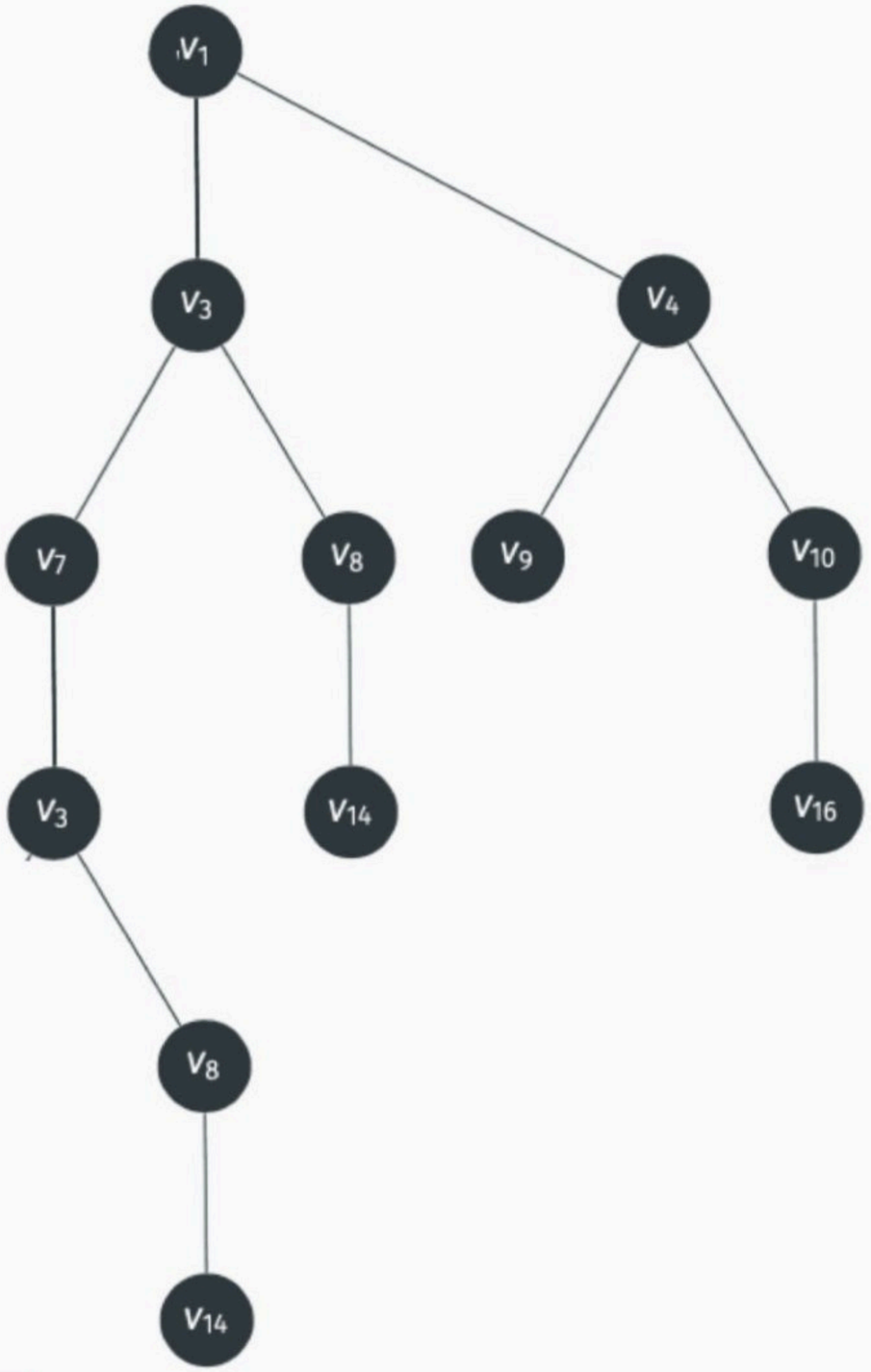
I dag skal vi repetere...

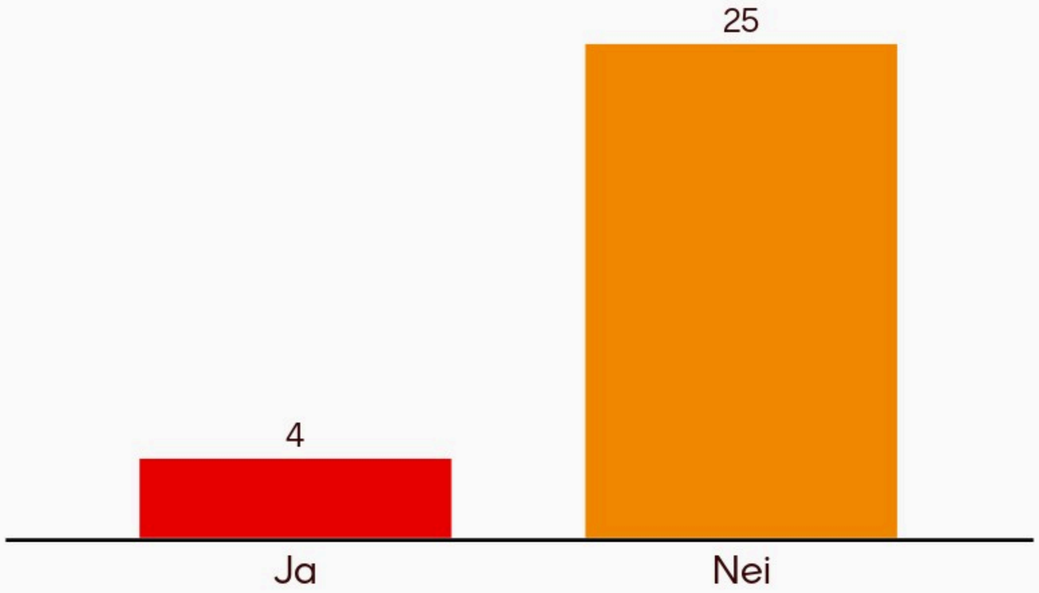
- Trær
- Binærsøk
  - (Binære trær)
- Binære søketrær
- Balanserte binære søketrær

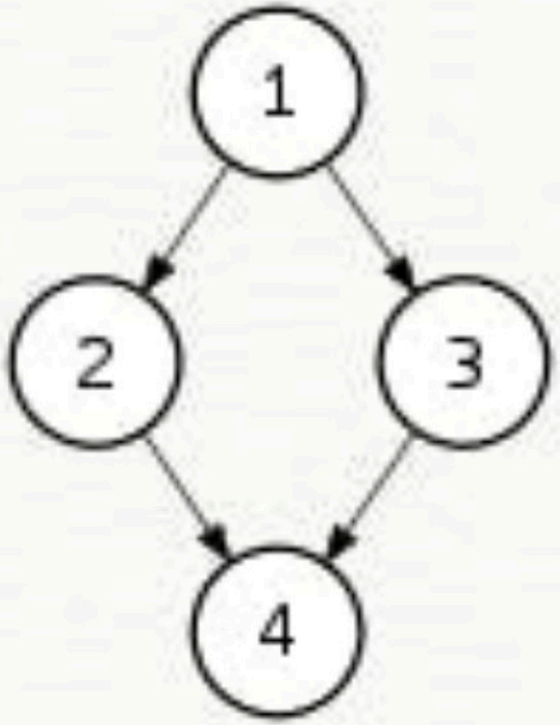


Trær - Definisjon

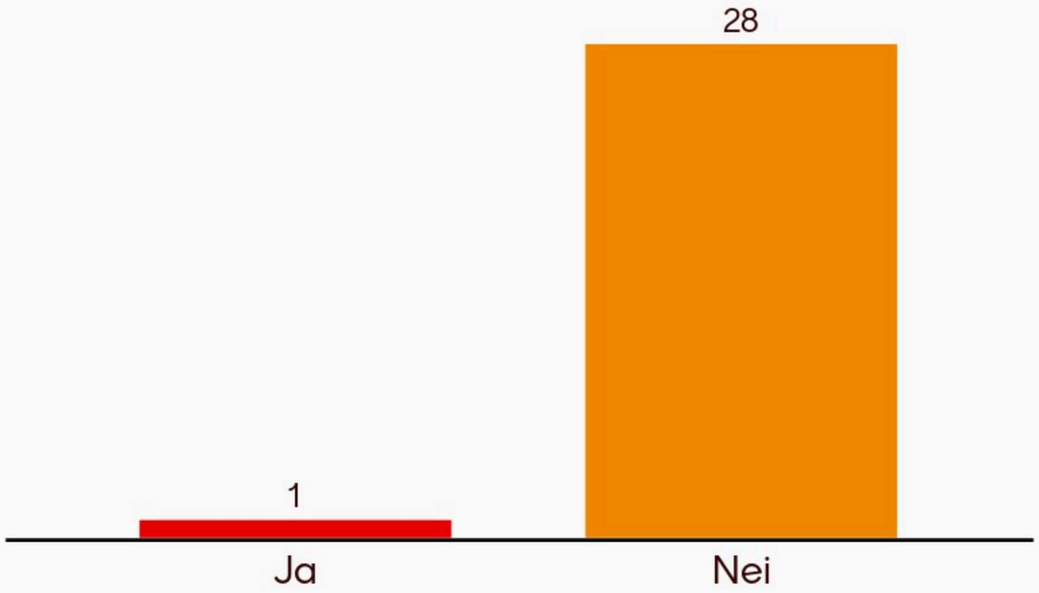
- Det tomme treet (null)
- *eller*
- En node med:
  - En peker til data/element
  - 0 eller flere pekere til barnenoder
  - Med nøyaktig én forelder (utenom rota)
- Et tre kan ikke inneholde sykler

