1a. Een abstracte datastructuur is een beschrijving van een datastructuur, met de specificatie van wat er opgeslagen wordt (de data en hun structuur) en welke operaties op de data zijn toegestaan.

a. Een stapel is een lineaire lijst van data items, waarvan de toegang slechts een een kant mogelijk is. Daarom is de laatst toegevoegde waarde de eerste die verwijderd kan worden, en spreken we van LIFO. Typische operaties die men gewoonlijk bij een stapel aantreft zijn Push (voeg een bepaald item toe op de stapel), Pop (verwijder het laatst toegevoegde element), en Top (geef de waarde van het laatste element, zonder deze te verwijderen). Daarnaast verwacht men gewoonlijk IsLeeg (zijn er nog elementen) en Init (initialiseer een stapel).

Veel gebruikte implementaties zijn de array (de top van de stapel is dan het hoogst in gebruik zijnde element en deze index wordt voor snelle toegang bijgehouden) en een gelinkte lijst (een pointer naar het bovenste element, en vandaar naar lager gelegen elementen).

c. Union-Find is een ADT die een partitie (opdeling in een aantal disjuncte niet-lege deelverzamelingen) bijhoudt van een domein (gebruikelijk de getallen $1, \ldots, n$). Initieel bestaat de partitie uit losse deelverzamelingen met elk een element.

De operaties zijn, zoals de naam al aangeeft:

- Find: gegeven een element uit het domein, bepaal de naam van de partitie waartoe het element behoort.
- Union: gegeven twee [namen van] deelverzamelingen in de partitie, wordt een nieuwe partitie gevormd door deze twee samen te voegen; er wordt een nieuwe naam bepaald voor de vereniging.

Een belangrijke toepassing is het algoritme van Kruskal voor het bepalen van een minimale opspannende boom in een graaf. Union-Find zorgt dat er geen lijnen gekozen worden die een kring zouden vormen met eerdere keuzes.

Iets over implementatie.

2a. Draden lopen van een element naar zijn LWR-opvolger voorzover dat element zelf geen rechter kind heeft. Draden zijn dus $4 \to 3$, $5 \to 2$, $7 \to 1$, $11 \to 9$, $12 \to 8$, en ook $8 \to$ NIL. De laatste link is weliswaar technisch niet naar een opvolger maar is conceptueel vaak handig. [Illustratie]

Overigens, de nummering van de knopen uit het voorbeeld van deze opgave is in pre-orde.

b. Het eerst te bezoeken element is het meest linker element in de boom, vanuit de wortels dus linksaf totdat de linker pointer NIL is.

Daarna bepalen we herhaald de opvolger van de laatst bezochte knoop, en bezoeken deze. De LWR-opvolger is ofwel de knoop aangegeven door een draad (als de rechter pointer een draad is) dan wel het meest linkse element van de rechter deelboom (als de rechter pointer een tak is), In dat laaststst geval gaan we een keer rechts, en weer zo lang mogelijk naar links.

Opmerking. Draad en rechter kind gebruiken dezelfde pointer. Er is slechts een bit die in de knoop aangeeft wat de betekenis van de rechter pointer is: kind of opvolger (=draad).

c. De plaats waar de nieuwe sleutel wordt toegevoegd is als bij aan BZB, linksaf bij een sleutel kleiner dan de sleutel in de knoop, en anders rechtsaf. Let op dat we rechtsaf alleen takken volgen, en geen draden!

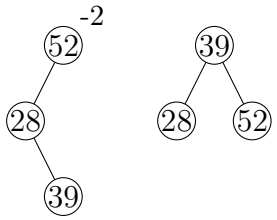
Nu de draden aanpassen. Er komen geen draden naar de nieuwe knoop. Als de nieuwe knoop zelf een linker kind is komt er een draad naar de vader. Als de nieuwe knoop een rechter kind is krijgt deze een draad naar de knoop die oorspronkelijk de opvolger is van de vader, bewaar dat adres dus. (Het is niet nodig, maar die opvolger kan ook gevonden worden tijdens het bepalen van de plek van de nieuwe knoop. Het is het punt waar voor het laatst linksaf werd geslagen.)

3a. Een AVL-boom is een binaire zoekboom waarin voor elke knoop de hoogtes van beide deelbomen ten hoogte één verschillen.

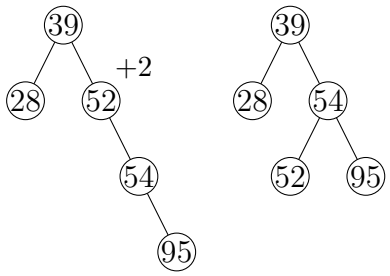
Dat geeft voldoende informatie over de *structuur* van AVL-bomen: de toegestane vorm en de plaatsing van de sleutels.

