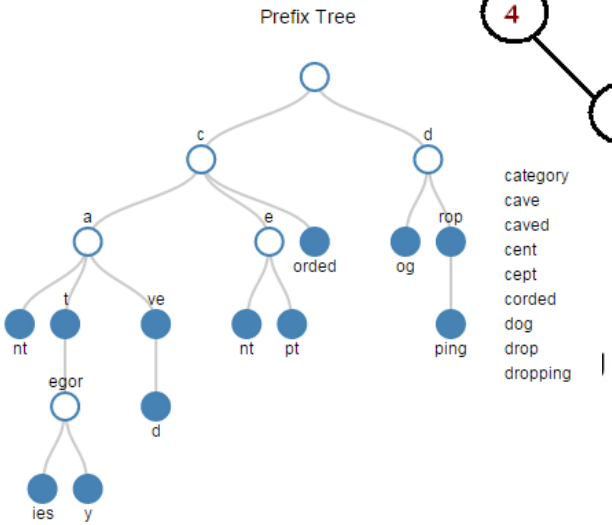
Trær

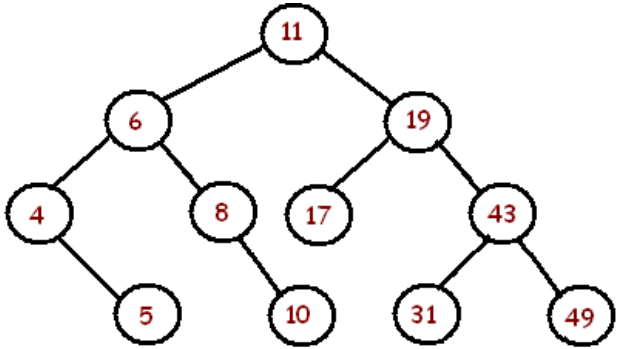


IN2010 – Uke 4

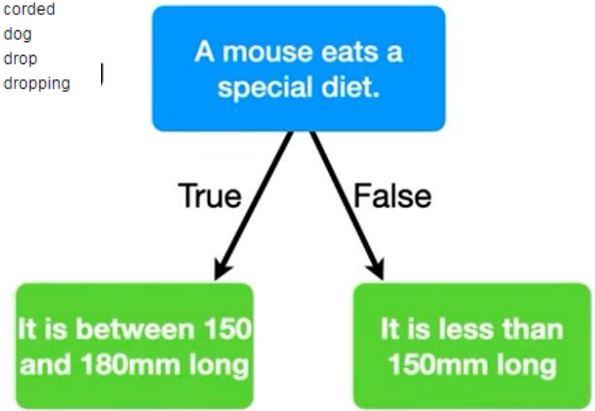


Trær

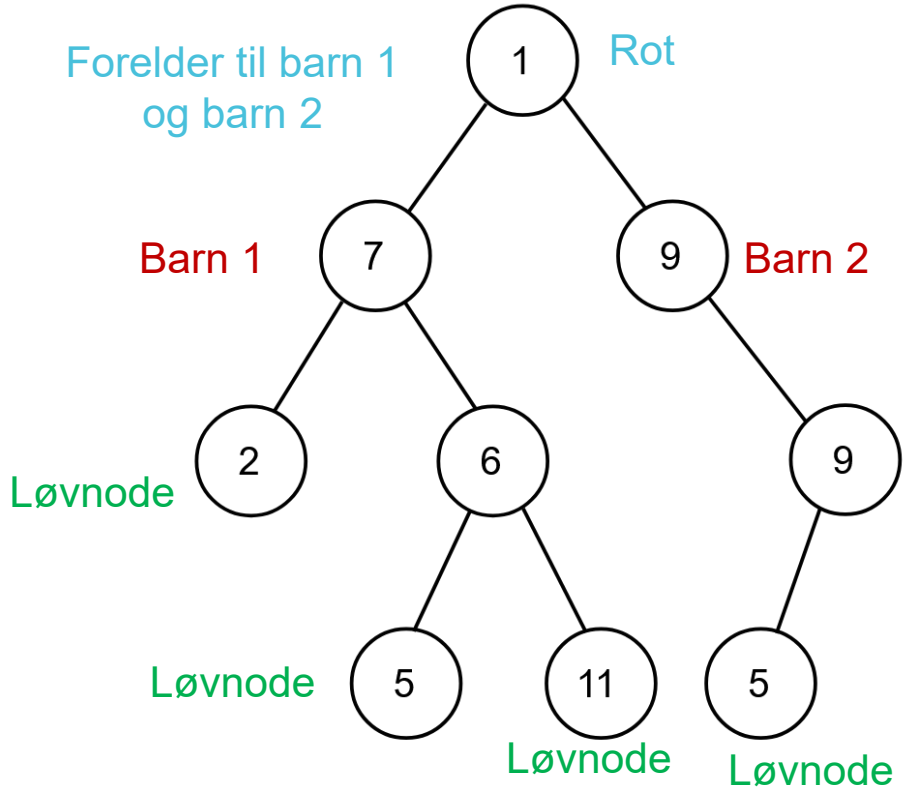




category
cave
caved
cent
cept
corded
dog
drop
dropping



Noen begreper



Flere begreper

Dybde:

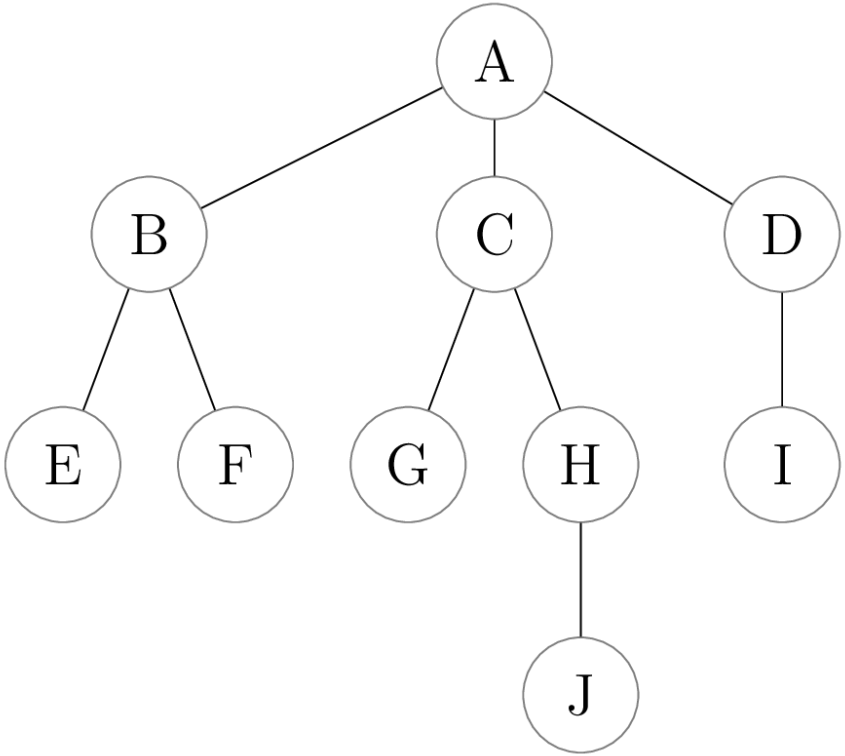
- Roten til et tre har dybde 0
- Tomme trær har dybde -1
- Dybden til en node er én (1) mer enn foreldrenoden

Høyde:

- Høyden til et tre er hva enn dybden til den dypeste noden er

OBS: Noder har dybde, trær har høyde

Kjappe oppgaver (5-7 min)

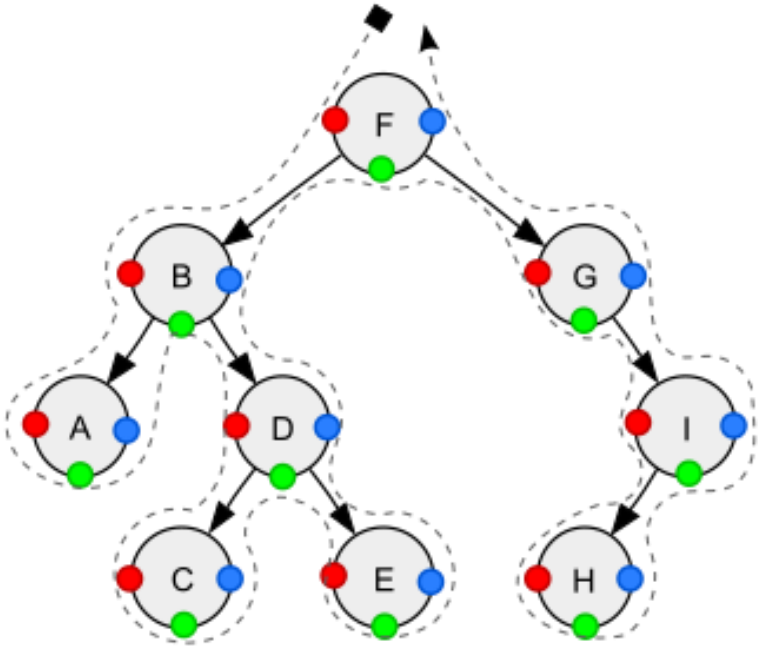


1) Hvilke noder er løvnoder?
2) Hva er dybden til A?
3) Hva er dybden til J?
4) Hva er høyden til treet med rotnoden A?

Traversering

Preorder vs Postorder

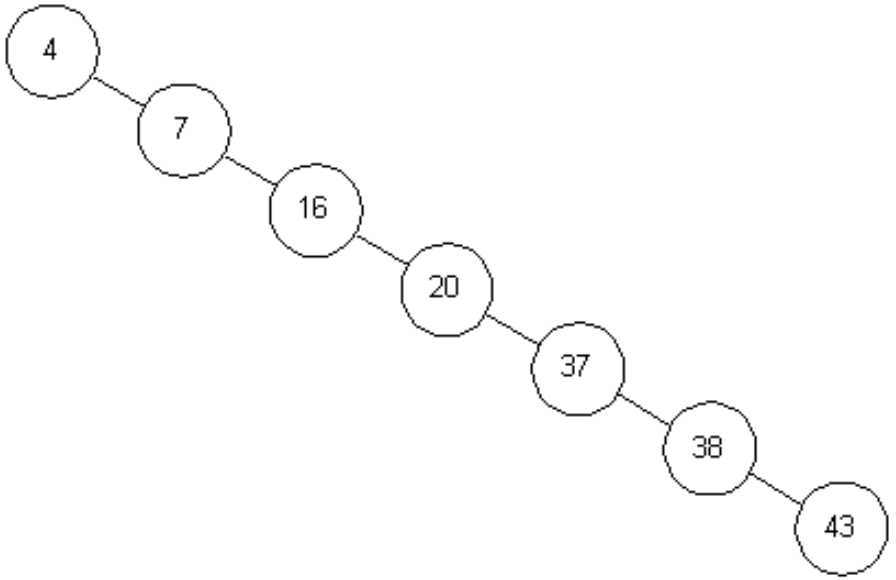
Piltriks (Preorder mot venstre,
Postorder mot høyre)