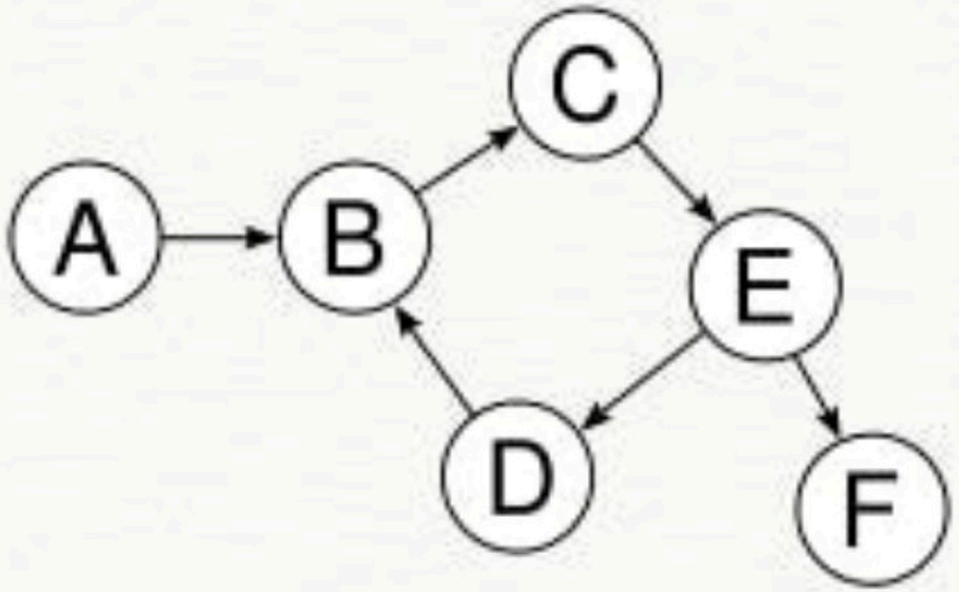
Er det et tre?





Er det et tre?



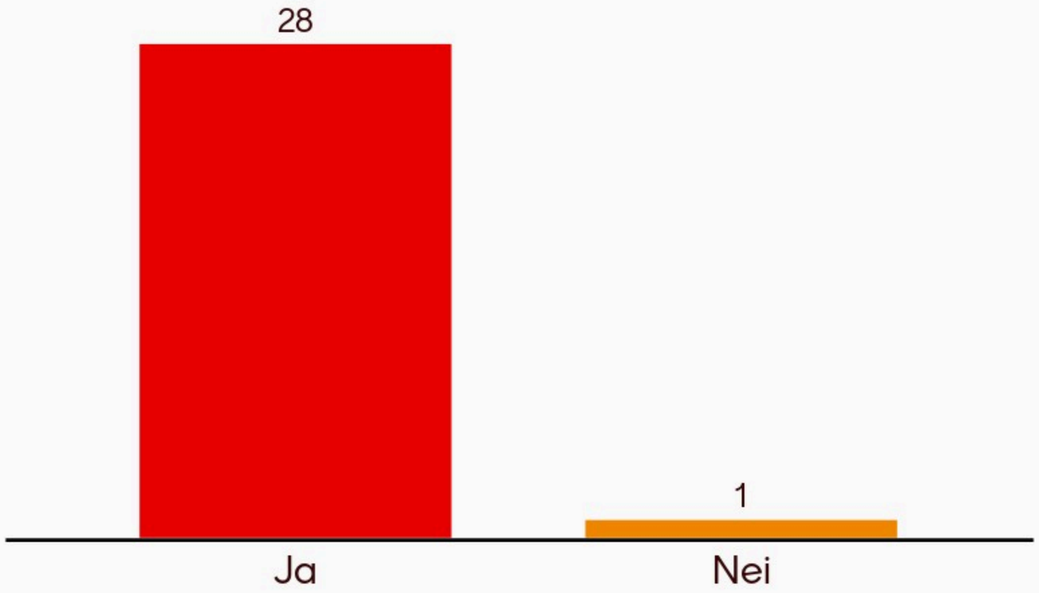


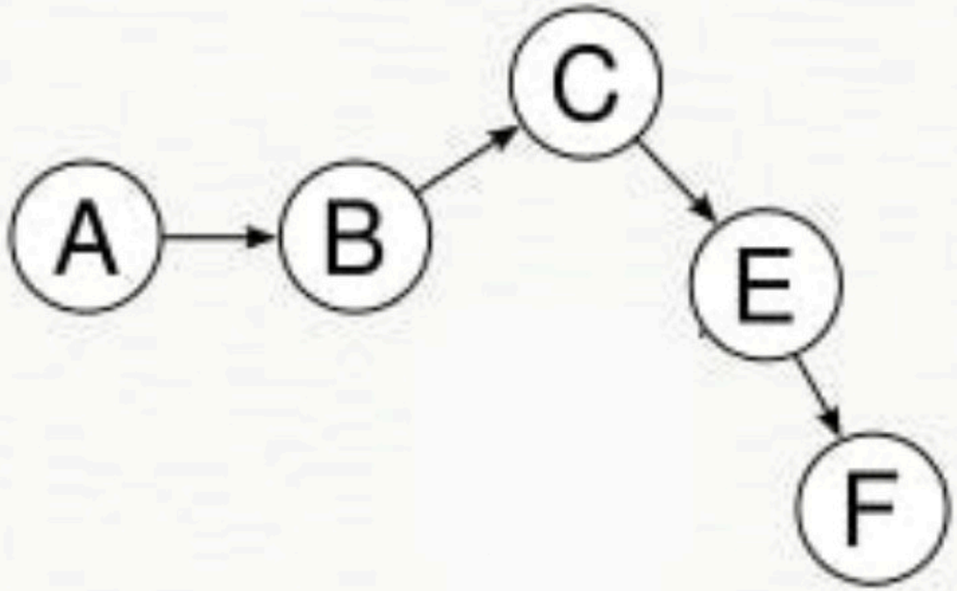
Hva får vi dersom vi legger til en sykel i et tre? (Som i forrige oppgave)