(nb. De AVL-boom is niet 'zelf-organiserend'; dat geldt voor splay trees, waar de rotaties die uitgevoerd worden bij het zoeken en plaatsen de boom redelijk in vorm houden. Hier wordt de vorm expliciet afgedwongen.)

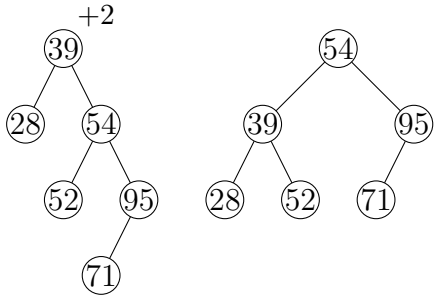
b. Lege boom. Toevoegen $52, 28, 39$. Uit balans bij 52, LR, dus dubbele rotatie naar rechts.



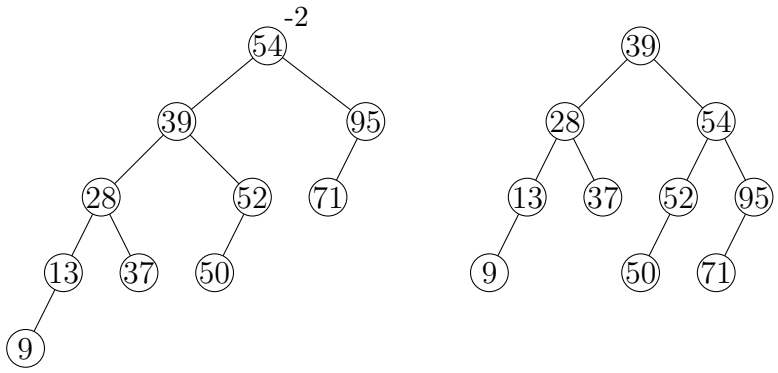
Toevoegen $54, 95$. Uit balans bij 52 (!), RR, dus enkele rotatie naar links.



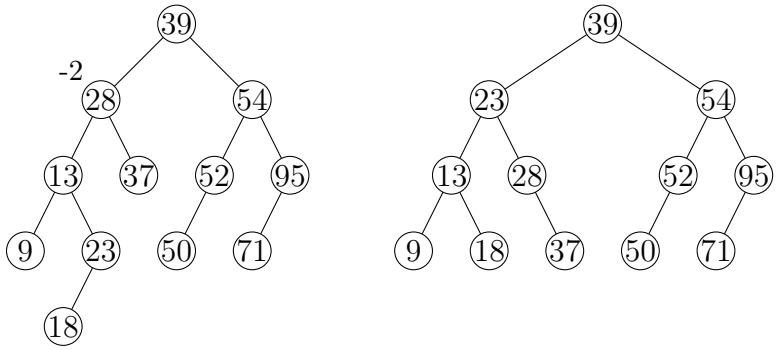
Toevoegen 71. Uit balans bij 39, RR, dus enkele rotatie naar links.



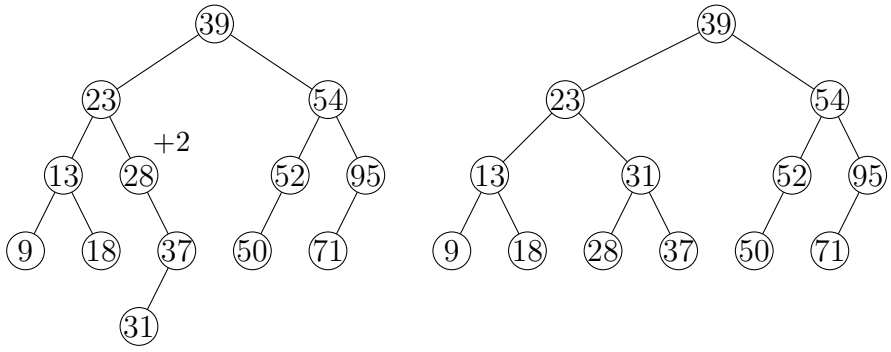
Toevoegen $39, 37, 50, 13, 9$. Uit balans bij 54, LL, dus enkele rotatie naar rechts.