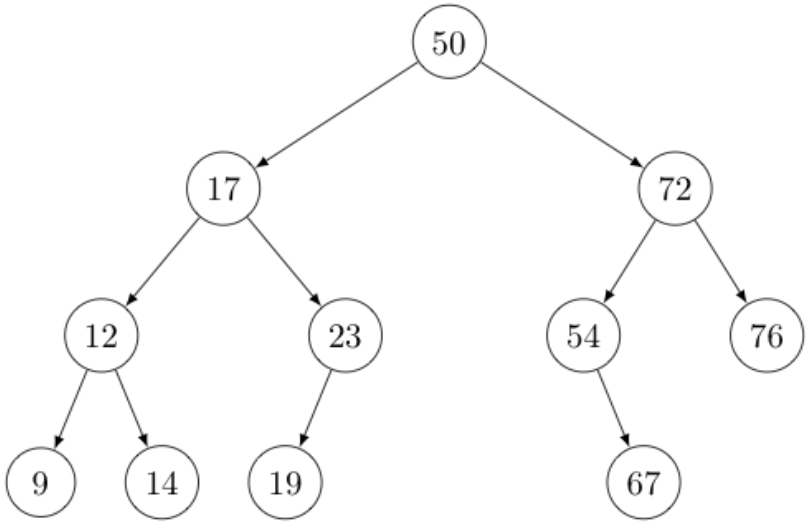
Binære søketrær

Mindre elementer går til venstre
Større elementer går til høyre

Innsetting/oppslag:
Worst case scenario – O(n)
Balansert scenario – O(log(n))



Kjappe oppgaver (7 min)



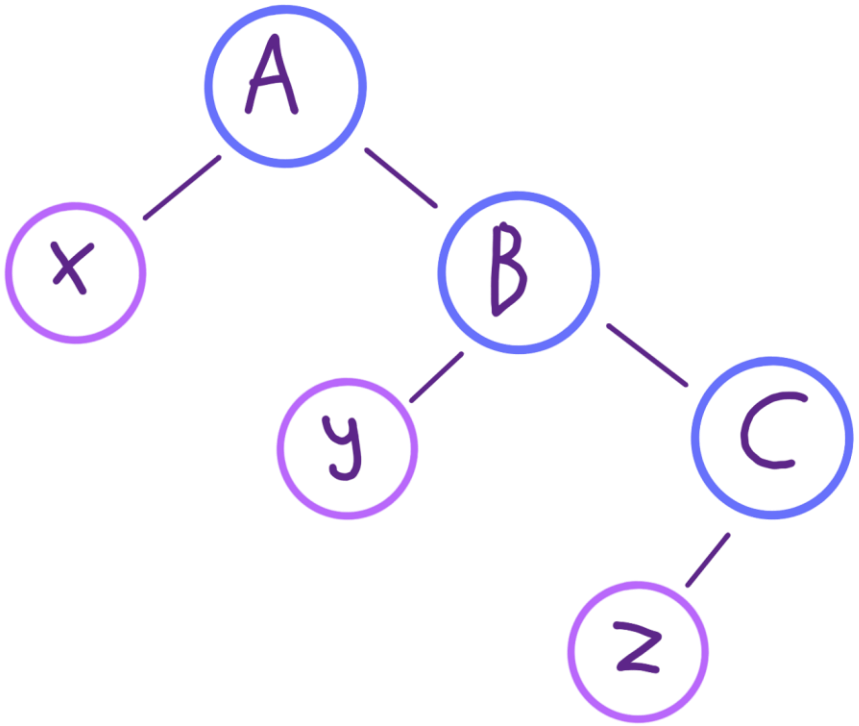
a) Hva skjer om vi setter inn tallet ‘25’?
b) Hva skjer om vi setter inn ‘13’?
c) Hva skjer om vi sletter tallet ‘17’?