IN2010

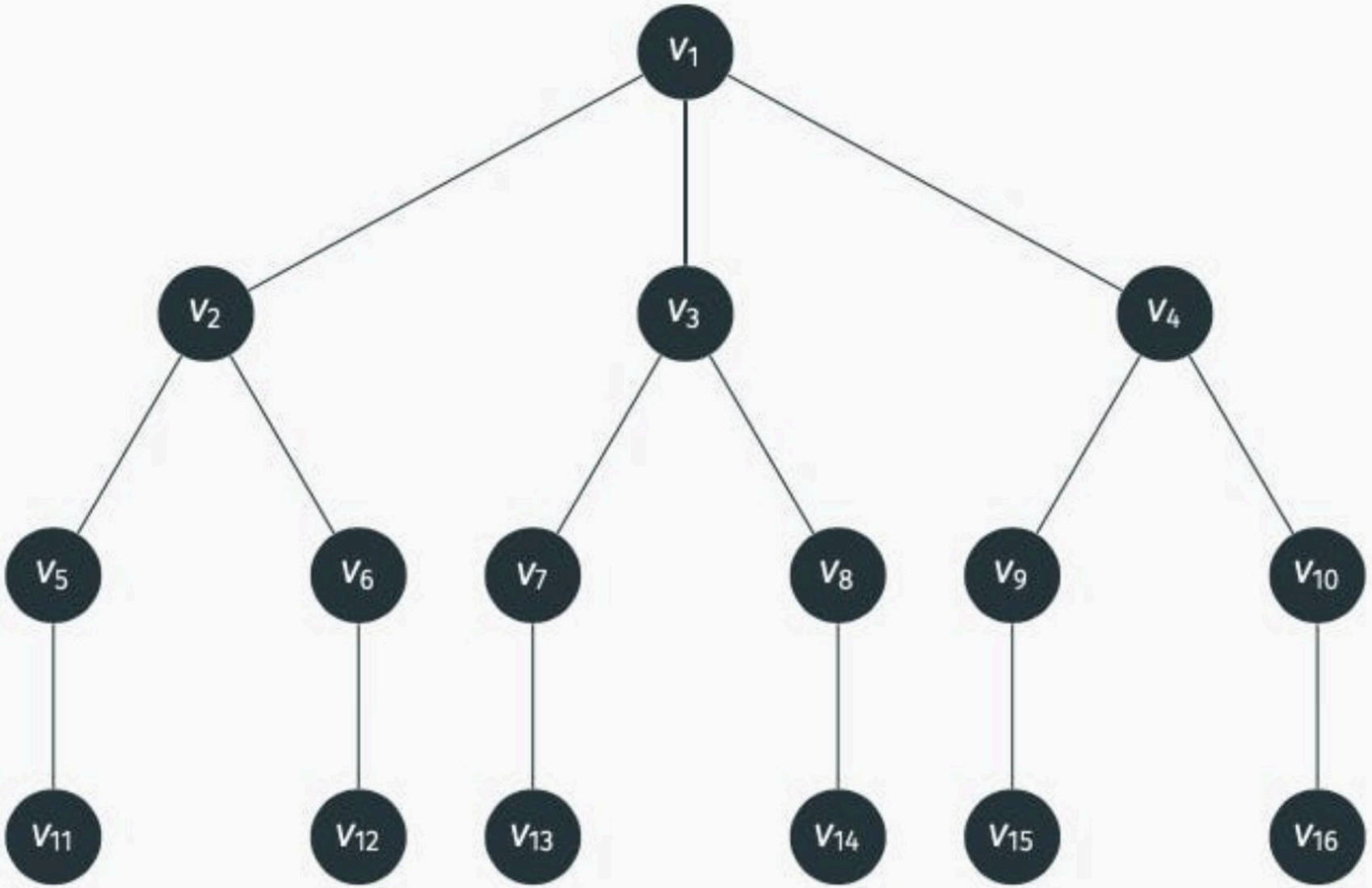
Graf	Graf	graf
graf	En garf	En graf
Uendelig tre	graf	Graf

Er det et tre?

