Løsningsforslag eksamen DAT200 høst 2019

Oppgave 1 (Automatisk rettet)

O(1)

Oppgave 2 (Automatisk rettet)

O(n^2)

Oppgave 3 (Skulle ha vært automatisk rettet, feil i oppsett)

O(n*log(n))

Jeg ser det er en feil i oppsettet av den automatiske rettingen her, så man må gå gjennom denne manuelt. I følge oppsettet er O(1) riktig! Sett 6 poeng til de som har riktig svar, O(n*log(n)).

Oppgave 4 (Manuelt rettet)

Dynamisk programmering er en programmeringsteknikk hvor man løser et problem gjennom å løse delproblemer, og mellomlagrer resultatene fra delproblemer for å løse hele problemet. Delproblemene kan overlappe i dynamisk programmering. Dynamisk programmering brukes for å unngå å gjøre samme jobb flere ganger.

Ett eksempel er måten man regner ut Fibonacci-tallene på. F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. Så for å regne ut det n-te Fibonacci-tallet så må du først regne ut det n-1-te og det n-2-te Fibonacci-tallet. For å løse dette kan man starte med starttilfellene 0 og 1 og så regne ut det 2. Fibonacci-tallet, så det 3. Fibonacci-tallet og så videre helt opp til n, og hvor man hele tida har de to foregående tallene lagret i variabler.

Oppgave 5 (Manuelt rettet)

En stabel er en datastruktur hvor man både setter inn og tar ut fra slutten. Stabler har typisk følgende operasjoner

- Push: Legger et nytt objekt på toppen av stabelen
- Pop: Fjerner et objekt fra toppen av stabelen
- Peek: Ser på øverste objekt i stabelen men uten å fjerne det
- Size: Høyden til stabelen. (mindre viktig enn de tre foregående. Å erstatte size med is_empty() gir fortsatt full score da det viktigste er å skille mellom en tom stabel og en med innhold)

Staber er en LIFO – Last in, first out, struktur.

Stabler er typisk implementert med lister, gjerne array-lister, hvor man bruker append metoden for push, sletter siste element for pop og returnerer siste element for peek.

Oppgave 6 (Manuelt rettet)

Når man sorterer ei liste kan lista inneholde flere elementer som for sorteringen er like. For eksempel hvis man sorterer personer på navn kan man ha navnebrødre. En stabil sorteringsalgoritme

vil la slike like elementer være i samme rekkefølge i resultatet som de hadde i den opprinnelige lista. Ustabile sorteringsalgoritmer garanterer ingenting her.

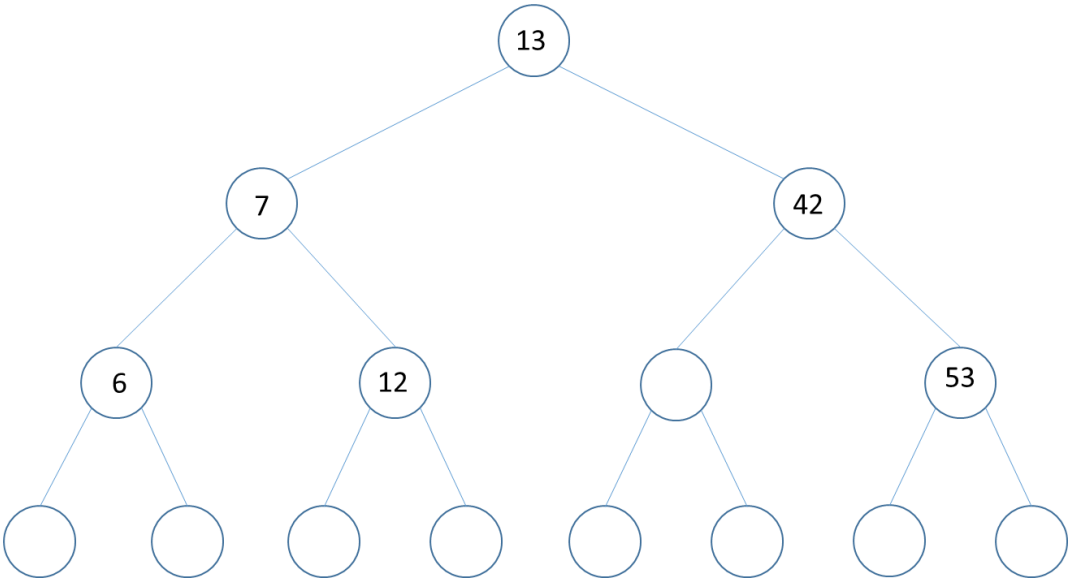
Oppgave 7 (Automatisk rettet)

Pivot: 37

Første liste: 29, 23, 1, 0, 0

Andre liste: 47, 93, 46, 52, 89

Oppgave 8 (Automatisk rettet)



Oppgave 9 (Automatisk rettet)

Innsetting i en hashtabell, average case: O(1)

Innsetting i et AVL tre: O(log(n))

Å finne et element i en hashtabell, average case: O(1)

Å finne et element i en hashtabell, worst case: O(n)

Å finne et element i et AVL tre: O(log(n))

Å finne det minste elementet i en hashtabell: Strukturen støtter ikke denne operasjonen

