemaakt.

- Tijdens het afdalen komen we in een rode of rood gemaakte knoop p.
- Die heeft dan zeker een zwart kind c, dat rood moet worden.
- Er zijn acht mogelijkheden die in groepen van twee uiteenvallen naargelang c een linker- of rechterkind van p is.

- We veronderstellen dat c een linkerkind is van p.

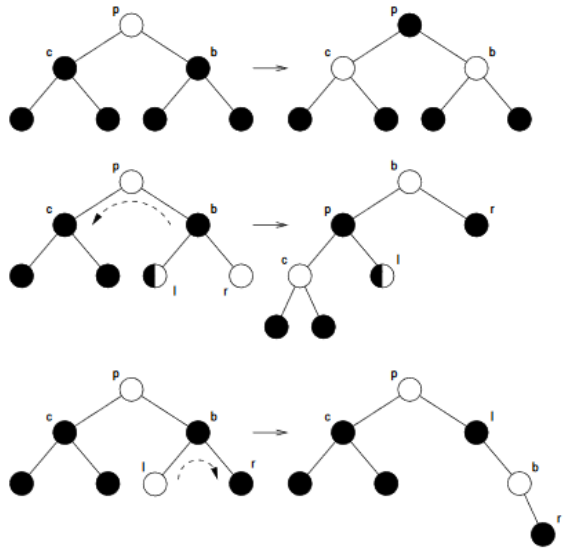
1. **Knoop c heeft minstens één rood kind.**



Figuur 1.11

- Als we naar een rode knoop moeten afdalen zitten we terug in de beginsituatie.
- Als c de fysisch te verwijderen knoop is of als we naar een zwarte knoop moeten afdalen:
  - Roteer c samen met zijn rood kind zodat c nu als ouder zijn oorspronkelijk rood kind heeft.
  - Wijzig de kleur van c naar zwart.
  - Wijzig de kleur van zijn oorspronkelijk kind naar rood.

2. **Knoop c heeft twee zwarte kinderen.**



Figuur 1.12

1.2.7 Vereenvoudigde rood-zwarte bomen

- De implementatie is omslachtig door de talrijke speciale gevallen.

- Eenvoudigere varianten bestaan:
  - Een **AA-boom** geeft aan dat enkel een rechterkind rood moet zijn.
  - Een **Binary B-tree** beperkt het aantal gevallen maar behouden toch de asymptotische efficiëntie.
  - Een **left-leaning red-black-tree** stelt de eis dat een zwarte knoop enkel een rood rechterkind mag hebben als het reeds een rood linkerkind heeft.

1.3 Splaybomen

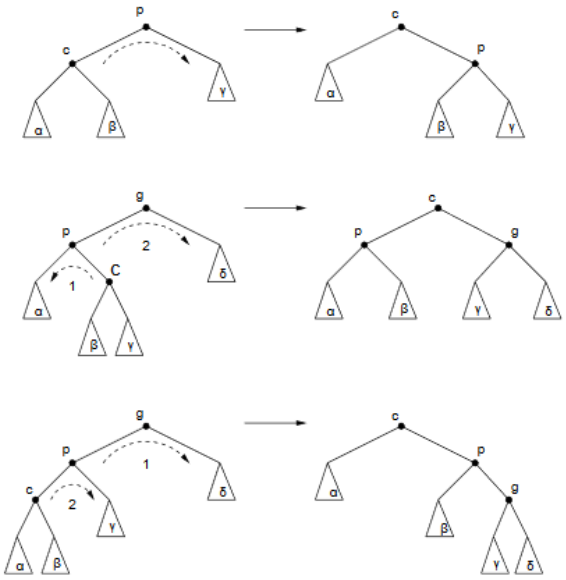
- Garanderen dat elke reeks opeenvolgende operaties efficiënt is.
- Als we m operaties verrichten op de splay tree, waarbij n keer toevoegen, dan is de performantie van deze reeks $O(m \lg n)$.
- Uitgemiddeld is dit $O(\lg n)$.
- Individuelde operaties mogen inefficiënt zijn, maar de boom moet zo aangepast worden zodat een reeks van die operaties efficiënt zijn.
- **Basisidee:** Elke knoop die gezocht wordt, toegevoegd of verwijdert wordt, zal de wortel worden van de boom, zodat opeenvolgende operaties op die knoop efficiënt zijn.
- Een willekeurige knoop tot wortel maken gebeurd via de ***splay-operatie***.
- De weg naar een diepe knoop bevat knopen die ook diep liggen. Terwijl we een knoop wortel maken, moeten de knopen op het zoekpad ook aangepast worden, zodat ook de toegangstijd van deze knopen verbetert, anders blijft de kans bestaan dat een reeks van operaties inefficiënt is.
- Er moet geen extra informatie bijgehouden worden voor knopen, wat geheugen uitspaart.
- De splay-operatie wordt gedefinieerd voor zowel bottom-up als top-down splaybomen.

