1a) Een ADT wordt gekenmerkt door de opgeslagen gegevens en de beschikbare operaties.

De Priority Queue bevat en verzameling elementen waarbij elk element en eigen waarde (prioriteit) heeft. Verwijderen van eementen gebeurt alleen op grond van die prioriteit.

Operaties. Init: geeft een lege Priority Queue. IsEmpty: geeft terug of de PQ leeg is.I Size: geeft het aantal elementen Insert: voegt een element met zijn prioriteit toe aan de PQ. GetMax: geeft het element met maximale prioriteit. DelMax: verwijdert het element met maximale prioriteit.

Opm. Het is ook mogelijk de PQ te bekijken met minimale waarde (in plaats van maximale waarde).

Non-standaard operaties zijn IncreaseKey (waarde gegeven element verhogen) en Meld (samenvoegen van twee PQ's). De eerste is speciaal van toepassing bij Dijkstra's algoritme.

1b) Een *leftist tree* is een binaire boom (1) met heap-ordening (de sleutel van een knoop is groter dan die van zijn kinderen) waar (2) in elke knoop de 'bladafstand' van het rechter kind niet groter is dan de bladafstad van het linker kind. De bladafstand van een knop is de afstand tot het dichtstbijzijnde blad.

Opm. Er is dus zowel een beperking op de plaatsing van de sleutels als op de structuur van de boom.

De *rits* operatie voegt twee bomen recursief samen. Kies van de twee bomen de boom met de hoogste wortel en knip daar de rechtersubboom vanaf. De nieuwe rechter subboom is de boom die ontstaat door (recursief) de oude rechter subboom te ritsen met de tweede boom (die met de kleinere wortel). Herstel bij terugkomst van de recursie de bladafstandseigenschap door links en rechts te verwisselen waar nodig.

De PQ operaties zijn als volgt. Insert: rits de huidige boom met een boom met een enkele knoop (bestaande uit het nieuwe element). GetMax: is de knoop in de wortel van de leftist tree. DeleteMax: verwijder de wortel; de nieuwe boom ontstaat dor de twee subbomen in elkaar te ritsen.

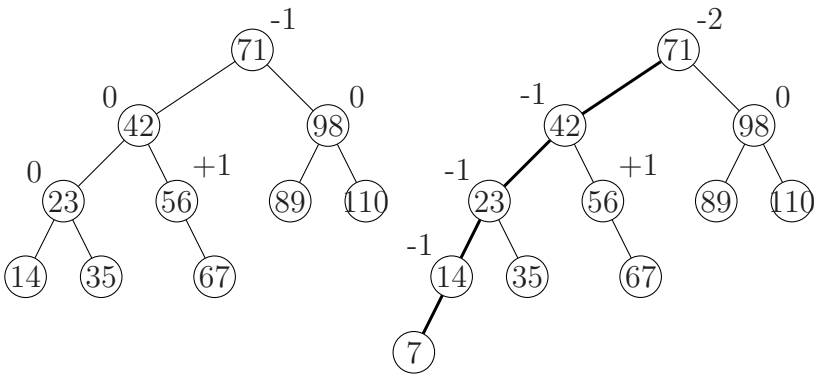
1c) Voorzie de zoekboom van extra informatie: sla in elke knoop het aantal elementen op dat in de linker subboom zit.

Zoek nu recursief naar het k-de element, als de wortel waarde ℓ bevat.

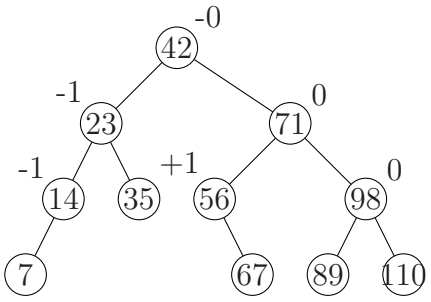
Als $k = \ell + 1$: we zitten in de gezochte knoop. Als $k \leq ell$: zoek naar de k-de knoop in de linker subboom. Als $k > \ell + 1$: zoek naar de $k - \ell - 1$-ste knoop in de linker subboom.