AVL trær

Som binære søketrær MEN de må også være balansert

- Altså må høyre og venstre subtre ha en høydeforskjell på mindre eller er lik 1

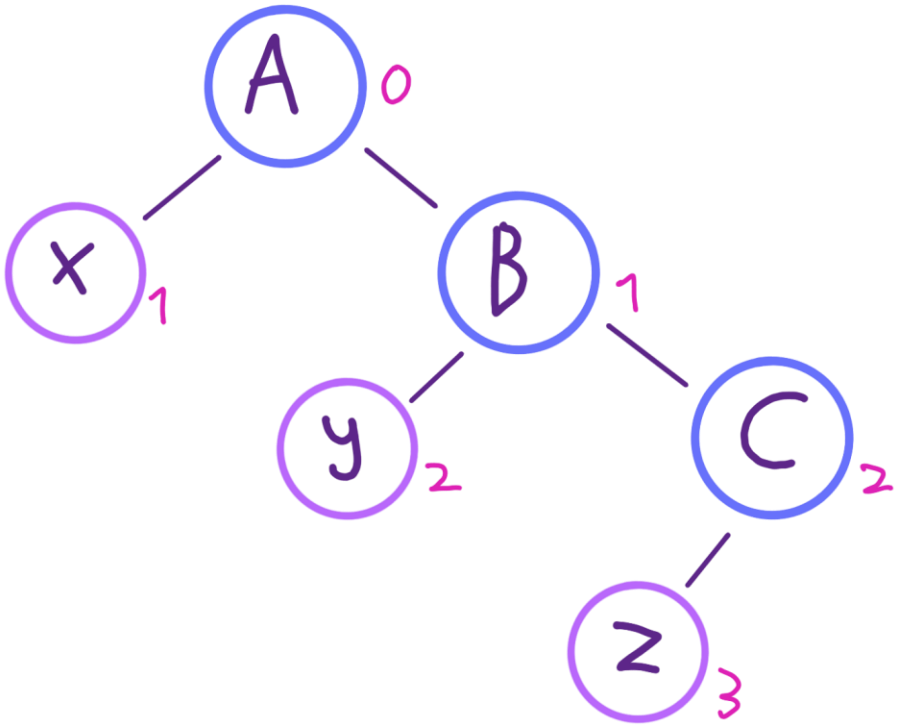
Hvordan ‘jevner vi ut’ treet? Vi roterer det!



ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

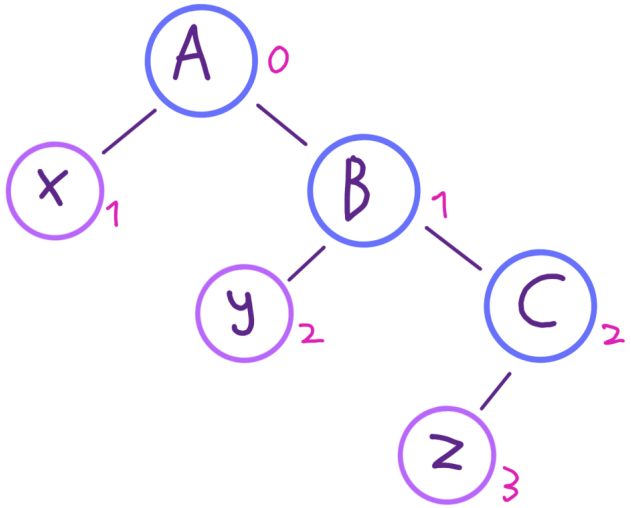


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

Balansefaktoren: 1 – 3 = -2

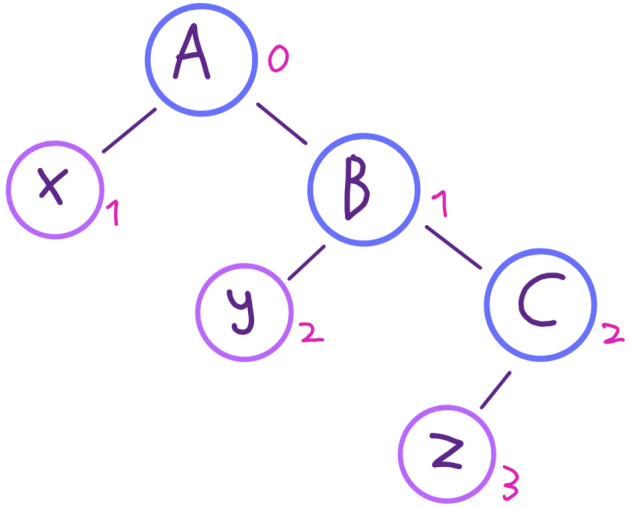


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

Balansefaktoren: 1 – 3 = -2

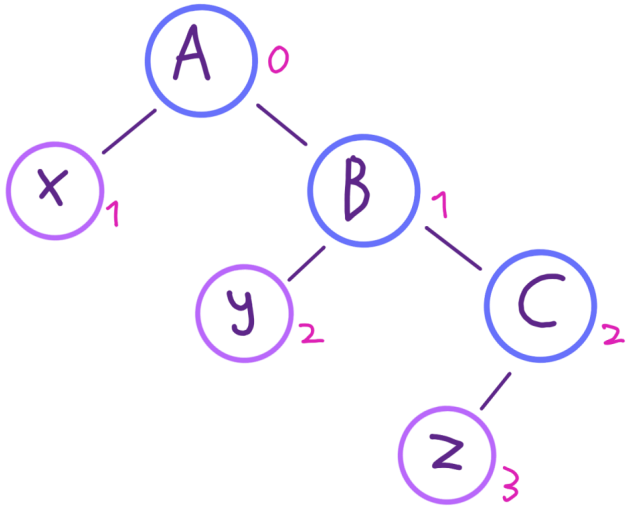


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

Balansefaktoren: 1 – 3 = -2

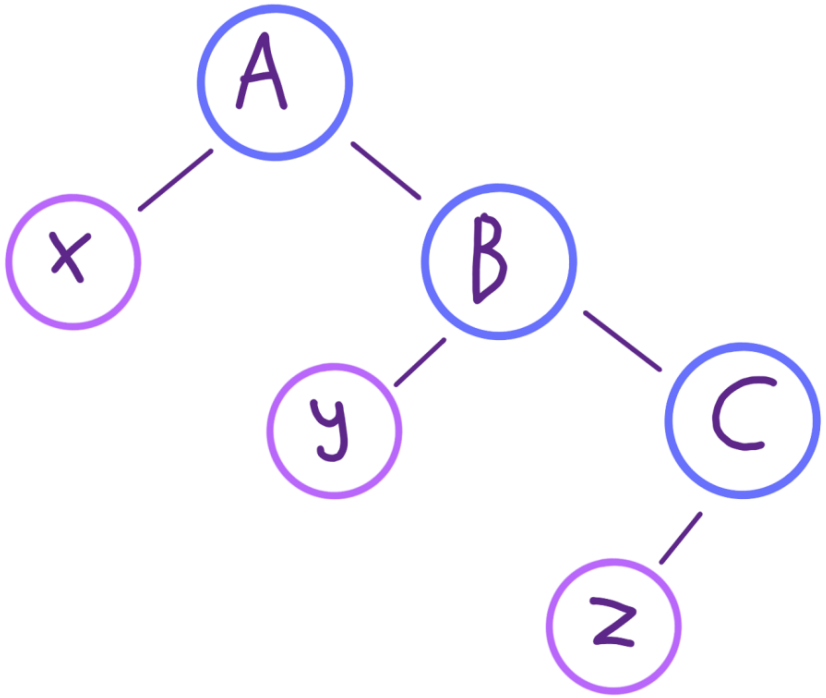