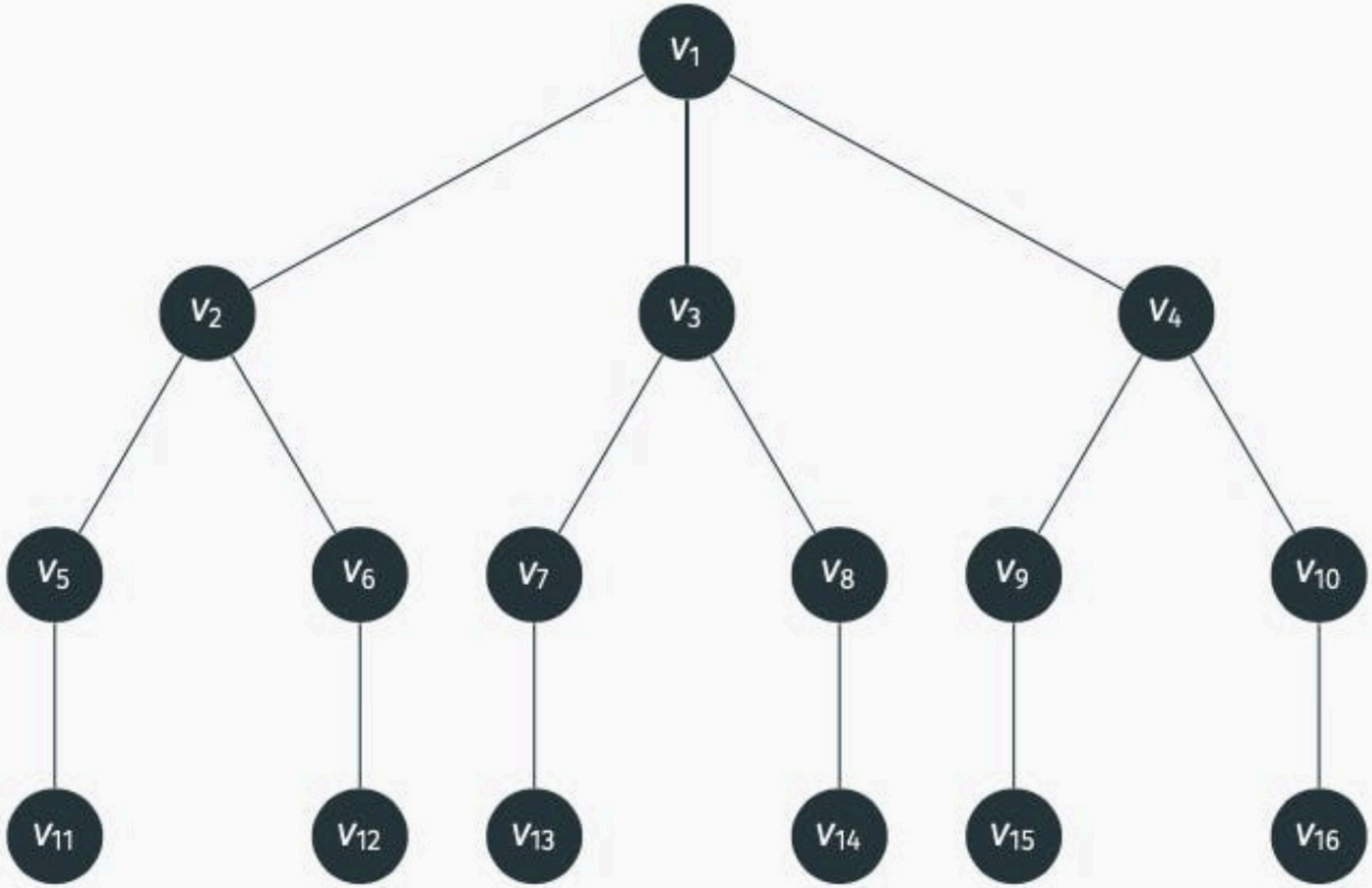




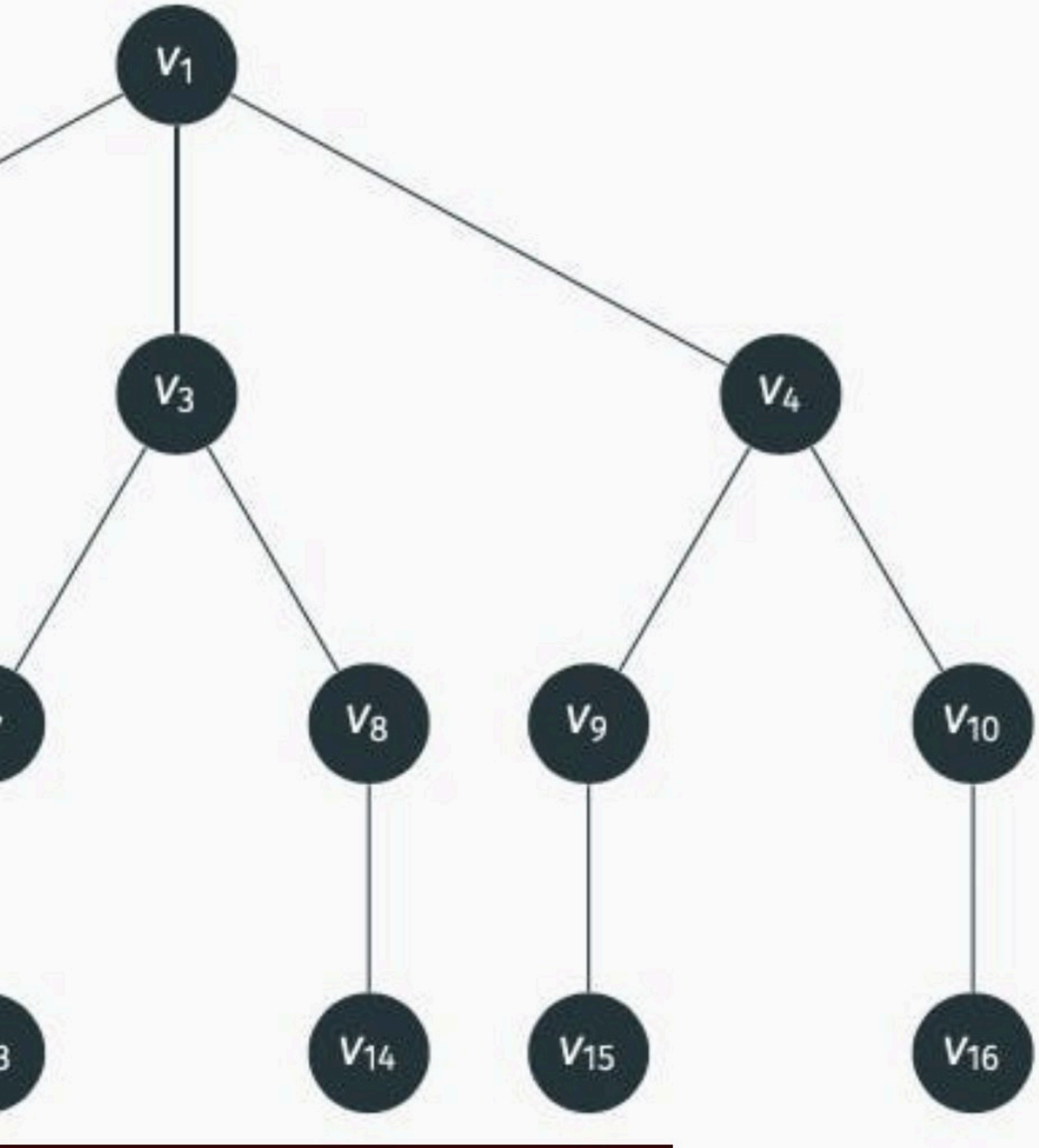
Trær - Terminologi



Rot, barn, forelder, søsken, løvnode (ekstern node), intern node

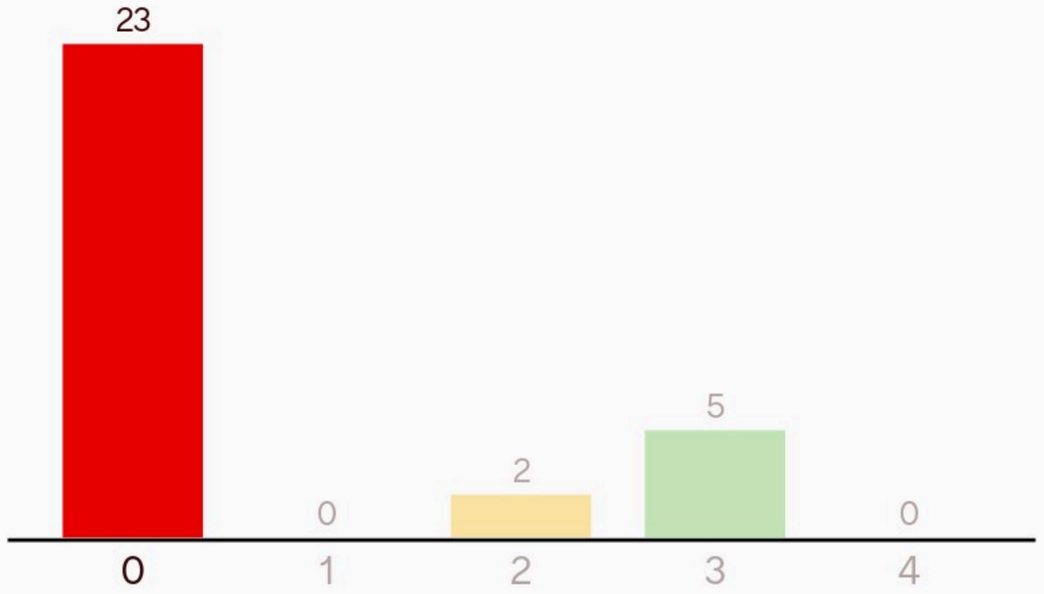


Subtre, forfedre, etterkommere, dybde, høyde



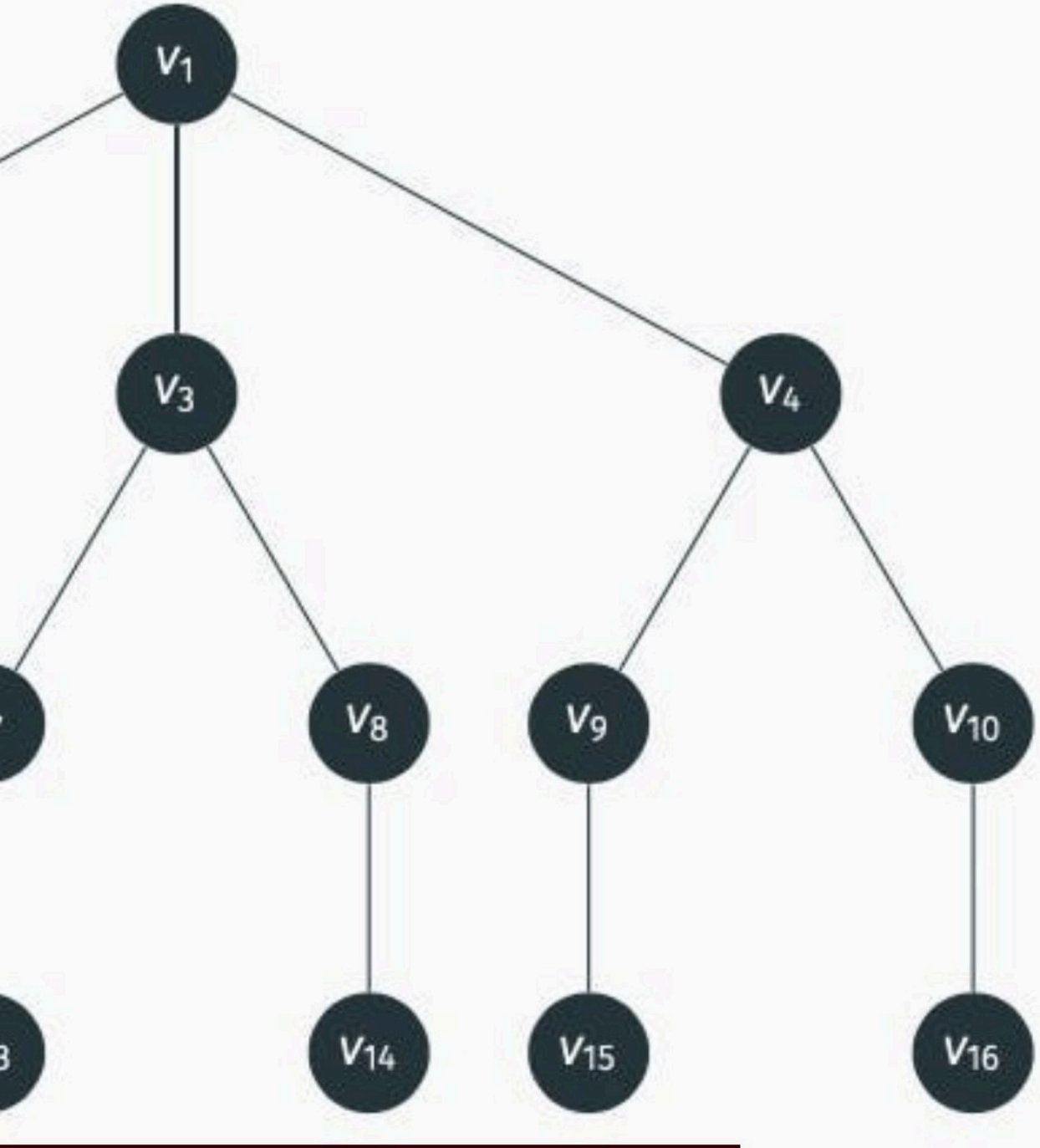
Hva er dybden til V1/roten?