1.3.1 Bottom-up splayboom

- De knoop wordt eerst gezocht zoals bij een gewone zoekboom.
- De splay-operatie gebeurt van onder naar boven.
- Een knoop kan naar boven gebracht worden door hem telkens te roteren met zijn ouder.
- Om de toegangstijd van knopen op de zoekweg ook te verbeteren, zijn er drie verschillende rotaties:

1. **De ouder p van c is wortel.**
   - Roteer beide knopen zodat c de wortel wordt.
2. **Knoop c heeft nog een grootouder.**
   - Er zijn vier gevallen, die uitvallen in groepen van twee, naar gelang dat p een linker- of rechterkind is van grootouder g.
   - We veronderstellen dat p linkerkind is van g.

   (a) **Knoop c is een rechterkind van p.**
   - Roteer p en c naar links.



Figuur 1.13: Bottom-up splay.

- Roteer g en c naar rechts.

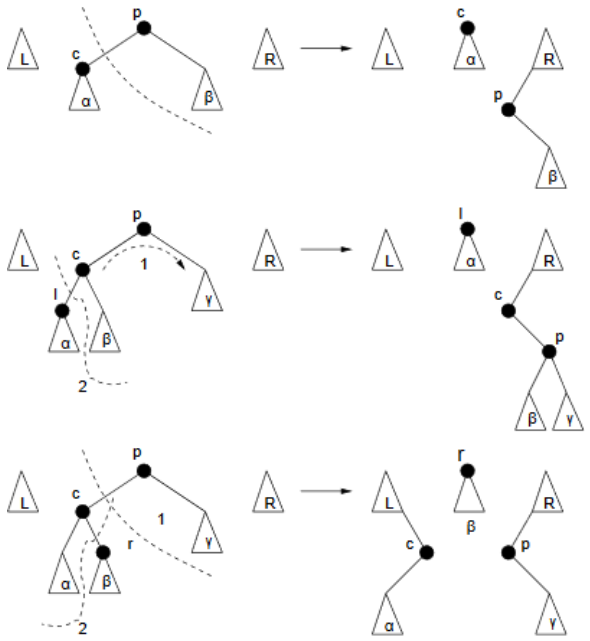
(b) **Knoop c is een linkerkind van p.**

- Roteer g en p naar rechts.
- Roteer p en c naar rechts.

- De **woordenboekoperaties verlopen nu als volgt:**
  - **Zoeken.** De knoop wordt eerst gezocht zoals een gewone zoekboom. Daarna wordt deze tot wortel gemaakt via de splay-operatie.
  - **Toevoegen.** Toevoegen gebeurt ook zoals een gewone zoekboom. De nieuwe knoop wordt dan tot wortel gemaakt met de splay-operatie.
  - **Verwijderen.** Verwijderen gebeurt ook zoals een gewone zoekboom. Daarna wordt de ouder van die knoop tot wortel gemaakt met de splay-operatie.

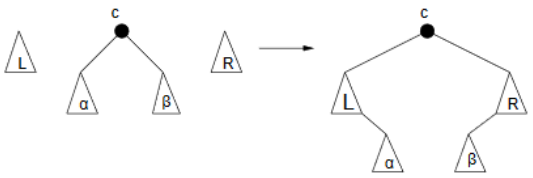
1.3.2 Top-down splayboom

- De splayoperatie wordt uitgevoerd tijdens de afdaling, zodat de gezochte knoop wortel zal zijn als we hem bereiken.
- De boom wordt in drie zoekbomen opgedeeld, L, M en R.
  - Alle sleutels in L zijn kleiner dan die in M.
  - Alle sleutels in R zijn groter dan die in M.
- Eerst is M de oorspronkelijke boom en zijn L en R ledig.
- De huidige knoop op de zoekweg is steeds de wortel van M.
- Stel dat we bij een knoop p uitkomen, en dan nog verder moeten naar een knoop c.
- Er zijn dan twee groepen van 3 gevallen, afhankelijk of c een linker- of rechterkind is van p.



