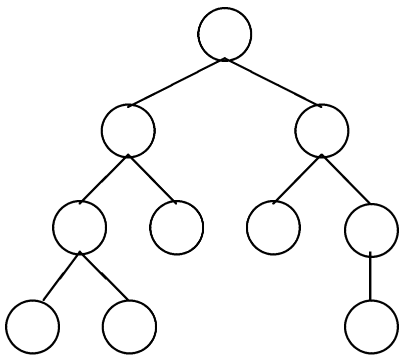
Opgave 1 (20%)

Et binært træ kan som bekendt repræsenteres som en værdi af typen:

Type Tree = **Prod**(left, right: Tree)

hvor et tomt træ angives af standardværdien ?-Tree.

Vi definerer *fuldstændigheden* af et binært træ som den mindste afstand fra roden til en knude, der ikke har to sønner; for et tomt træ defineres fuldstændigheden som ?-Int. Fx har fuldstændigheden af træet:



værdien 2, og et træ med kun en enkelt knude har fuldstændighed 0.

Skriv en procedure:

Proc Fuldstændighed[T: Tree] → (Int)

der beregner fuldstændigheden af træet T. Der lægges vægt på, at besvarelsen er letlæselig, detaljeret og korrekt.

Opgave 2 (20%)

En tekst er som bekendt et *palindrom*, hvis den er lig med sin spejling. Fx er den tomme tekst og teksterne `"a"`, `"cc"`, `"pip"` og `"abba"` alle palindromer.

Betragt følgende algoritme:

Algoritme: Palindrom
Stimulans: t: Text
Respons: pal: Bool, pal $= \forall j \in 0..|t|/2 : t.(j) = t.(|t| - j - 1)$
Metode:

```
i:=0
do { I }
    (i< | t |/2) ∧ (t.(i) = t.(| t |-i-1)) → i:=i+1
od
pal:= (i = | t |/2)
```

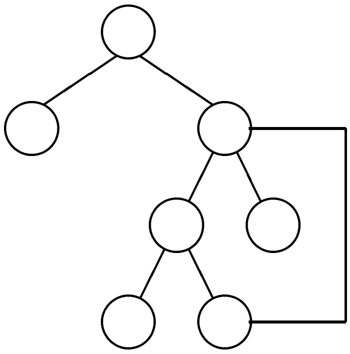
> Bevis, at algoritmen Palindrom er gyldig og korrekt. Vink: start med at definere en passende invariant.

Opgave 3 (20%)

I en vilkårlig ikke-orienteret vægtet graf med n knuder og m kanter kan man som bekendt finde det letteste udspændende træ ved hjælp af Prims eller Kruskals algoritmer i tid $O((n+m)\log n)$ eller $O(m \log m)$.

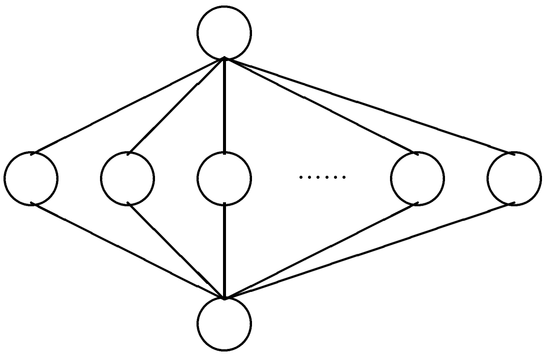
I denne opgave skal vi se, hvordan man kan udnytte egenskaberne ved nogle specielle grafer til at opnå forbedrede tidskompleksiteter.

Et *løkke-træ* er en ikke-orienteret vægtet graf, der har form som et binært træ i hvilket netop et af bladene har en ekstra kant til en anden knude:



a) Angiv en algoritme, der i tid $O(n)$ finder det letteste udspændende træ for et løkke-træ med n knuder. Argumentér for algoritmens korrekthed og effektivitet. Bemærk: grafen er givet i form af en almindelig kantlisterepræsentation uden ekstra information; det vides således ikke på forhånd, hvilken knude der er "roden"..

En *pol-graf* er en ikke-orienteret vægtet graf, der består af to såkaldte poler adskilte af et lag af parvist forbundne kanter:



b) Angiv en algoritme, der i tid $O(n)$ finder det letteste udspændende træ for en pol-graf med n knuder. Argumentér for algoritmens korrekthed og effektivitet. Bemærk at grafen igen er givet form af en almindelig kantlisterepræsentation uden ekstra information.

Opgave 4 (20%)

Vi skal i denne opgave undersøge, hvordan RASMUS-operatoren *faktor* kunne implementeres i TRINE. En RASMUS-relation med skema:

a:Int	b:Int	c:Int	d:Int

kan som bekendt repræsenteres som en værdi af TRINE-typen:

Type Rel = **List**(**Prod**(a, b, c, d: Int))

En beregning af RASMUS-udtrykket:

```
!(R)|a,b: e
```

starter med at beregne de relevante faktorpar, der i TRINE kunne repræsenteres som en værdi af typen:

Type Fak = **List**(**Prod**(a, b: Int, r: **List**(**Prod**(c, d: Int))))

Betragt relationen:

a:Int	b:Int	c:Int	d:Int
1	1	3	4
1	2	3	4
2	1	6	7
2	1	3	4
1	1	3	5
2	2	3	4

a) Angiv hvorledes faktorparrene for denne relation ser ud som værdi af type Fak.

b) Skitser er effektiv algoritme, der for en relation af type Rel beregner faktorparrene af type Fak. Hvad bliver tidskompleksiteten?

Opgave 5 (20%)

En sædvanlig ækvivalensrelation over mængden $0 \ldots N-1$ understøtter som bekendt operationerne **Init**[R], **Equiv**[R](x_1,x_2) og **Join**[R](x_1,x_2). Vi vil kalde en ækvivalensklasse *atomar*, hvis den kun indeholder et enkelt element. Vi ønsker at udvide denne datastruktur med nogle ekstra operationer:

- **Size**[R] der returnerer antallet af ækvivalensklasser i R;
- **Max**[R] der returnerer størrelsen af den største ækvivalensklasse i R;
- **Asize**[R] der returnerer antallet af atomare ækvivalensklasser i R; og
- **Amin**[R] der returnerer det mindste element i $0 \ldots N-1$, der udgør en atomar ækvivalensklasse i R.

a) Giv en formel specifikation af operationen **Amin**.

b) Angiv en modificeret implementation af ækvivalensrelationer, således at **Init**, **Equiv** og **Join** bevarer tidskompleksiteterne $O(N)$, $O(\log N)$ og $O(\log N)$, og at **Size**, **Max**, **Asize** og **Amin** alle får tidskompleksiteter i $O(1)$.