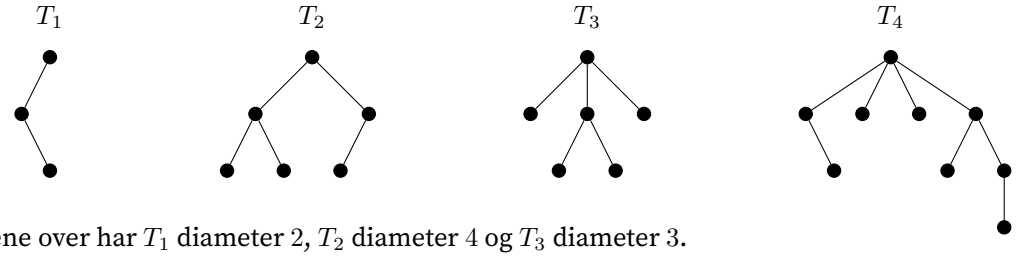
Diameteren til et tre 12 poeng

Vi definerer *diameteren* til et tre som lengden til den lengste stien mellom to noder.



I eksemplene over har T_1 diameter 2, T_2 diameter 4 og T_3 diameter 3.

(a) Hva er diameteren til T_4?

> Svaret er 5. Det gis 1 poeng for riktig svar.

I de neste deloppgavene begrenser vi oss til *binære* trær. Hvis v er en node, så er:

- v.`left` venstre barn av v
- v.`right` høyre barn av v

(b) Vil den lengste stien i et binært tre alltid gå gjennom rotnoden? Begrunn svaret.

> Nei. Se for deg et komplett binærtre med mange noder; der vil den lengste stien gå gjennom rotnoden. Hvis du lager en ny rotnode som har det komplette binærtreet som sitt venstre subtre, så vil den lengste stien gå gjennom de samme nodene, og ikke gjennom den nye rotnoden.
>
> - Opptil 2 poeng for hele deloppgaven.
> - Det gis 1 poeng for riktig svar.
> - Det gis 1 poeng for en god begrunnelse.

(c) Skriv en prosedyre som finner diameteren til et gitt *binærtre*.

```
Input: Rotnoden v av et binærtre
Output: Returnerer diameteren til treet
1 Procedure Diameter(v)
  | // ···
```

Lavere kjøretidskompleksitet er mer poenggivende.

Hvis vi først antar at rotnoden er med i den lengste stien, så vil den lengste stien gå fra den dypeste noden i det venstre subtreet, via rotnoden, og ned til den dypeste noden i det høyre subtreet. Høyden av et subtre angir antall kanter til den dypeste noden i treet.

Hvis vi antar at h_L og h_R er høydene til venstre og høyre subtre av rotnoden, så vil diameteren være gitt av $h_L + h_R + 2$.

Mer generelt vil den lengste stien som går gjennom en node være gitt av $h_L + h_R + 2$ gi (der vi antar at h_L og h_R er høydene av den aktuelle noden).

Altså kan problemet løses ved å for hver node beregne lengden på den lengste stien den inngår i, og til slutt returnere den lengste av disse lengdene. Dette kan gjøres i lineær tid.

```
1  Procedure Diameter(v)
2      d ← 0
3
4      Procedure DiameterRec(v)
5          if v = null then
6              return −1
7          h_L ← DiameterRec(v.left)
8          h_R ← DiameterRec(v.right)
9          d ← max(d, h_L + h_R + 2)
10         return 1 + max(h_L, h_R)
11
12     DiameterRec(v)
13     return d
```

Her er problemet løst ved å gå gjennom treet én gang. Den definerer en indre prosedyre for å unngå å bruke en global variabel, men dette er helt uvesentlig med tanke på vurdering av oppgaven.

Det er muligens enklere å løse dette ved å dele opp problemet i flere steg. For eksempel kan høyden i hver node beregnes først, deretter en lokal diameter for hver node, og til slutt løpe over alle nodene og finne den største diameteren. Ingen slike avveininger har noen påvirkning på poenggivningen.

Det kan være fort gjort å ende opp med en kvadratisk løsning.

- Opptil 8 poeng for en løsning i lineær tid.
- Opptil 5 poeng for en løsning i kvadratisk tid.
- Opptil 3 poeng for gode forsøk, men som ikke vil fungere.

(d) Oppgi kjøretidskompleksiteten på algoritmen din for et binærtre med n noder.

1 poeng for riktig kjøretidskompleksitet avhengig av svaret som ble gitt i forrige deloppgave. Det kan gis poeng selv hvis det er store mangler i forrige deloppgave.