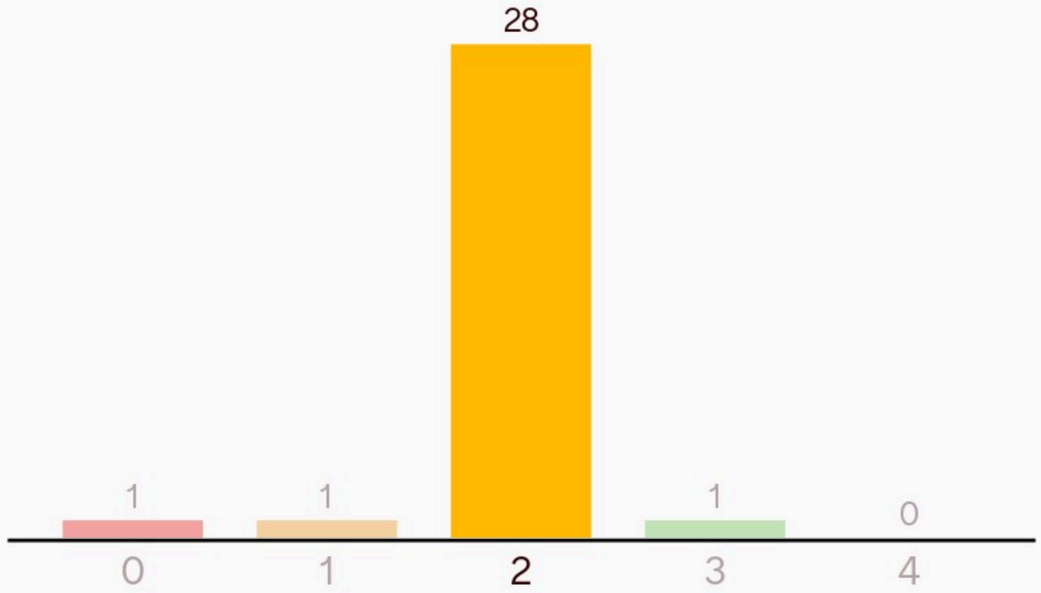
IN2010

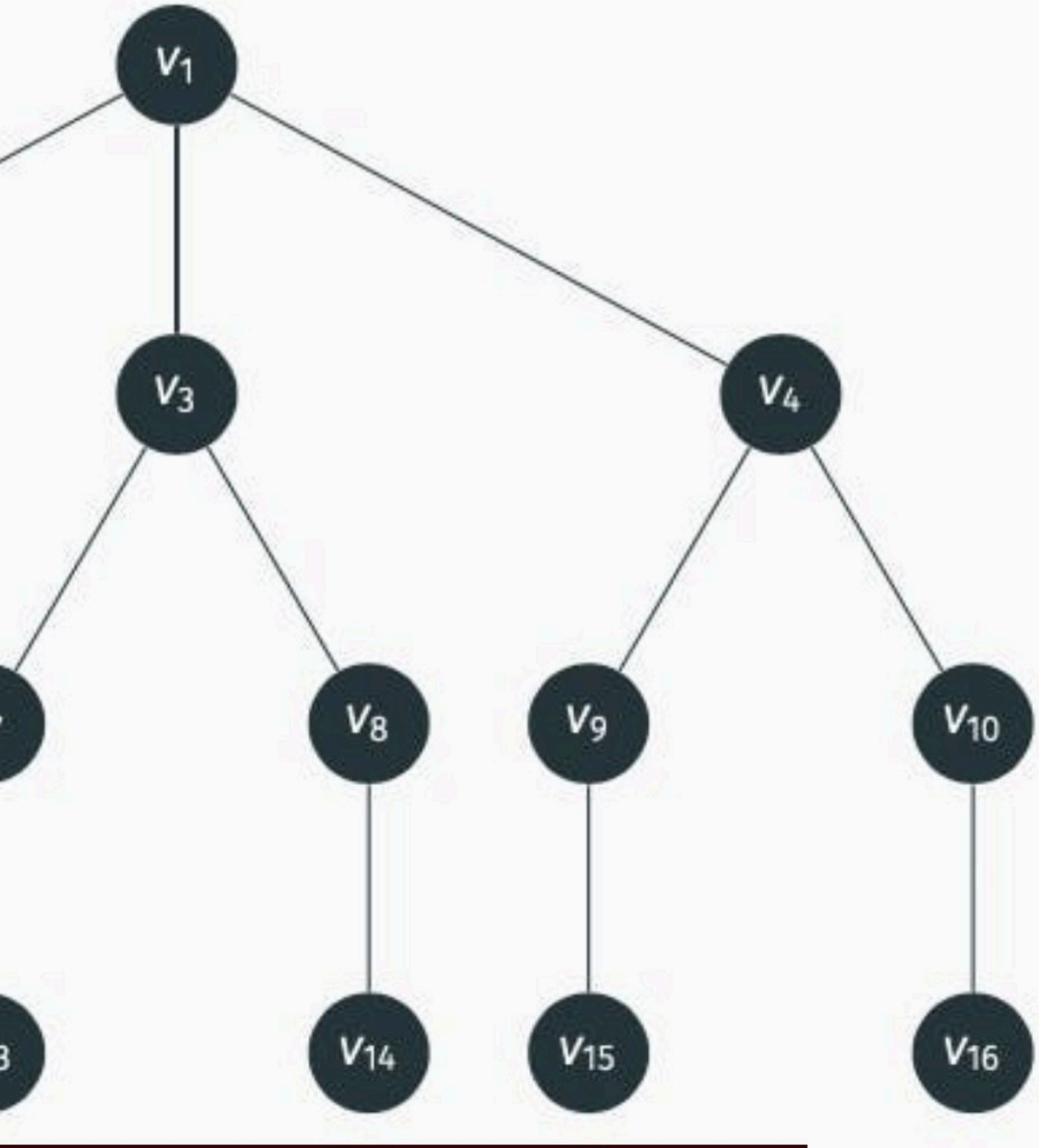


Hva er dybden til V9?