Figuur 1.14: Top-down splay.

- We veronderstellen dat c een linkerkind is van p.
  1. **Knoop c is de laatste knoop op de zoekweg.**
     - Knoop p wordt het nieuwe kleinste element in R samen met zijn rechtse deelboom.
     - Knoop c wordt de wortel van M.
  2. **Knoop c is niet de laatste knoop op de zoekweg.**
     - **We moeten verder afdalen naar het linkerkind l van c.**
       - Roteer p en c naar rechts.
       - Knoop c wordt het kleinste element in R samen met de rechtse deelboom van c.
       - De linkse deelboom van c wordt de nieuwe M met als wortel l.
     - **We moeten verder afdalen naar het rechterkind r van c.**
       - Knoop p wordt het kleinste element in R samen met de rechtse deelboom van p.
       - Knoop c wordt het nieuwe grootste element in L.
       - De rechtse deelboom van c wordt de nieuwe M met als wortel r.
- Als de gezochte knoop c wortel van M is, wordt de splayoperatie afgerond met een **join-operatie**.



Figuur 1.15: Samenvoegen na top-down splayen.

- De **woordenboekoperaties verlopen nu als volgt:**

- **Zoeken.** De knoop met de gezochte sleutel wordt tot wortel gemaakt. Als de sleutel niet gevonden wordt dan is zijn opvolger of voorloper de wortel.
- **Toevoegen.**
- **Verwijderen.**

1.3.3 Performantie van splay trees

- Niet eenvoudig aangezien vorm van de boom vaak verandert.
- We willen aantonen dat een reeks van m operaties op een splay tree met maximaal n knopen een performantie van $O(m \lg n)$ heeft.
- Er wordt een **potentiaalfunctie** Φ gebruikt.
- Elke mogelijke vorm van een splayboom krijgt een reeël getal toegewezen aan de hand van deze potentiaalfunctie.
- Efficiënte operaties die minder tijd gebruiken dan de geamortiseerde tijd per operatie doen het potentiaal stijgen.
- Niet-efficiënte operaties die meer tijd gebruiken dan de geamortiseerde tijd per operatie doen het potentiaal dalen.
- De geamortiseerde tijd van een operatie wordt gedefinieerd als de som van haar werkelijke tijd en de toename van het potentiaal.
  - Stel t_i de werkelijke tijd van de $i-$de operatie.
  - Stel a_i de geamortiseerde tijd van die operatie.
  - Stel Φ_i het potentiaal na deze operatie.

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

- De geamortiseerde tijd van een reeks m operaties is de som van de individuelde geamortiseerde tijden:

$$\begin{aligned}\sum_{i=1}^{m} a_i &= \sum_{i=1}^{m} (t_i + \Phi_i - \Phi_{i-1}) \\ &= t_1 + \Phi_1 - \Phi_0 + t_2 + \Phi_2 - \Phi_1 + t_3 + \Phi_3 - \Phi_2 + \cdots + t_m + \Phi_m - \Phi_{m-1} \\ &= \Phi_m - \Phi_0 + \sum_{i=1}^{m} t_i\end{aligned}$$

- Als de potentiaalfunctie zo gekozen wordt zodat het eindpotentiaal Φ_m zeker niet kleiner is dan de beginpotentiaal Φ_0, dan vormt de totale geamortiseerde tijd een **bovengrens** van de werkelijke tijd want de boom zal zeker niet slechter zijn.
- De eenvoudigste potentiaalfunctie geeft voor elke knoop i een gewicht s_i die gelijk is aan het aantal knopen in de deelboom waarvan hij wortel is. De potentiaal van de boom is dan de som van de logaritmen van deze gewichten:

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

- We noemen $\lg s_i$ de rang r_i van knoop i.