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

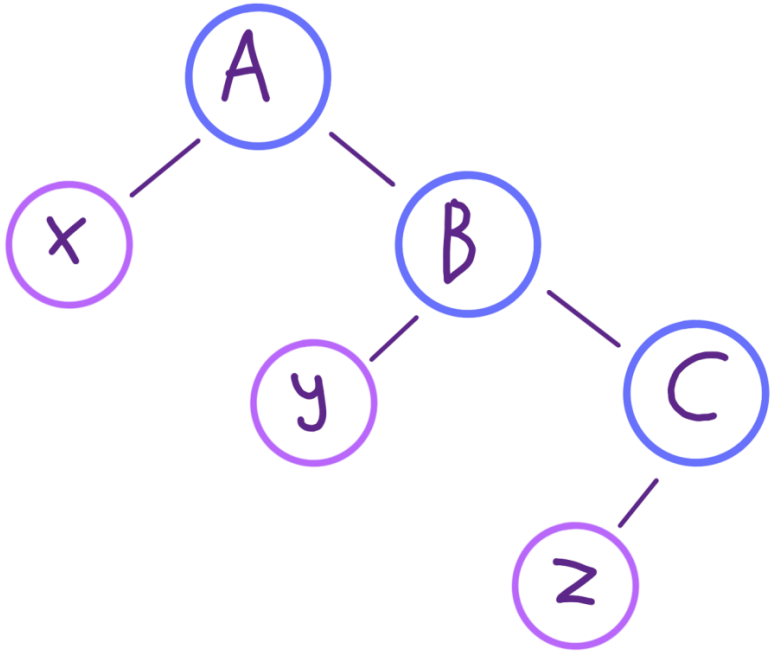
Balansefaktoren: 1 – 3 = -2

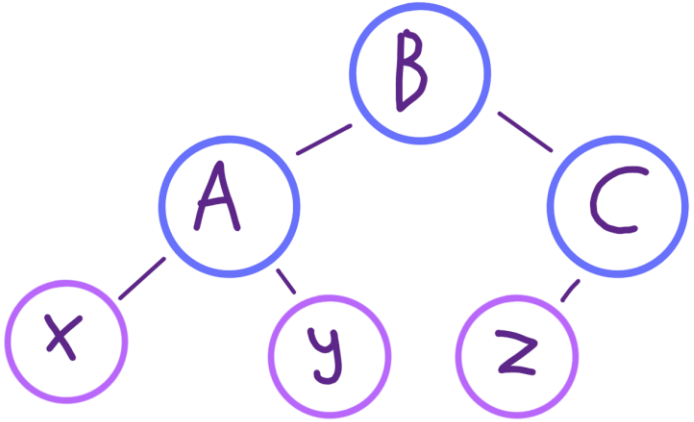


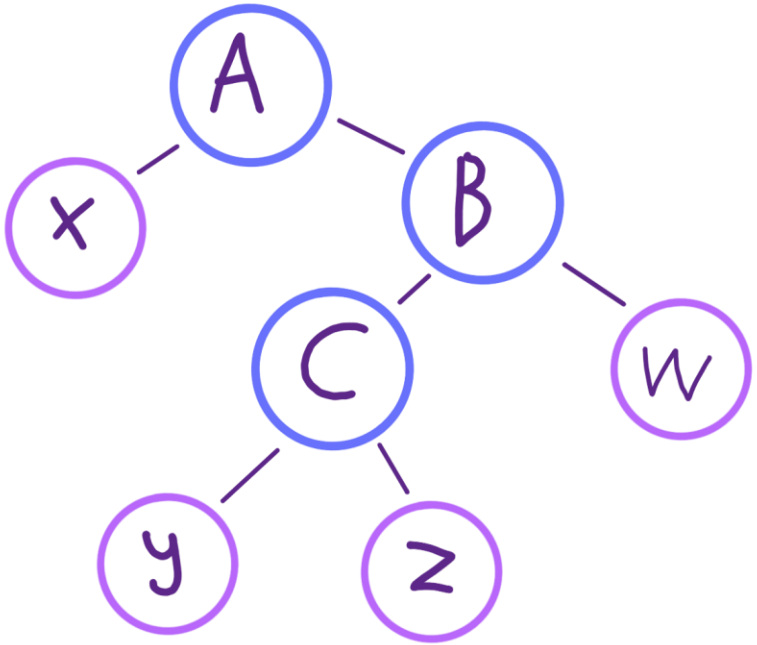
ALGORITHM: VENSTREROTASJON AV ET BINÆRTRE

Input: En node z
Output: Roter treet til venstre, slik at y blir den nye roten

```
1 Procedure LeftRotate(z)
2     y ← z.right
3     T1 ← y.left
4     y.left ← z
5     z.right ← T1
6     SetHeight(z)
7     SetHeight(y)
8     return y
```