IN2010

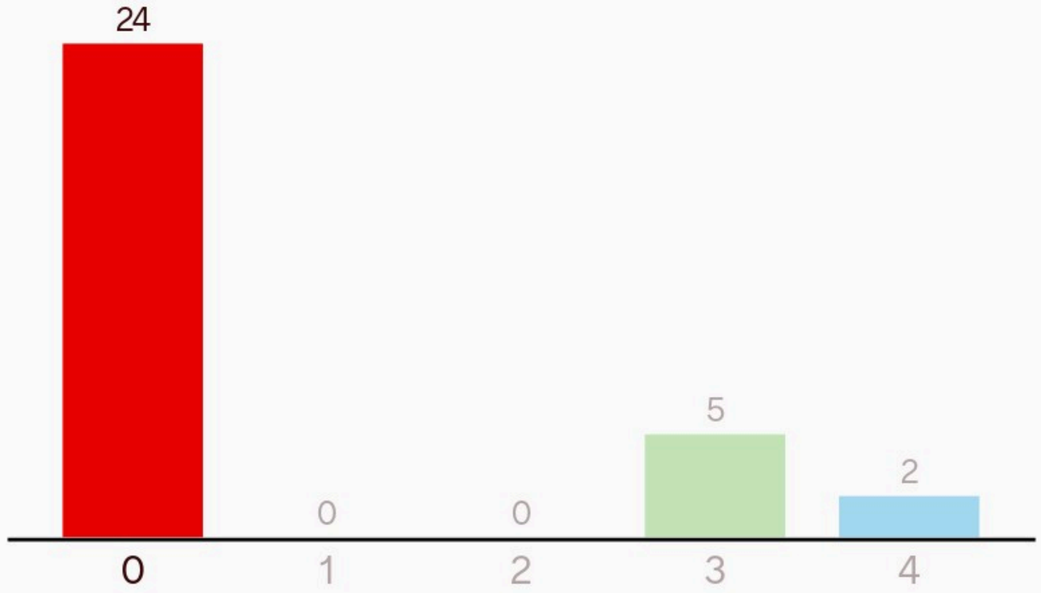






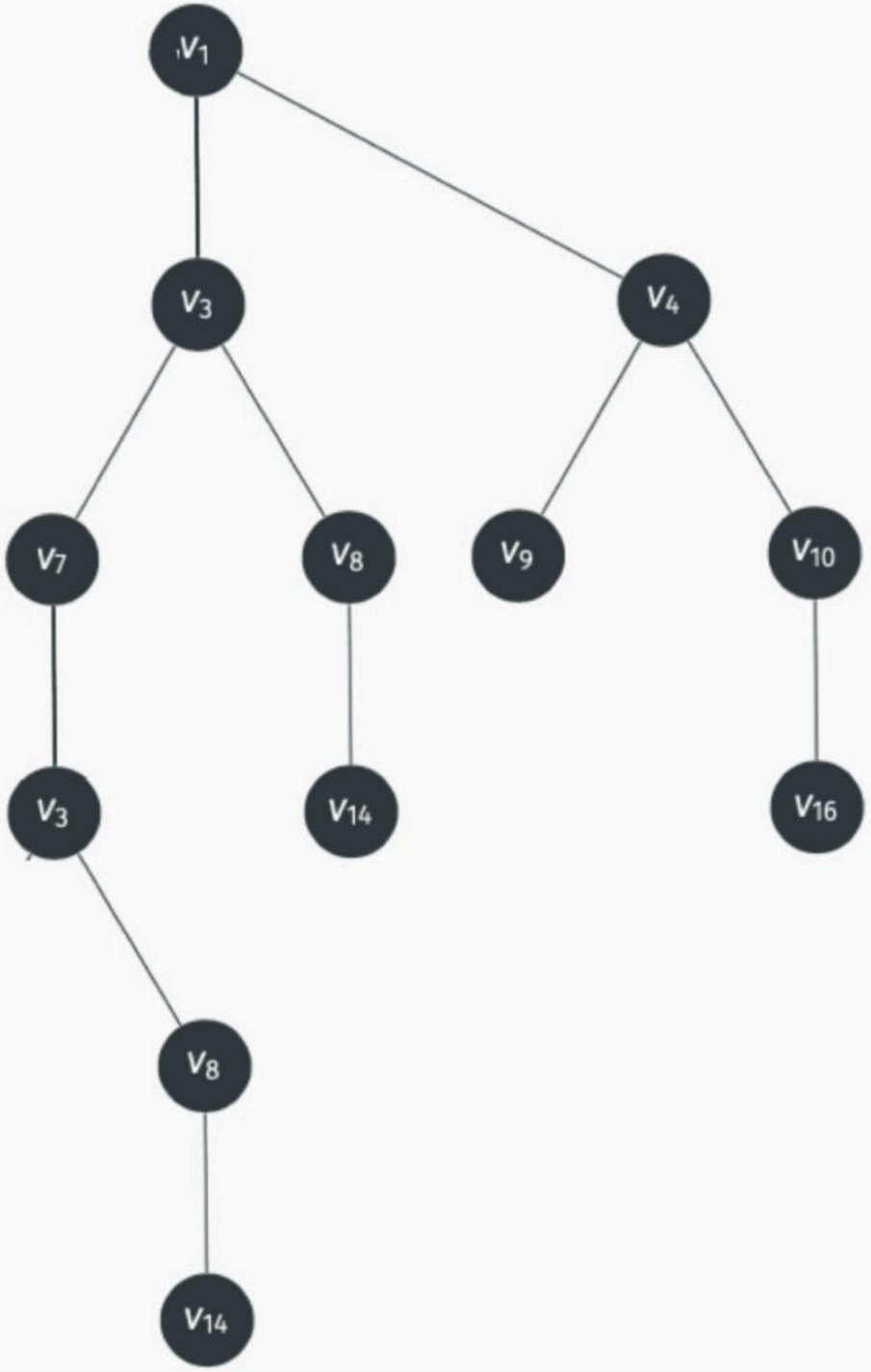
(Obs: Ikke dybde lenger!) Hva er høyden til V15?

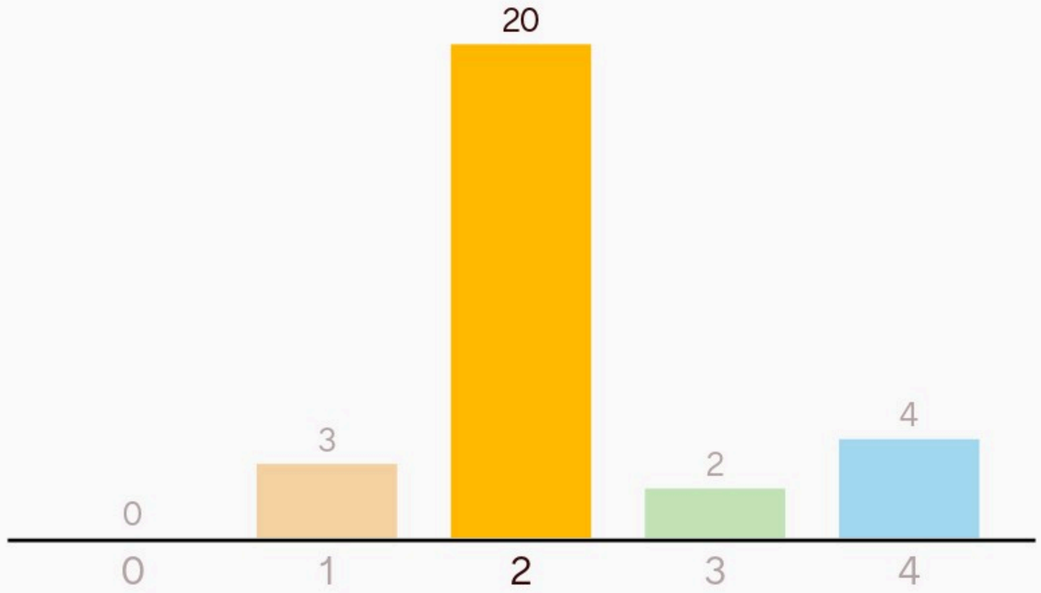
IN2010

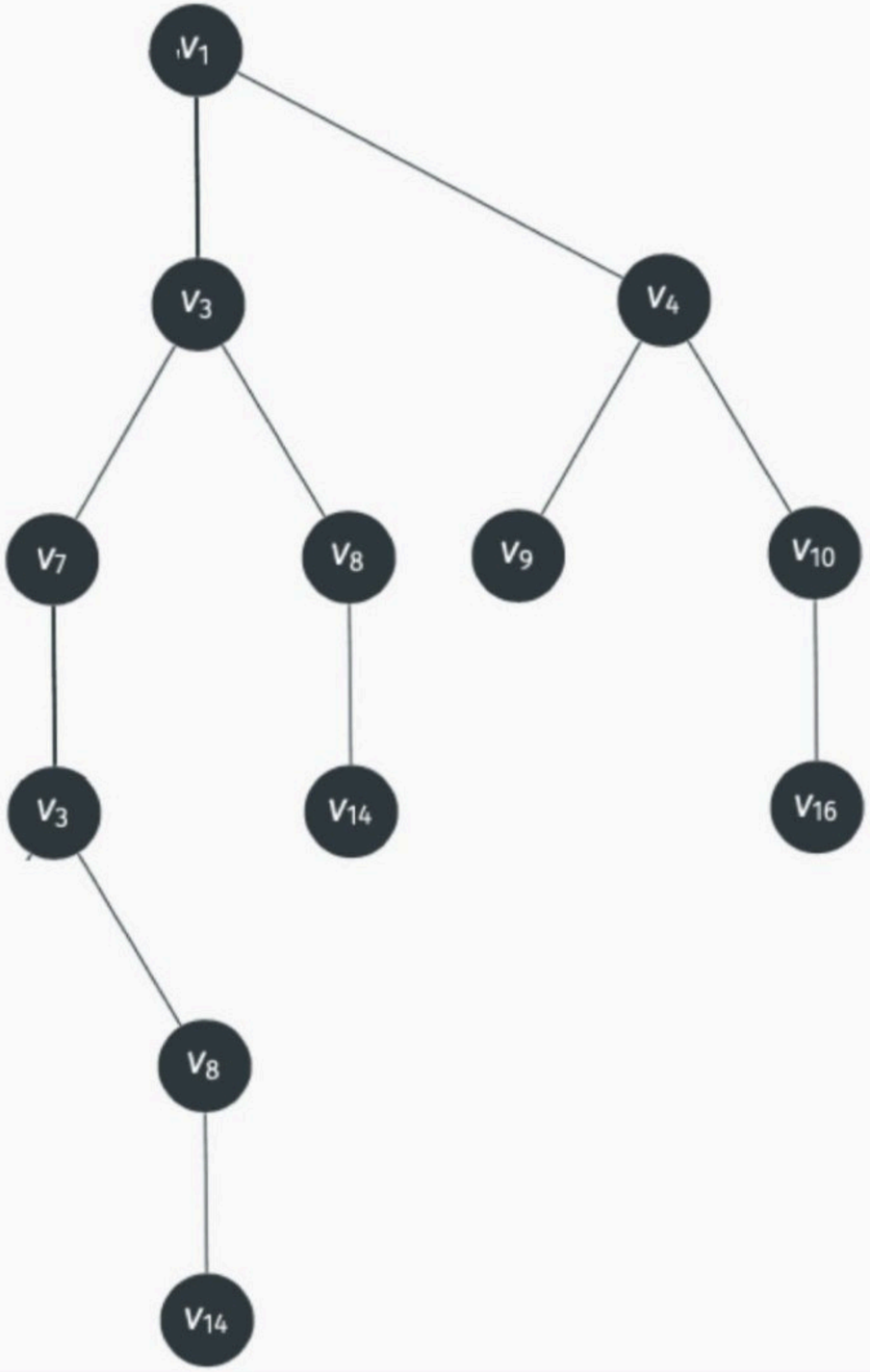


Hva er høyden til V4?