- Performantie-analyse van bottom-up splayboom:
  - Performantie is evenredig met de diepte van de knoop, en dus met het aantal uitgevoerde rotaties.
  - We willen aantonen dat de geamortiseerde tijd voor het zoeken naar een knoop c gevolgd door een splay-operatie op die knoop gelijk is aan

    $$O(1 + 3(r_w - r_c))$$

    waarbij r_w de rang van de wortel is en r_c de rang van de gezochte knoop.
    - Als c reeds de wortel is, dan is $r_w = r_c$ en blijft het potentiaal dezelfde.

      $$O(1 + 3(r_w - r_c)) = O(1)$$

    - Anders moeten zoveel splay-operaties uitgevoerd worden als de diepte van de knoop (moet niet gekend zijn).
      - Een zig wijzigt de rang van c en p

        $$a < 1 + r'_c - r_c$$

      - Een zig-zag wijzigt de rang van c, p en g

        $$a < 2(r'_c - r_c)$$

      - Een zig-zig wijzigt de rang van c, p en g

        $$a < 3(r'_c - r_c)$$

  - De bovengrenzen voor de drie operaties bevatten dezelfde positieve term $r'_c - r_c$ maar met verschillende coëfficiënten.
  - De totale geamortiseerde tijd is een som van dergerlijke bovengrenzen, maar kan niet vereenvoudigt worden als coëfficiënten niet gelijk zijn.
  - Aangezien het bovengrenzen zijn, wordt de grootste coëfficiënt genomen.
  - In de som vallen de meeste termen nu weg, behalve de rang van c voor en na de volledige splay-operatie.
  - De geamortiseerde tijden van de woordenboekoperaties op een bottom-splay tree met n knopen zijn nu:
    - **Zoeken.** $O(1 + 3 \lg n)$ want $s_w = n$.
    - **Toevoegen.** $O(1 + 4 \lg n)$.
      Op de zoekweg worden de rang van knopen $p_1, p_2, ...p_k$ op de zoekweg gewijzigt. Stel s_{p_i} het gewicht van knoop p_i voor het toevoegen en s'_{p_i} het gewicht van knoop p_i na het toevoegen. De potentiaaltoename is dan

      $$\lg\left(\frac{s'_{p_1}}{s_{p_1}}\right) + \lg\left(\frac{s'_{p_2}}{s_{p_2}}\right) + \cdots + \lg\left(\frac{s'_{p_k}}{s_{p_k}}\right) = \lg\left(\frac{s'_{p_1}}{s_{p_1}}\frac{s'_{p_2}}{s_{p_2}} \cdots \frac{s'_{p_k}}{s_{p_k}}\right)$$

      Deze is nooit groter dan

      $$\lg\left(\frac{s'_{p_1}}{s_{p_k}}\right) \leq \lg(n+1)$$

    - **Verwijderen.** Het effect van verwijderen is nooit positief.
  - De geamortiseerde tijd voor een reeks van m woordenboekoperaties is de som van de geamortiseerde tijden voor de individuele operaties.
  - Stel n_i het aantal knopen bij de $i-$de operatie wordt die tijd $O(m + 4\sum_{i=1}^{m} \lg n_i) = O(m + 4m \lg n) = O(m \lg n)$.

1.4 Gerandomiseerde zoekbomen

- De performantie van de woordenboekoperaties op een gewone zoekboom is $O(\lg n)$ als elke toevoegvolgorde even waarschijnlijk is.
- Gerandomiseerde zoekbomen maken gebruik van een random generator om de operatievolgorde te neutraliseren.
- Deze bomen blijven steeds random.
- Een **treap** is een gerandomiseerde zoekboom.
  - Elke knoop krijgt naast een sleutel ook een prioriteit, die door de random generator wordt toegekend als de knoop toegevoegd wordt.
  - De prioriteiten van de knopen voldoen aan de heapvoorwaarde: de prioriteit van een kind is maximaal even hoog als die van zijn ouder.
- De woordenboekoperaties:
  - **Zoeken.** Zoeken moet geen rekening houden met de prioriteiten en verloopt zoals een normale binaire zoekboom.
  - **Toevoegen.** Eerst wordt er normaal toegevoegd. De knoop wordt nadien naar boven geroteerd om aan de heapvoorwaarde te voldoen.
  - **Verwijderen.** De te verwijderen knoop krijgt de laagste prioriteit, zodat die naar beneden geroteerd wordt. Dit blad kan dan verwijdert worden.

1.5 Skip lists

- Een meerwegszoekboom geïmplementeerd met gelinkte lijsten.
- Alle bladeren zitten op dezelfde diepte.
- Elke lijstknoop heeft plaats voor één sleutel en één kindwijzer.
- Een knoop met k kinderen bevat $k - 1$ sleutels, zodat er één sleutelplaats over blijft.
- (zoekt gewoon eens een foto op)