A
x
B
y
C
z

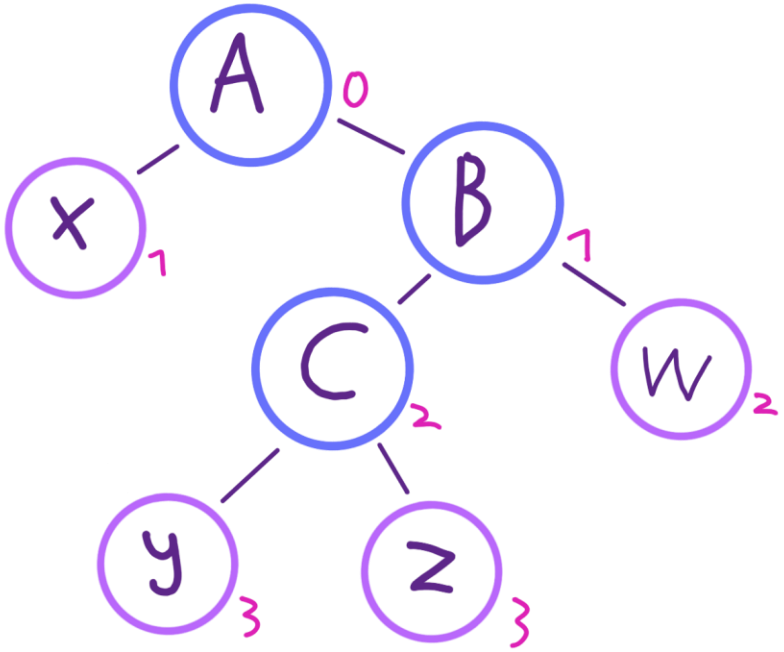

B
A
C
x
y
z



ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

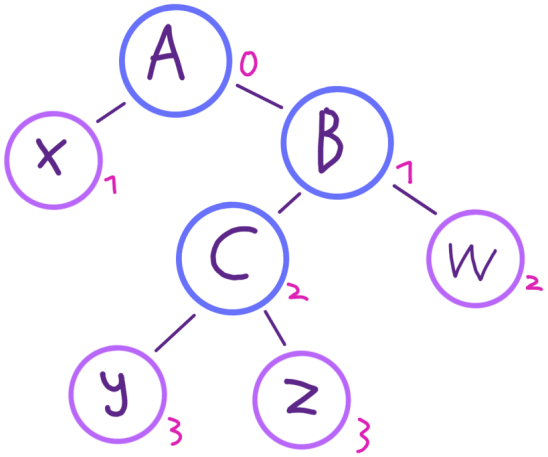


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

Balansefaktoren: 1 – 3 = -2

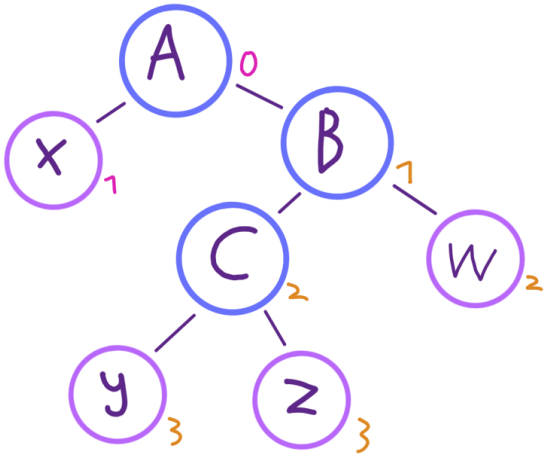


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

Balansefaktoren: 1 – 3 = -2

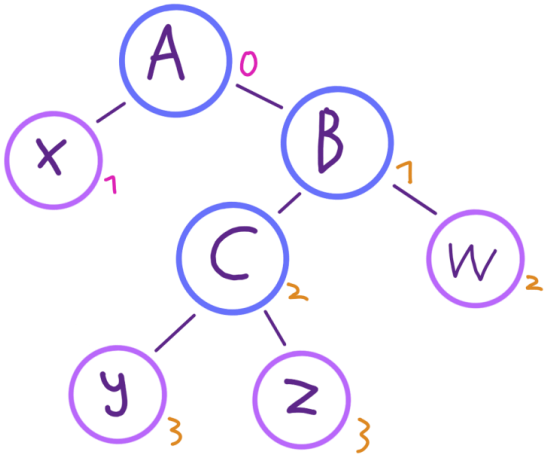


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

Balansefaktoren (til høyre subtre):
3 – 2 = 1

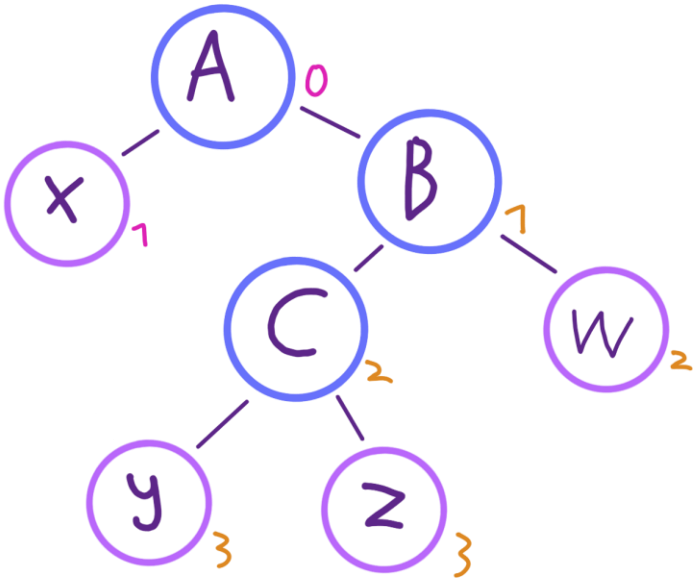


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```

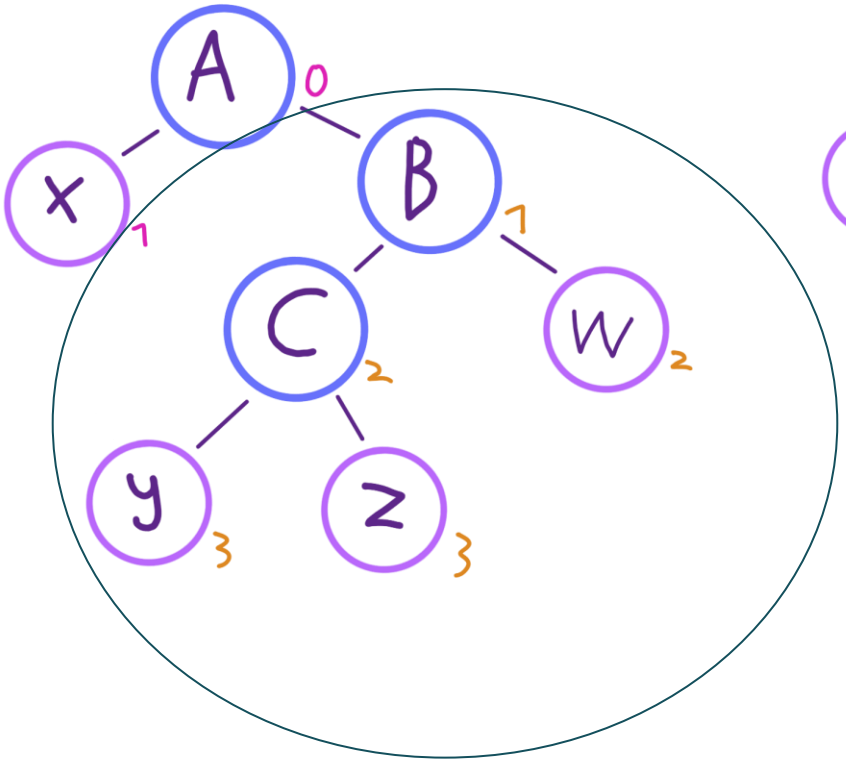
Balansefaktoren (til høyre subtre):
3 – 2 = 1