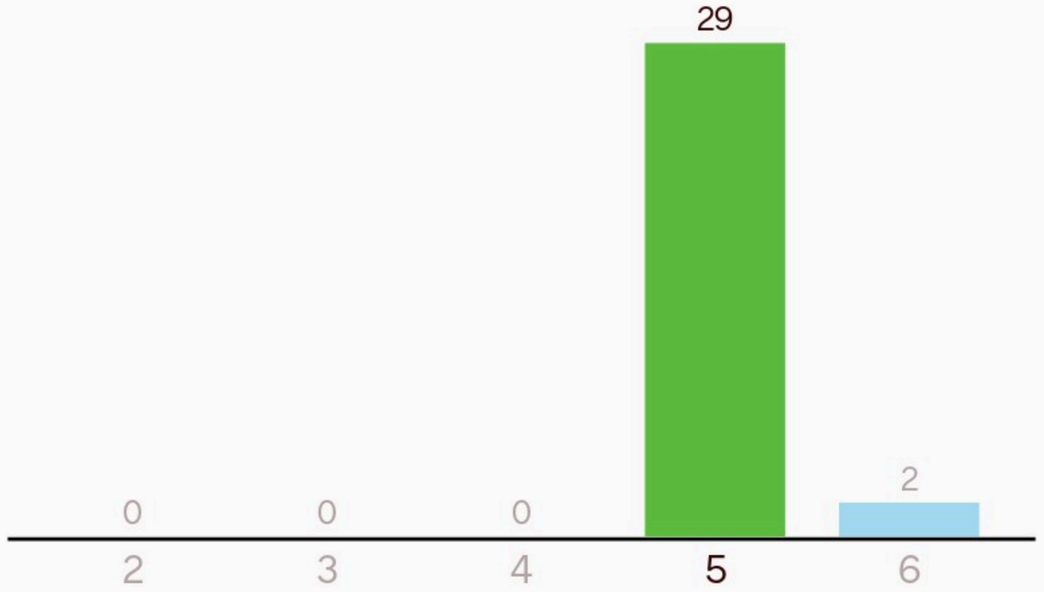
IN2010







Hva er høyden til treet?



Algorithm 9: Preorder traversering

Input: En node v (som ikke er **null**)
Output: Utfør en operasjon på v først og barna til v etterpå

```
Procedure Preorder(v)
    Operate on v
    for v' ∈ v.children do
        Preorder(v)
    end
```

Algorithm 10: Postorder traversering

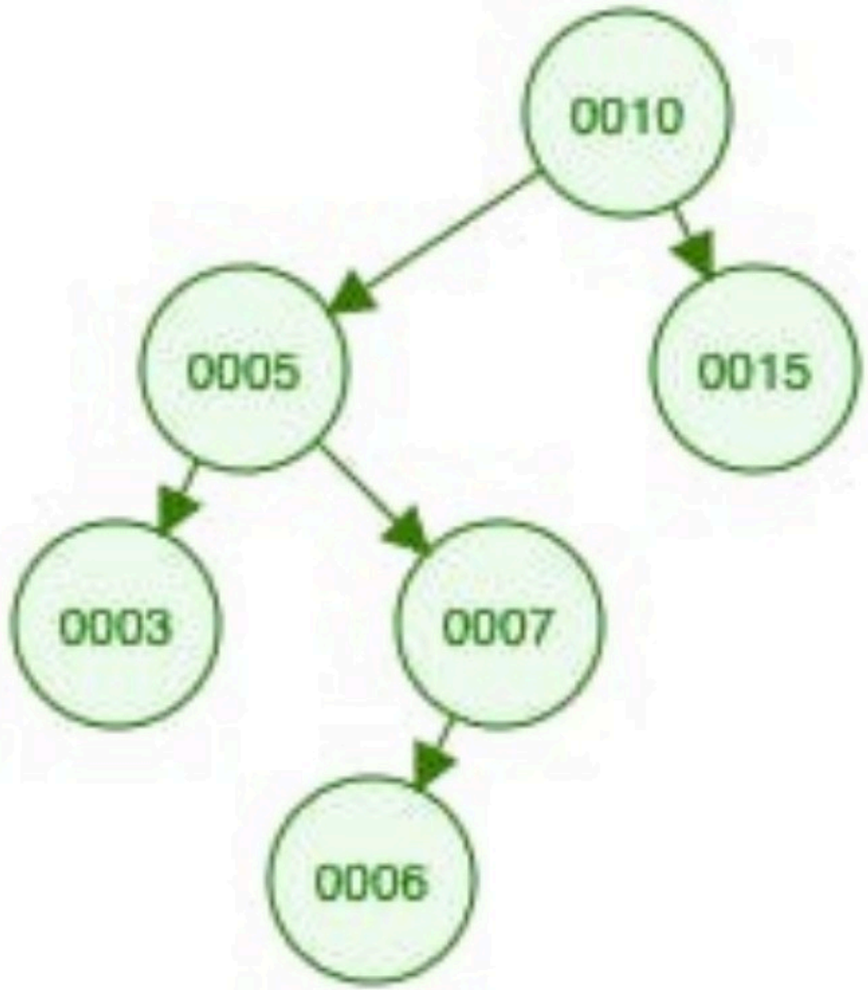
Input: En node v (som ikke er **null**)
Output: Utfør en operasjon på barna til v først og v etterpå

```
Procedure Postorder(v)
    for v' ∈ v.children do
        Postorder(v)
    end
    Operate on v
```

Traversere trær

Litt som DFS: går så langt ned vi klarer, før vi går en ny vei

To rekkefølger å utføre operasjoner i:
- Preorder: Seg selv først, og så barna
- Postorder: Barna først, så seg selv

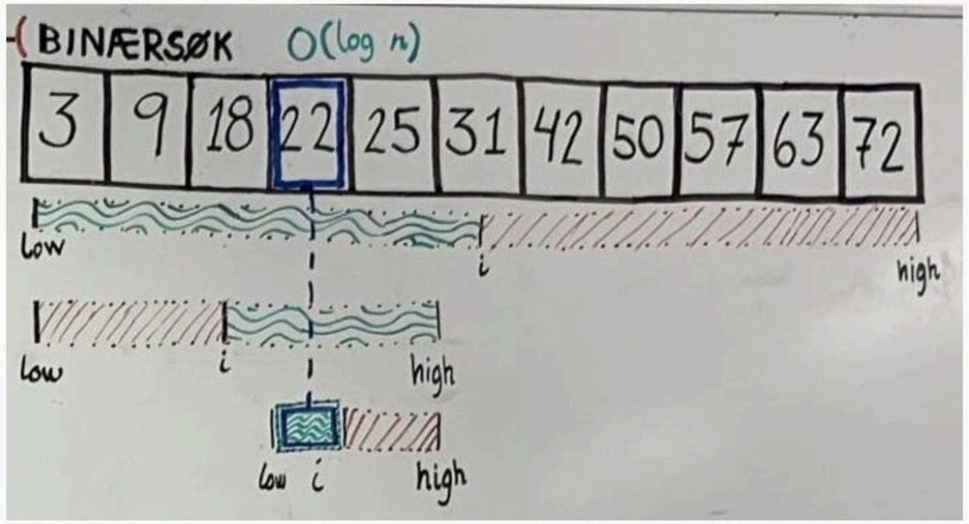


Oppgave

Hvordan rekkefølge skrives tallene ut i når du printer dette treet ved hjelp av:

- Preorder (seg selv først)
- Postorder (barn først)

Binærsøk



Input: Sortert liste, elmentet vi leter etter
Output: True/False

Framgangsmåte:
- Se på det som ligger i midten av arrayet
- Let videre på riktig side av arrayet
- Forsett helt til du finner elementet / False

O(log(n)), pga. halveringene

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Alg: Binærsøk
Input: Et ordnet array A, element x
Output: Hvis x er i arrayet A, return true ellers false
-------------------------------------------------------------------------------
def Binærsøk(A, x):
    low = 0
    high = A.length - 1

    while low <= high do:
        i = (low + high) / 2 #Rundes av til heltall

        # Vi har funnet den vi leter etter
        if A[i] == x then:
            return true

        # Vi er for langt til venstre, flytter low oppover
        else if A[i] < x then:
            low = i + 1

        # Vi er for langt til hoyre, flytter high nedover
        else if A[i] > x then:
            high = i - 1
    end
    return false
-------------------------------------------------------------------------------
```

Pseudokode for binærsøk

Binære *søke*trær

- Binære trær: Maks 2 barn
- Mindre til venstre, større til høyre
- Binære søketrær vs. binærsøk
  - Binærsøk: algoritme
  - Binær søketre: datastruktur

Innsetting (og oppslag)

Gå ned gjennom treet til du finner en plass å putte inn det nye elementet

Gå til venstre om elementet er mindre enn den noden vi ser på nå, og til høyre om den er større.

O(høyden på treet) => O(n)

