Øvelse 1

Skriv en algoritme, der tager en string som input og returnerer det oftest forekommende ord. Stringen er almindelig tekst (store og små bogstaver), og ord er adskilte af blanke, kommaer, punktummer eller kombinationer af disse (se eksempel nedenfor – du kan antage, at der altid er en blank imellem to ord, og at den sidste karakter altid er et punktum). Ord antages at være ens, selv om nogle forekomster begynder med lille bogstav og andre med stort bogstav.

Du kan antage som precondition for algoritmen, at input parameteren kun indeholder små bogstaverer [a-z] store bogstaver [A-Z], blanke, kommaer (,) og punktummer (.).

Eksempel:

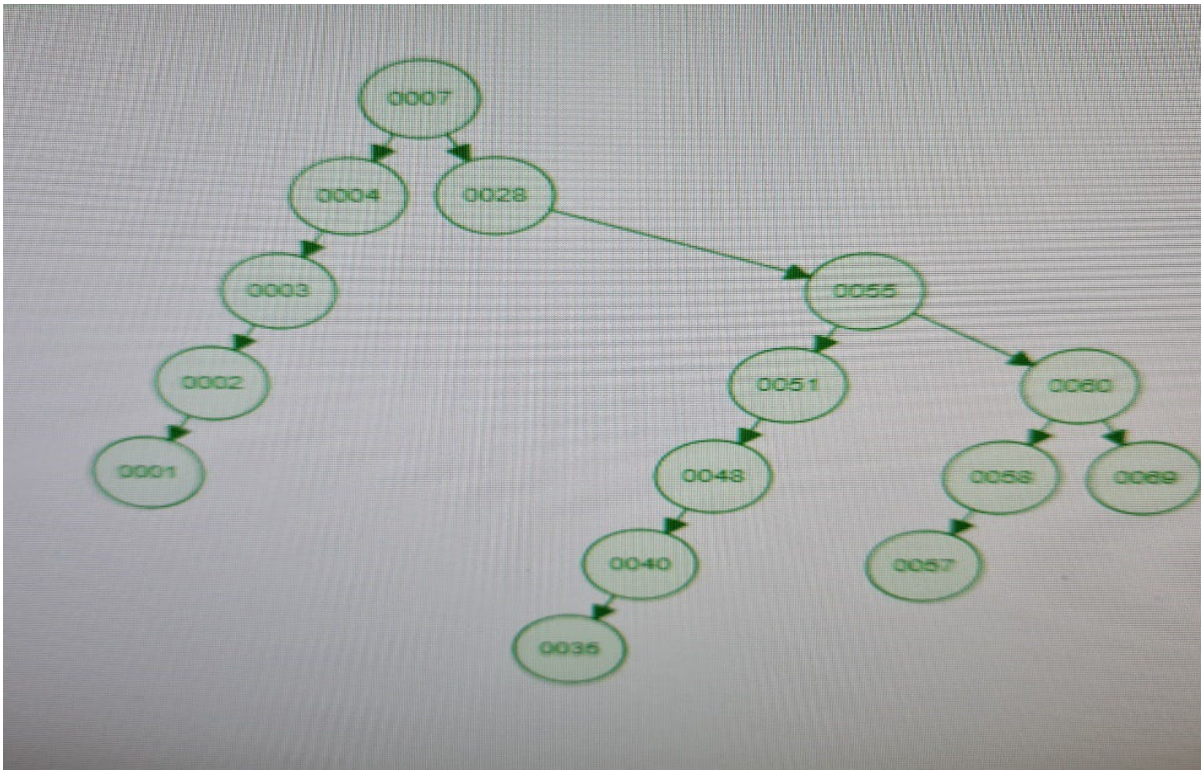
{The cattle were running back and forth, but there was no wolf to be seen, heard, or smelled, so the shepherd decided to take a little nap in a bed of grass and early summer flowers. Soon he was awakened by a sound he had never heard before.}

Det korrekte svar I dette eksempel er ordet **a**, som forekommer tre gange.

