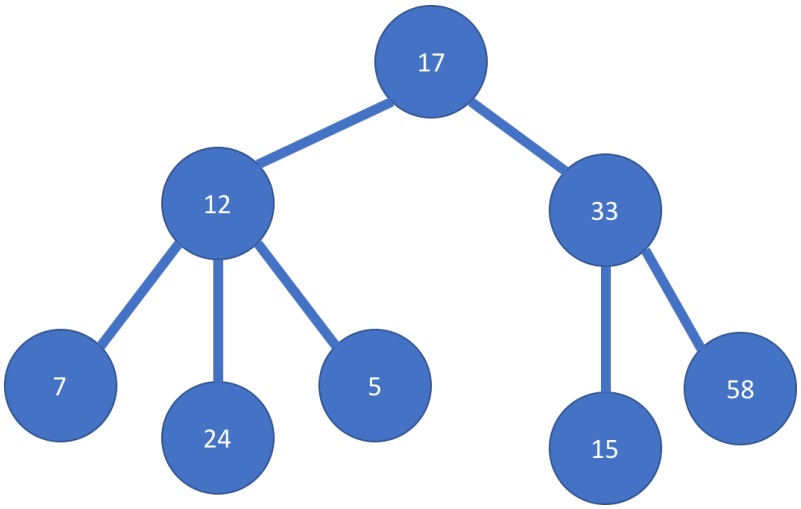
Opgave 1: Bomen snoeien (6p)

In deze opgave breidt je de datastructuur voor algemene bomen uit met de mogelijkheid om op basis van een eigenschap knopen te verwijderen. Bekijk bijvoorbeeld de volgende boom.



(a) (2p) Maak een generieke klasse `Tree<T>` voor algemene bomen en gebruik die ome en object te maken dat de boom uit bovenstaande figuur representeert. Dit is in dit geval dus een object van type `Tree<Integer>`. Demonstreer je code door de boom naar `System.out` te printen. Je printfunctie hoeft geen kunstwerk te worden, maar moet wel de structuur van de boom en de waardes van de knopen laten zien.
Als het je niet lukt om een generieke Tree klasse te maken, mag je ook een niet-generieke boom met int waardes maken. In dat geval krijg je slechts 1 punt.

(b) (2p) Voeg een methode `void removeNodesGreaterOrEqualTo(T x)` die alle nodes verwijdert waarvan de waarde groter of gelijk is dan het meegegeven argument. Als je een node verwijdert, verwijder je automatisch ook al zijn kinderen. **Maak gebruik van recursie.**

(c) (2p) Voeg nog een methode `void removeMaximum` toe die de knoop met de hoogste waarde uit de boom verwijdert (58 in het voorbeeld). Maak hiervoor gebruik van een hulpfunctie `T findMaximum()` die de hoogste waarde in de boom vindt.
De verwijder techniek is hierbij hetzelfde als bij opdracht 1b.

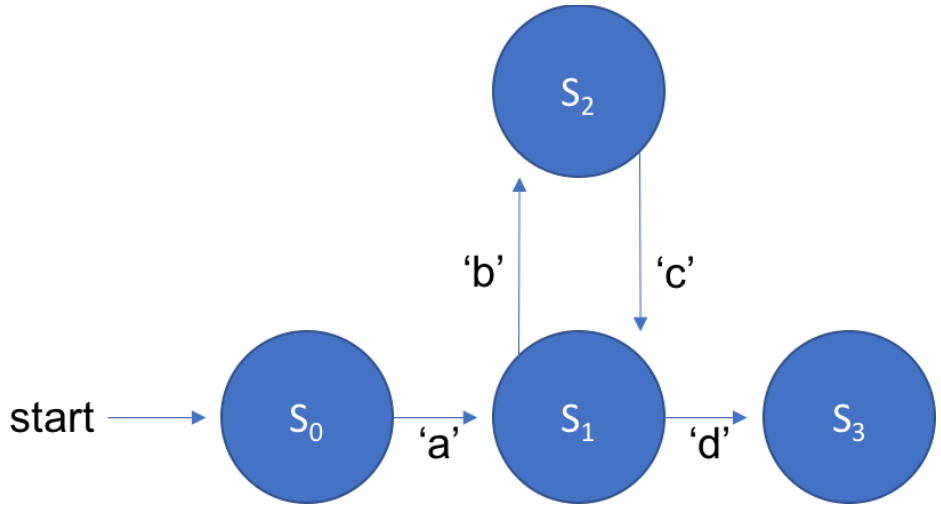
Opgave 2: Reguliere expressies matchen (4p)

Je hebt misschien al wel eens gebruik gemaakt van reguliere expressies om invoer in user interfaces te valideren. Reguliere expressies zijn een notatie om patronen uit te drukken waar strings aan moeten voldoen. Ze bestaan uit stukjes tekst en een aantal speciale karakters om aan te geven dat een stukje tekst optioneel is (het teken '?'), of een willekeurig aantal keer herhaald mag worden (het teken '*').

Een eenvoudige reguliere expressie is bijvoorbeeld ''a(bc)*d?''. Dit kun je lezen als: De string begint met het karakter 'a', dan een willekeurig aantal keer de substring "bc", en dan optioneel nog het teken 'd'.

- Strings die voldoen aan deze expressie zijn bijvoorbeeld:
  "a", "abc", "abcbcbc", "ad", "abcd" en "abcbcd".
- Strings die niet voldoen zijn bijvoorbeeld:
  "aaa", "abcbd", "def", "xyyyz" en "abcdd".

Een efficiënte manier om het kijken of strings voldoen aan reguliere expressies te implementeren is met behulp van eindige automaten. Dit zijn grafen waarin de vertices toestanden representeren, en de degens teostandsovergangen. Onderstaande automaat is een implementatie van het voorgaande voorbeeld ''a(bc)*d?''.



De vertices met een dubbele omranding (S1 en S3) *… (tekst) …*

(a) (2p) Maak een datastructuur om eindige automaten te representeren. Zorg dat de toestanden informatie bevatten over of ze accepterend zijn, en dat degens de bijbehorende karakters bevatten. Je kunt hiervoor je implementatie van grafen uit het huiswerk als uitgangspunt nemen.
Maak een object dat de automaat (graaf) uit het bovenstaande representeert. Demonstreer je code door met een toString methode de graaf naar System.out te printen. In de output moet alle informatie over de vertices en edges te zien zijn.

(b) (2p) Voeg een methode `bool match(String s)` aan je klasse voor automaten toe. Deze methode bepaalt voor een string of hij voldoet aan de reguliere expressie die bij de automaat hoort.
Het algoritme om dit te bepalen is als volgt.:

- Houdt bij in welke vertex van de graaf je bent. Begin de toestand S0.
- Loop nu van begin tot eind karakter voor karakter door de te testen string (s) heen.
- Telkens als je een karakter leest “volg” je de uitgaande edge behorende bij het karakter.
- Als er geen uitgaande edge bestaat voor dat karakter weet je dat de string niet matcht en ben je klaar.
- Als je de hele string doorlopen hebt kijk je in welke toestand je bent geëindigd. Als dit een van de “accepterende” states is (S_1 of S_3) dan match de string. Zo niet, dan matcht de string niet.

Demonstreer je code door het resultaat van match te laten zien voor de voorbeeldstrings uit de beschrijving van de opgave.