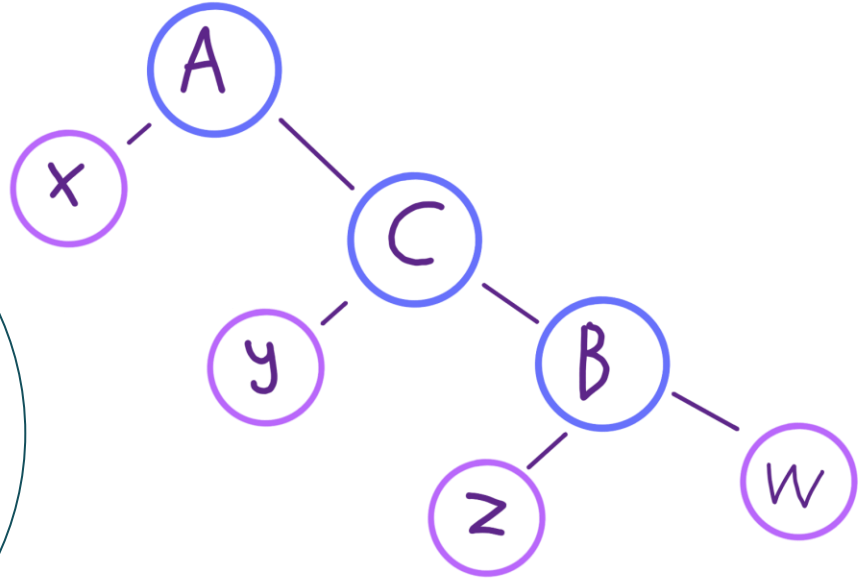


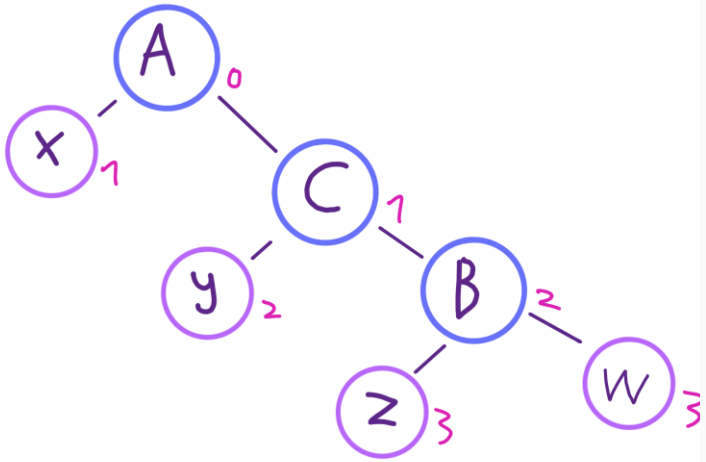
ALGORITHM: HØYREROTASJON AV ET BINÆRTRE

Input: En node z
Output: Roter treet til høyre, slik at y blir den nye roten

```
1 Procedure RightRotate(z)
2     y ← z.left
3     T2 ← y.right
4     y.right ← z
5     z.left ← T2
6     SetHeight(z)
7     SetHeight(y)
8     return y
```


A
0
x
1
B
1
C
2
w
2
y
3
z
3

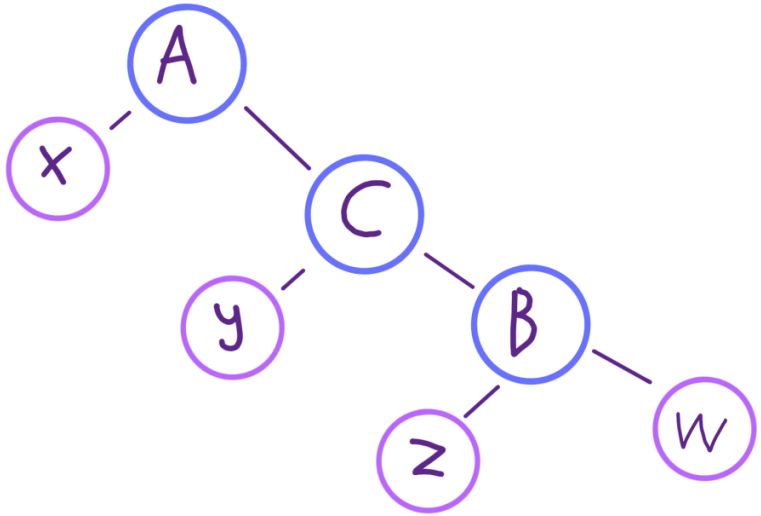

A
x
C
y
B
z
w

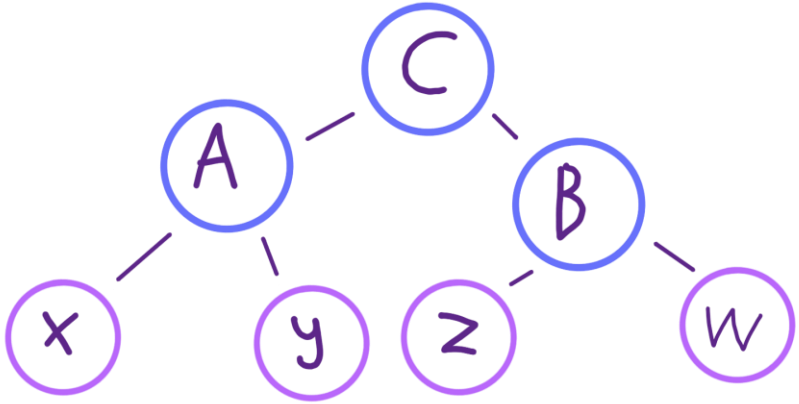


ALGORITHM: BALANSERING AV ET AVL-TRE

Input: En node v
Output: En AVL-balansert node

```
1  Procedure Balance(v)
2      if BalanceFactor(v) < −1 then
3          if BalanceFactor(v.right) > 0 then
4              v.right ← RightRotate(v.right)
5          return LeftRotate(v)
6      if BalanceFactor(v) > 1 then
7          if BalanceFactor(v.left) < 0 then
8              v.left ← LeftRotate(v.left)
9          return RightRotate(v)
10     return v
```


A
x
C
y
B
z
w


C
A
B
x
y
z
w

Ukesoppgaver + oblighjelp