Algoritmen skal være optimeret hvad angår køretid, og der skal angives et Store-O estimat for din løsning. Svaret skal begrundes.

Øvelse 2

Nedenstående figur forestiller et binært søgetræ.



Vi vil definere et begreb, som vi kalder en *gren,* og som karakteriseres på følgende måde. Det er en samling af tre noder, hvorom det gælder:

- Node X har ét barn.
- Node X har ingen søskende.
- Node X's barn har ingen søskende.
- Node X's barn har ét barn, som er et blad.

I ovenstående træ opfylder noderne 3 og 48 definitionen af X, og træet indeholder altså to grene.

Programmeringsopgaven går ud på at skrive en metode til implementering i dit træ (det behøver ikke at være et binært søgetræ, men skal være et binært træ), som kan returnere antallet af grene i træet. I vores eksempel er det korrekte svar som nævnt to. Det antages, at roden *ikke* kan være en del af en gren.

Tip: Det kan anbefales at skrive en hjælpemetode med følgende signatur:

```
BinaryNode getOnlyChild(BinaryNode node)
```

Metoden returnerer parameterens eventuelle enebarn. Hvis der er to eller ingen børn, returneres `null`.

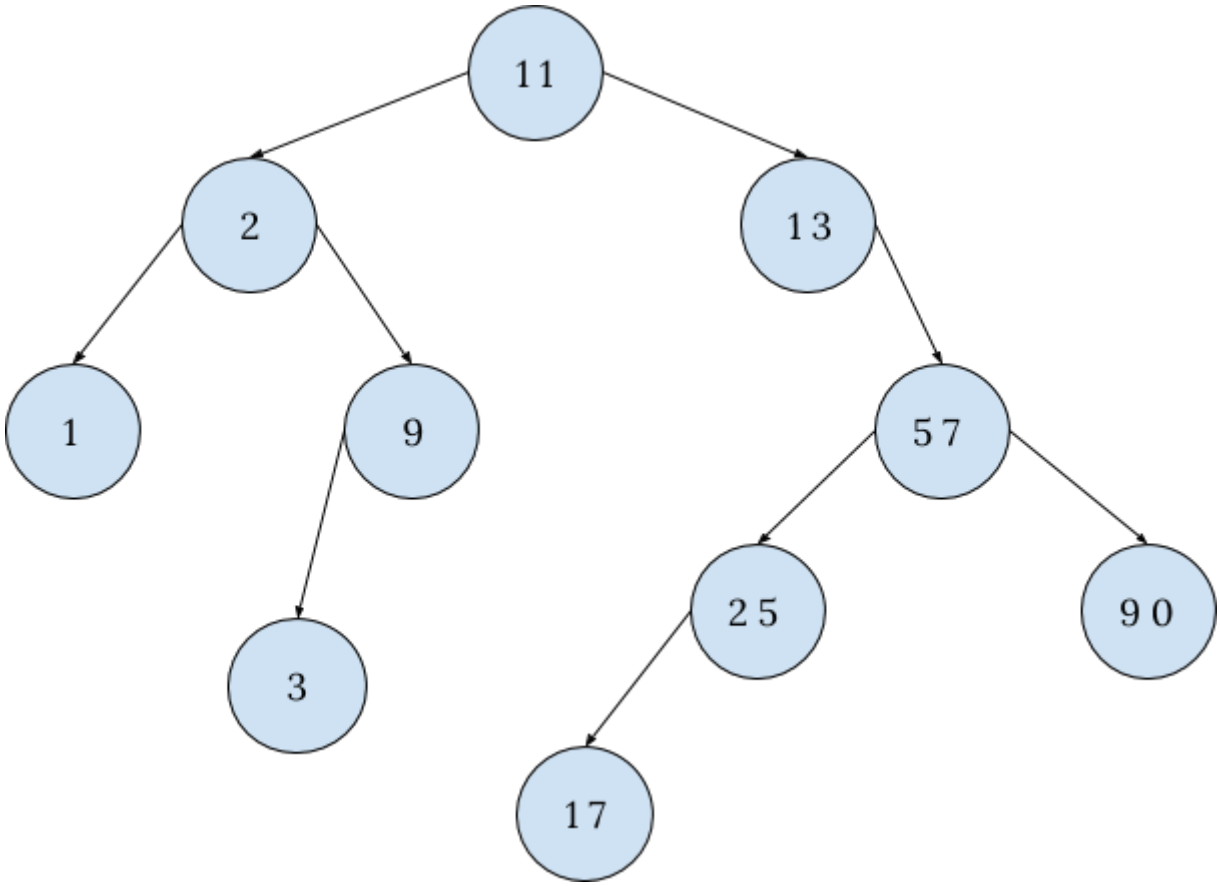
Husk kun at aflevere den kode du har tilføjet til dit træ og ikke også resten af træet.