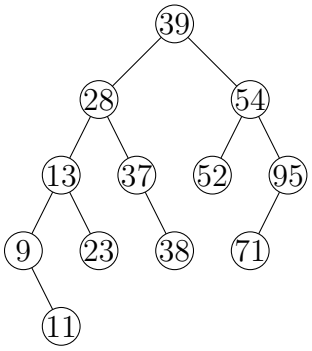
Toevoegen 23, 18. Uit balans bij 28, LR, dus dubbele rotatie naar rechts.
Merk op dat als we vóór roteren eerst nog 31 hadden toegevoegd, de boom weer in balans zou zijn. AVL-bomen kijken echter niet vooruit.



Toevoegen 31. Uit balans bij 28, RL, dus dubbele rotatie naar links.



c. Teken een zogenaamde Fibonacci boom, die we hierboven al bijna zagen. Als hier 52 wordt verwijderd krijgen we onbalans bij 54. Rotatie daar levert een boom die minder hoog is dan de oorspronkelijke boom, wat tot gevolg heeft dat we vervolgens in de wortel 39 uit balans raken.



Helaas, een dubbele rotatie telt hier niet als twee verschillende rotaties.

4a. De adresfunctie bij een hash-methode moet een goede spreiding geven van de mogelijke sleutels en efficient te berekenen zijn.

De stapfunctie mag geen delers met de tabelgrootte gemeenschappelijk hebben, om er voor te zorgen dat herhaald stappen nemen uiteindelijk alle adressen van de tabel bereikt.

(ook: de stapfunctie moet onafhankelijk gekozen worden van de adresfunctie, om secundair clusteren te voorkomen, zie hierna.)

b. Primaire clustering. Bij lineair hashen neigen er blokken te ontstaan van aaneengesloten bezette adressen. (Dit leidt tot slechte zoekresultaten, omdat soms zo'n blok geheel bezocht wordt voordat we concluderen dat een sleutel niet in de tabel zit.) Grotere blokken groeien ook sneller, omdat alle sleutels die op een adres uit het blok toegevoegd worden uitkomen op het adres net voor het blok.

Secundaire clustering. Bij hashmethoden waarbij synoniemen hetzelfde zoekpad volgen. (Dit is geen zichtbaar 'blok', maar synoniemen zitten elkaar wel in de weg.) Oplossing is het kiezen van een dubbele hashmethode.

c. De adressen:

K	10	22	31	4	15	28	17	88	59
K mod 11	10	0	9	4	4	6	6	0	4

We lopen naar links, dat staat expliciet in de opgave. Modulo zorgt ervoor dat de twee uiteinden van de tabel als het ware tegen elkaar liggen. De eerste rij bestaat uit de eerste sleutels, die meteen geplaatst kunnen worden. In de uitleg staat op welke adressen de volgende sleutels geprobeerd worden. Op die adressen heb ik de bestaande sleutel onderstreept.

0	1	2	3	4	5	6	7	8	9	10	
22				4					31	10	direct geplaatst
22			15	<u>4</u>					31	10	$h(15) = 4$, bezet, naar 3
22			15	4		28			31	10	28 direct geplaatst
22			15	4	17	<u>28</u>			31	10	$h(17) = 6$, bezet, naar 5
<u>22</u>			15	4	17	28		88	<u>31</u>	<u>10</u>	$h(88) = 0$, bezet, via $10, 9$ naar 8
22		59	<u>15</u>	<u>4</u>	17	28		88	31	10	$h(59) = 4$, bezet, via 3 naar 2
22		59	15	4	17	28		88	31	10	resultaat

d. De stapgroottes (die we overigens lang niet allemaal gebruiken):

K	10	22	31	4	15	28	17	88	59
$1 + K$ mod 10	1	3	2	5	6	9	8	9	10

Omdat $9 \equiv -2 \bmod 11$ geldt dat stappen met 9 naar links hetzelfde is als stappen met 2 naar rechts.

0	1	2	3	4	5	6	7	8	9	10	
22				4					31	10	direct geplaatst
22			15	<u>4</u>					<u>31</u>	10	$h(15) = 4$, bezet, $p(15) = 6$ via $4-6 \equiv 9$ naar $9-6 \equiv 3$
22			15	4		28			31	10	28 direct geplaatst
22	17		15	4		<u>28</u>			31	<u>10</u>	$h(17) = 6$, bezet, $p(17) = 8$ via $6-8 \equiv 9$ naar $9-8 \equiv 1$
<u>22</u>	17	88	15	4		28			<u>31</u>	10	$h(88) = 0$, bezet, $p(88) = 9$ naar $0-9 \equiv 2$
22	17	88	15	<u>4</u>	59	28			31	10	$h(59) = 4$, bezet, $p(59) = 10$ naar $4-10 \equiv 5$
22	17	88	15	4	59	28			31	10	resultaat