Anti-AVL 12 poeng

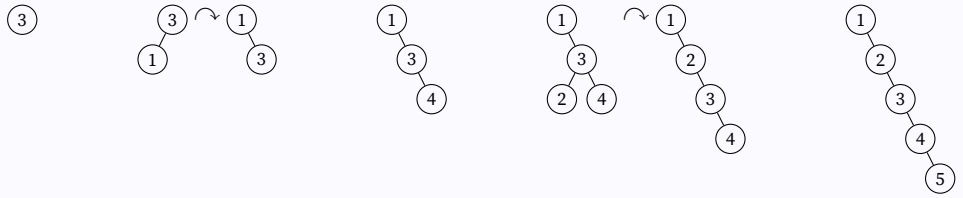
AVL-trær bruker venstre- og høyrerotasjoner for å sørge for at det binære søketreet ender opp balansert etter innsetting og sletting. Her er innsetting for en variant av AVL-trær som vi kaller *Anti-AVL*.

```
Input: Rotnoden v av et Anti-AVL-tre B og et
       element x
Output: Et Anti-AVL-tre med x satt inn
1  Procedure InsertAntiAVL(v, x)
2      if v = null then
3          v ← new Node(x)
4      else if x < v.element then
5          v.left ← InsertAntiAVL(v.left, x)
6      else if x > v.element then
7          v.right ←
            InsertAntiAVL(v.right, x)
8
9      if v.left ≠ null then
10         return RightRotate(v)
11
12     return v
```

```
Input: En node z
Output: Roten til det
        høyreroterte treet
1  Procedure RightRotate(z)
2      y ← z.left
3      T2 ← y.right
4
5      y.right ← z
6      z.left ← T2
7
8      return y
```

(a) Sett inn tallene $3, 1, 4, 2, 5$ med `InsertAntiAVL`.

- Beskriv kort hvordan det resulterende Anti-AVL-treet ser ut.
- Angi det totale antallet rotasjoner som forekommer i løpet av innsettingene.



Treet har 1 som rot, 2 som høyre-barn av roten, 3 som høyre barn av 2, og så videre. Altså er det en sortert lenket liste hvis du anser `right` som en neste-peker.

Det gjøres 2 rotasjoner total.

- Inntil 2 poeng.
- 1 poeng for beskrivelsen og 1 poeng for antall rotasjoner.

(b) Beskriv mer generelt hvordan Anti-AVL-trær ser ut.

Et Anti-AVL-tre er et binært søketre der hver node kun har et høyre-barn. Det vil se ut som en sortert lenket liste av høyre-pekere.

- Inntil 2 poeng.

(c) Et binært søketre er et binærtre der elementet i hver node er:

- større enn alle elementer i venstre subtre, og
- mindre enn alle elementer i høyre subtre.

Har Anti-AVL-trær den samme egenskapen? Forklar.

Ja, Anti-AVL er binære søketrær. En høyre-rotasjon bevarer BST-egenskapen. Siden innsetting i Anti-AVL-trær er identisk med ordinære binære søketrær, bortsett fra at det gjør en høyre-rotasjon i tilfellet der noden har et venstre barn, så vil innsetting fremdeles respektere BST-egenskapen.

- Inntil 3 poeng for en god forklaring.

(d) Hva er kjøretidskompleksiteten på oppslag i Anti-AVL-trær? Du kan anta at oppslag er implementert identisk som ordinære binære søketrær og AVL-trær. Begrunn svaret kort.

Oppslag i et Anti-AVL-tre er i $\mathcal{O}(n)$, der n er antall noder i treet. Hvis vi skal slå opp det største elementet i treet, vil vi følge $n - 1$ høyre-barn, som er $\mathcal{O}(n)$.

- Inntil 2 poeng.
- 1 poeng for riktig svar og 1 poeng for begrunnelsen.

(e) Beskriv kort hvordan innsetting i Anti-AVL-trær skiller seg fra innsetting i AVL-trær.

Den viktigste, og klareste forskjellen, er at AVL-trær bygger et svært balansert binært søketre, men Anti-AVL-trær bygger et svært *ubalansert* binært søketre.

AVL-trær er avhengig av at hver node lagrer høyden til sitt subtre, foretar rotasjoner dersom høydeforskjellen mellom to subtrær overstiger 1. Anti-AVL-trær sjekker kun om det venstre subtre er tomt.

- Inntil 3 poeng.
- 2 poeng for å peke ut den veldig tydelige forskjellen, og 1 poeng for å peke ut at AVL-trær gjør rotasjoner basert på høyden.