```
def Insert(v, x):
    # Setter inn node
    if v == null then:
        return new Node(x)

    # Node skal settes inn i venstre sub-tre
    else if x < v.data then:
        v.left = Insert(v.left, x)

    # Node skal settes inn i høyre sub-tre
    else if x > v.data then:
        v.right = Insert(v.right, x)

    return v
```

```
def Search(v, x):
    # Har enten funnet elementet,
    # eller funnet ut at det ikke er i treet
    if v == null then:
        return null
    else if x == v.data then:
        return v

    # Finner elementet i venstre sub-tre
    if x < v.data then:
        return Search(v.left, x)

    # Finner elementet i høyre sub-tre
    else if x > v.data then:
        return Search(v.right, x)
```

Visualisering

https://www.cs.usfca.edu/~galles/visualization/BST.html

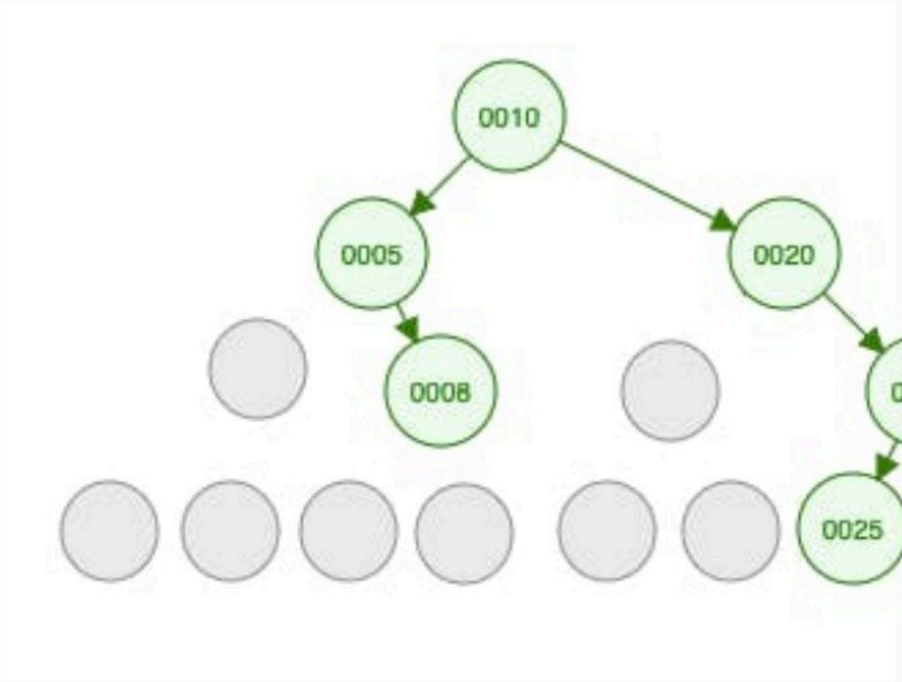
Algorithm 14: Slett en node i et binært søketre

Input: En node v og et element x

Output: Dersom x forekommer i en node u som en etterkommer av v, fjern u.

```
1  Procedure Remove(v, x)
2      if v = null then
3          return null
4      if x < v.element then
5          v.left ← Remove(v.left, x)
6          return v
7      if x > v.element then
8          v.right ← Remove(v.right, x)
9          return v
10     if v.left = null then
11         return v.right
12     if v.right = null then
13         return v.left
14     u ← FindMin(v.right)
15     v.element ← u.element
16     v.right ← Remove(v.right, u.element)
17     return v
```

Sletting



Balanserte binære søketrær

- Lærer to typer: AVL og rød-svarte
- Vi minsker høyden på treet
- ... sånn at vi kan på O(log(n)) på alle operasjoner

AVL

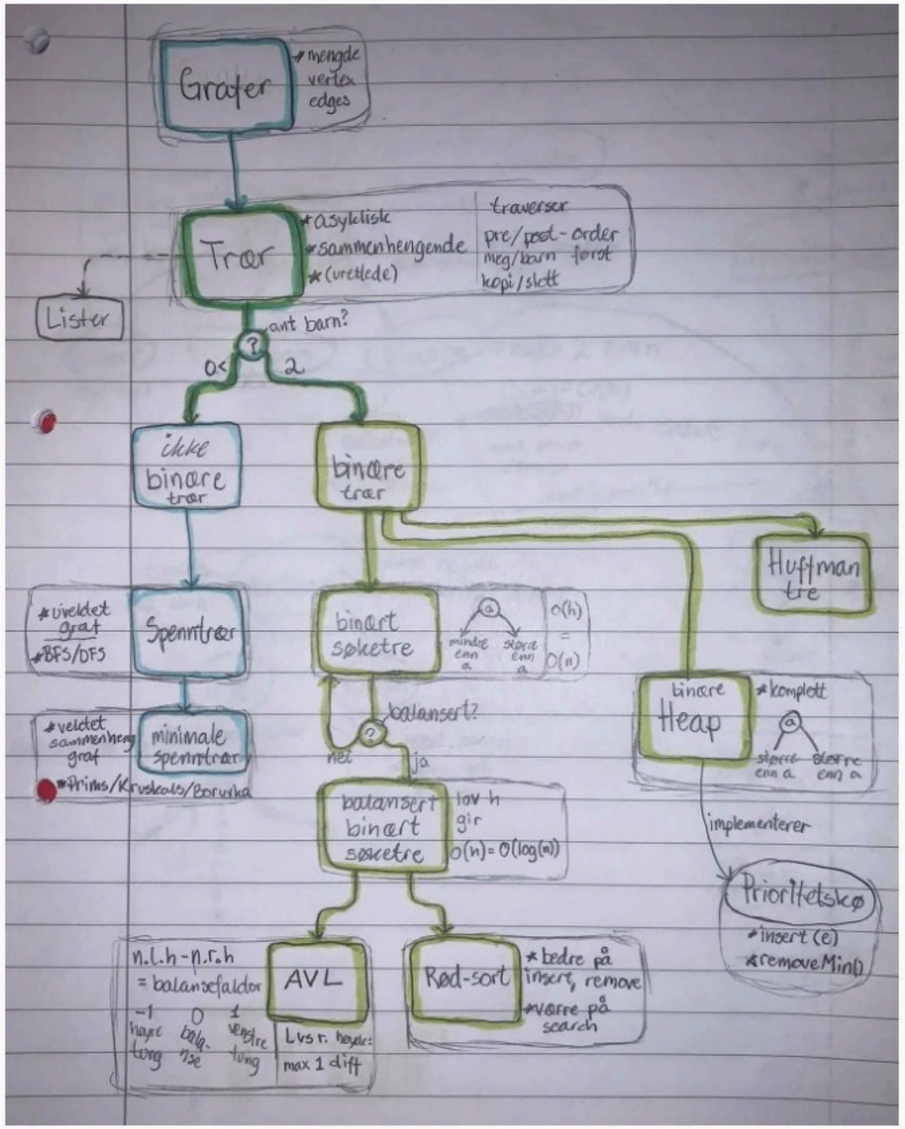
- Alle krav for vanlige trær
- og binære trær (2 barn)
- og binære søketrær (mindre <-> større)
- PLUSS: Hver node må være balansert
  - Høydeforskjell på venstre og høyre subtre må være <= 1
  - Dvs. balanser om den er mer enn 1

Binære søketrær vs. AVL

- → Oppslag: Identisk
- → Innsetting: Likt, men fiks høyde og balanser før ferdig
- → Sletting: Likt, men fiks høyde og balanser før ferdig

Rotasjoner!

Venstre, høyre og doble



Oppsummering, https://github.uio.no/veroninh/IN2010-Gruppe-3/blob/master/Figurer/Grafer%20og%20traer.JPG

Oppgaver

- Eksamensoppgaver
- Ukesoppgaver
- Skriv pseudokode for en algoritme som fjerner alle noder med data mindre enn et oppgitt tall
- Klarer du skrive opp pseudokoden for venstre eller høyre-rotasjoner uten å sjekke foilene?
- Bygg et AVL-tre steg for steg, ved å sette inn tallene: 41 38 31 12 19 8