2a) Omdat ze gebalanceerd zijn je de ADT Set operaties zoeken, toevoegen en verwijderen in gegarandeerd logaritmische tijd uitvoeren.

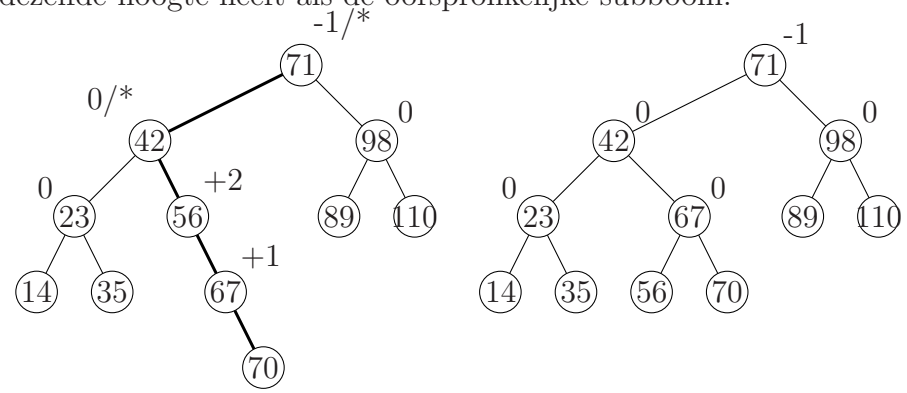
b)i Allereerst, voor de volledigheid, de balans in elke knoop (bladeren hebben altijd balans 0). Daarna voegen we 7 toe:



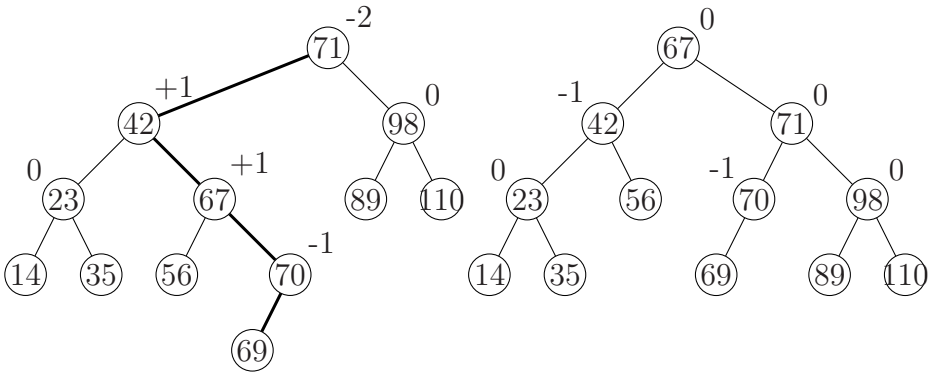
Zoek naar het laagste punt met knoop uit balans, omhoog vanuit de toegevoegde knoop naar de wortel. Enkele rotatie rechts om wortel 71 (situatie LL):



b)ii Voeg 70 toe. Laagste punt uit balans 56 heeft situatie RR, dus enkele rotatie om 56 naar rechts. De knopen boven 56 hebben alleen tijdelijk een andere balans, na rotatie is de oorspronkelijke waarde weer terug. Dat komt omdat na rotatie de subboom op die plek dezelfde hoogte heeft als de oorspronkelijke subboom.

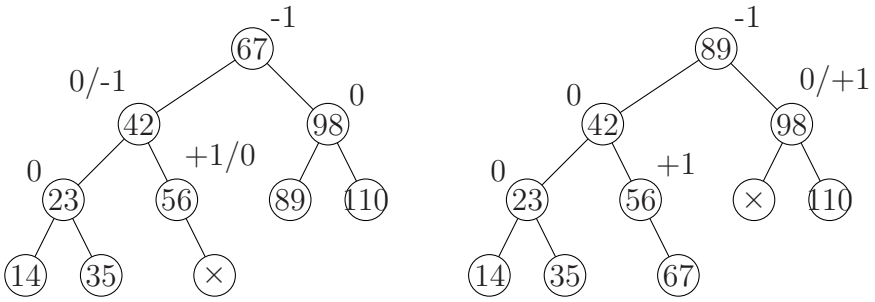


Voeg nu 69 toe. Uit balans in de wortel. Situatie LR, dubbele rotatie naar links, om 71:

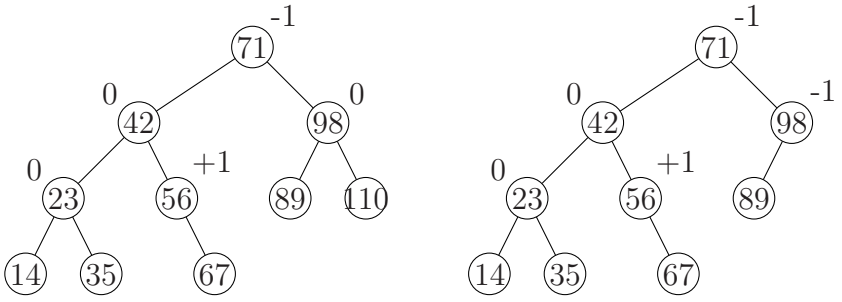


c)i Verwijderen gebeurt altijd uit een blad. Verwissel als de knoop niet in een blad staat deze met de voorganger of opvolger (in symmetrische ordening).

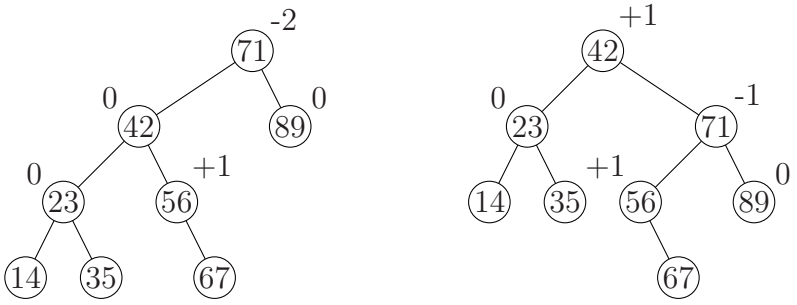
In deze boom blijft in elk van de twee gevallen de boom na verwijderen een AVL-boom. Er hoeven geen rotaties te worden toegepast om de vorm te herstellen.



c)ii Sleutel 110 staat in een blad. Na verwijderen is de boom nog een correcte AVL-boom.



Verwijder nu 98 (eerst verwisselen met voorganger 89). De resulterende boom is uit balans in de wortel. Enkele rotatie naar rechts om 71. Klaar.



3a) Een topologische ordening geeft een volgorde (=ordening) aan de knopen van de graaf on

zo'n manier dat voor elke pijl van de graaf geldt dat het vertrekpunt voor het aankomstpunt ligt ('de pijlen lopen van links naar rechts').

Een topologische ordening van de graaf $G = (V, E)$ is een lijst $(v_1, v_2, \ldots, v_n)$ zó dat als $(v_i, v_j) \in E$, dan $i < j$.

b) Als een knoop y kan worden bereikt vanuit knoop x dan moet knoop y ná knoop x in de topologische orening komen. DFS zekt de knopen die bereikbaar zijn vanuit x. Daarmee vindt DFS de topologische ordening in omgekeerde volgorde.

Als de knoop volledig is afgehandeld in de functie DFS kan deze op een stapel worden gezet. De elementen op de stapel staan dan in topologische ordening

Het kan zijn dat DFS niet alle knopen bezoekt. Roep DFS dan aan voor knopen die onbezocht blijven.

c) Het tijdstip waarop de projecten in knoop i kunnen beginnen is de maximale waarde van de tijdstippen waarop de projecten van inkomende takken geëindigd zijn (op hun starttijd met de gegeven lengte).