Supplerende opgave 1: hvad karakteriserer træet i figuren i øvrigt – hvad er træet, og hvad er det ikke? Derudover ønskes de sædvanlige numeriske oplysninger, og desuden: hvad er træets optimale højde, og hvordan kan den udtrykkes matematisk?

Supplerende opgave 2: hvordan ville du gribe opgaven an, hvis du skulle transformere det binære søgetræ i figuren til en prioritetskø? Hvilke trin ville det kræve, og hvad vil tidskompleksiteten (Store O) for din løsning være?

Øvelse 3

Nedenfor ses et binært søgetræ, som ikke er et AVL-træ:



List rækkefølgen I hvilken noderne vil blive besøgt I en en in order og i en level order traversering.

Hvad er træets internal path length?

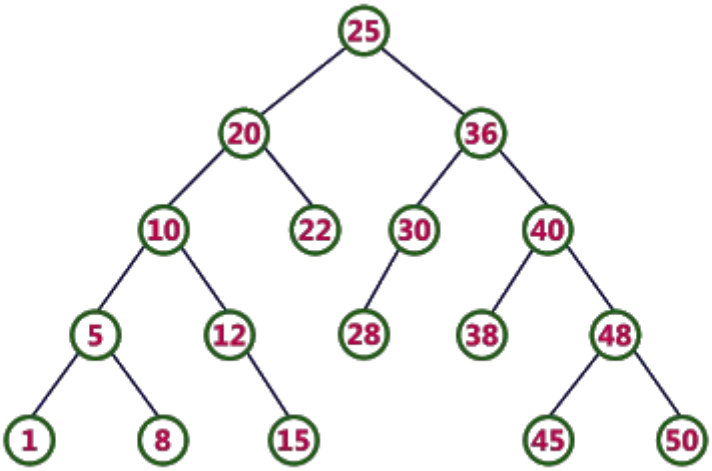
Ubalancen i træet er ved node 13, hvor højden af det højre subtræ er 3, og det venstre subtræ er 0.

Hvordan kan man omarrangere noderne i det højre subtræ, så hele træet bliver et AVL-træ?

Kunne træet have været et AVL-træ før den seneste operation (insert eller delete, men ikke rotation)? Eksempler på seneste operation kunne være indsættelse af node 3 eller sletning af node 12 (venstre barn af node 13). Begrund dit svar.

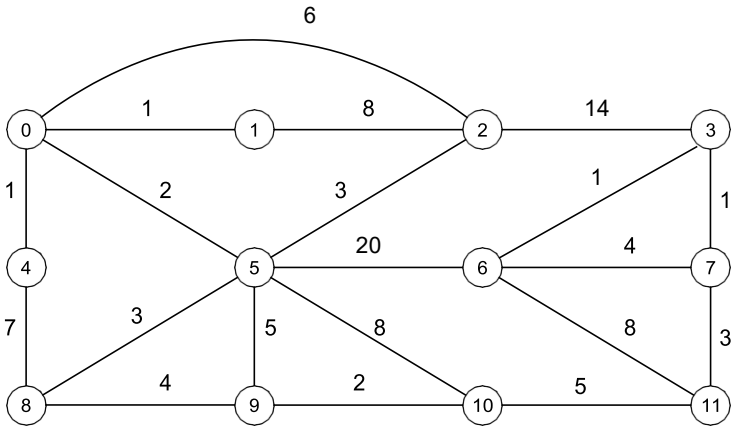
Øvelse 4

Nedenfor ses et binært søgetræ:



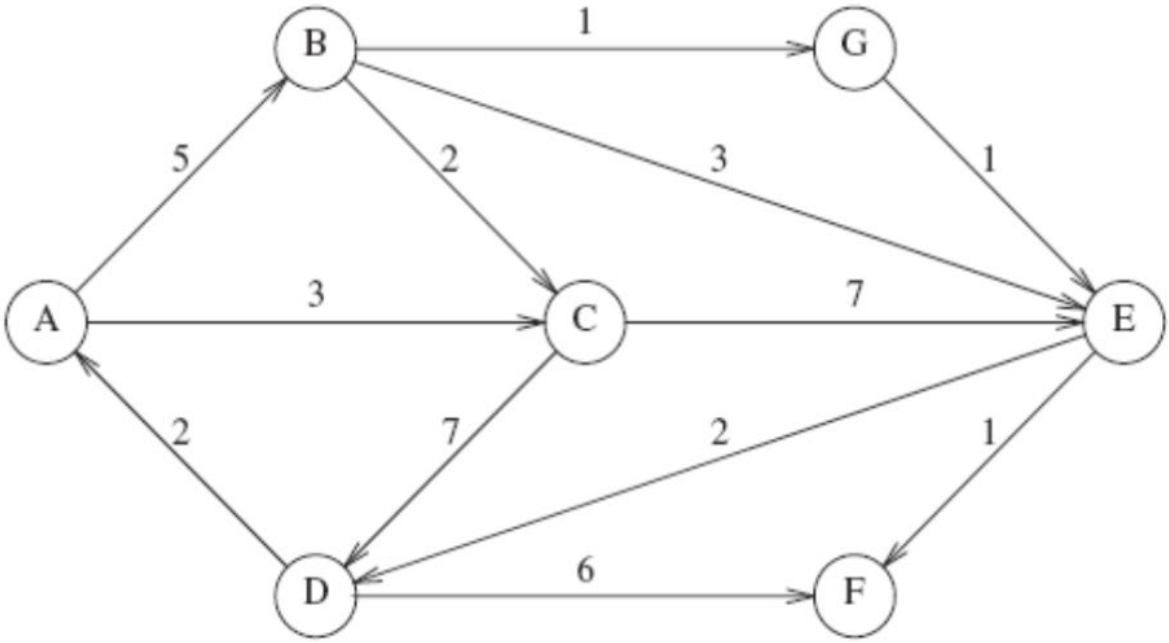
List rækkefølgen i hvilken noderne besøges i en post order og i en pre order traversering.
Hvad er træets internal path length?
Er det et AVL-træ? Hvorfor eller hvorfor ikke?

Øvelse 5



Find et *minimum spanning tree* for ovenstående graf. Svaret skal være en liste af edges/kanter, der viser i hvilken rækkefølge algoritmen vil etablere forbindelser mellem noderne. Angiv også træets totale væg samt hvilken algoritme, du har brugt.

Øvelse 6



Nedenstående tabel viser startkonfigurationen for en traversering af ovenstående graf med anvendelse af Dijkstras algoritme. Vis slutkonfigurationen for en traversering startende i node/vertex *A*. Du behøver ikke at vise mellemkonfigurationerne.

v	**known**	**d_v**	**p_v**
A	false	0	0
B	false	∞	0
C	false	∞	0
D	false	∞	0
E	false	∞	0
F	false	∞	0
G	false	∞	0