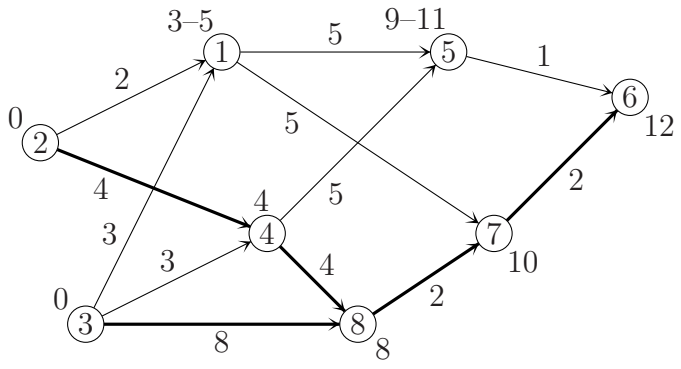
Er zijn twee manieren waarop we kunnen werken. Kijk in elke knoop terug naar de voorgangers, en bepaal de maximale waarde. Of we kijken in elke knoop vooruit en hogen een tijdstip dat in die knoop staat op als we via de pijl later aankomen. De eerste methode is makkelijk met de hand, maar de laatste methode verdient de voorkeur als we met adjacency-lijsten werken (die hebben alleen uitgaande takken):

```
foreach node x
do  start[x]=0;
od
foreach node x in topological order
do  foreach y in adjacency list of x
    do  // so (x,y) is an edge
        start[y] = max { start[y], start[x] + length(x,y) };
    od
od
```

Het voorbeeld op de handmatige methode. De topologische ordening die we gebruiken is $(2, 3, 1, 4, 5, 8, 7, 6)$ – ik geloof niet dat er stond dat de ordening algoritmisch bepaald moest worden?

2 start$[2] = 0$, geen voorgangers
3 start$[3] = 0$, geen voorgangers
1 start$[1] = \max\{0 + 2, 0 + 3\} = 3$
4 start$[4] = \max\{0 + 4, 0 + 3\} = 4$
5 start$[5] = \max\{3 + 5, 4 + 5\} = 9$
8 start$[8] = \max\{4 + 4, 0 + 8\} = 8$
7 start$[7] = \max\{3 + 5, 8 + 2\} = 10$
6 start$[6] = \max\{9 + 1, 10 + 2\} = 12$

Het moment waarop het totale project klaar is, is op tijdstip 12, als er geen vertragingen zijn. **extra.** Hieronder het project nogmaals, met eerste begintijden van de verschillende projecten, maar ook de laatste begintijden als je van achter naar voor werkt. Vet aangegeven de projecten waar geen vertraging mag optreden, deze zijn 'kritiek'.

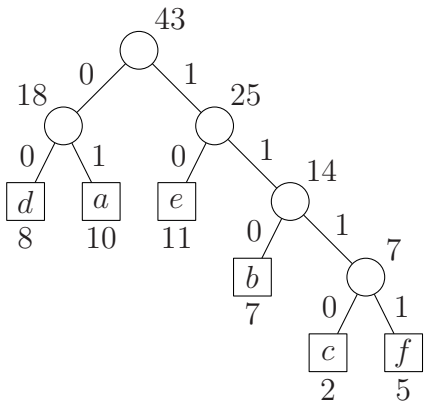


4a) Zie diktaat. Kijk onder Huffman. Bij gegeven letters en hun frequenties wordt de optimale prefixcode voor de letters bepaald. Complexiteit $\mathcal{O}(n \lg n)$ omdat in elk van de $n-1$ stappen de twee kleinste frequenties moeten worden bepaald. Dat kan met een binaire heap.

b) De volgende codeboom is een mogelijkheid. Geef tussenstappen!

De getallen bij de knopen geven de (totale) frequenties bij de subbomen weer. deze worden van klein naar groot gecombineerd.

Start met 2(c), 5(f), 7(b), 8(d), 10(a), en 11(e). Dan 2(c)+5(f) = 7(c+f); 7(c+f)+7(b) = 14(b+c+f); 8(d)+10(a) = 18(a+d); 11(e)+14(b+c+f) = 25(b+c+e+f); en tenslotte 18(a+d)+25(b+c+e+f) = 43(a+b+c+d+e+f).



Omdat hier a,d en e op dezelfde hoogte in de boom hangen kunnen we bijvoorbeeld e met a of d verwisselen. Dit levert een even goede boom op, die helaas geen boom is die Huffman zou vinden.