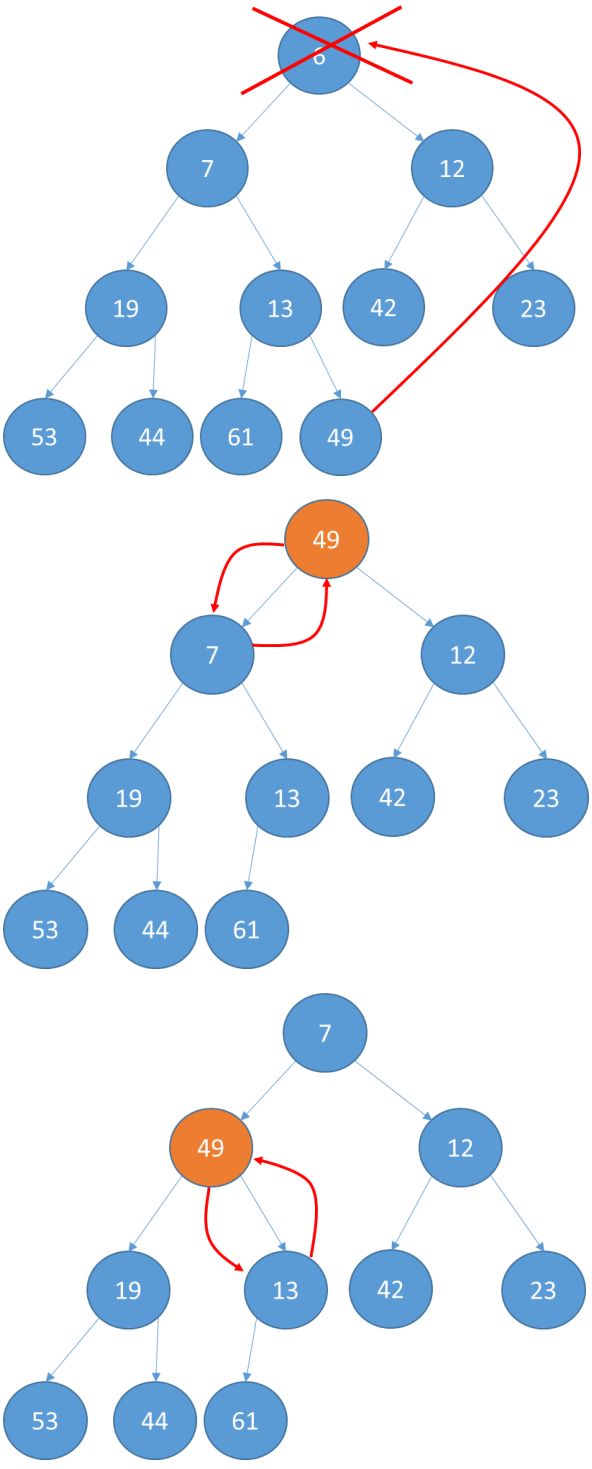
Å finne det minste elementet i et AVL tre: O(log(n))

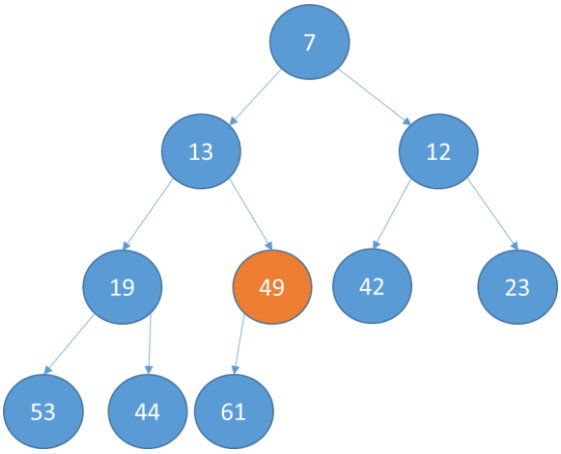
Oppgave 10 (Manuelt rettet)

Steg i prosessen:

1. Erstatt rota med siste element i haugen
2. Sammenlikn den nye rota med sine barn. 49 er større enn både 7 og 12. Bytt rota med sitt minste barn, 7.
3. Sammenlikn 49 med sine barn i den nye posisjonen. 49 er større enn både 13 og 19. Bytt 49 med sitt minste barn, 13.

4. Sammenlikn 49 med sine barn i den nye posisjonen. Den har nå bare ett barn, 61, som er større enn 49. 49 er nå i riktig posisjon.

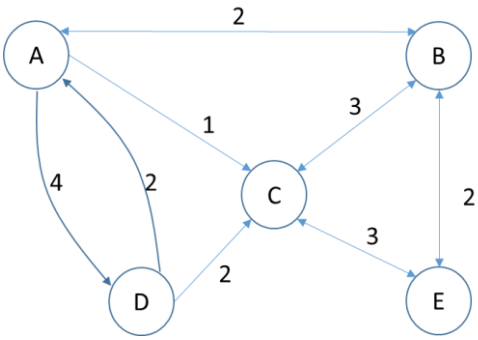




Oppgave 11 (Manuelt rettet)

Directed Acyclic Graph

Oppgave 12 (Manuelt rettet)



Siden det er noen kanter som går bare en vei, eller har ulik kostnad de to veiene, er dette en rettet graf.

Siden oppgaven ikke er entydig på hva som er «fra» og hva som er «til» skal en graf som har motsatt retning på alle kantene også få full score. Andre grafer vil få redusert score.

Oppgave 13 (Manuelt rettet)

Løsningsskisse:

- Start med Dijkstra's Algoritme fra timene
- Erstatt kodeblokken «if node_index == til_index» med en kodeblokk som sjekker om noden har riktig type – at noden er en brannstasjon. Hvis noden er en brannstasjon, legg et tuppel med noden og avstand til startnoden inn i ei liste. Hvis lista er 5 lang, returner.
    - Må ha fem for hver tunnelmunning siden man ikke vet hvor mange fra hver munning man trenger.
- Kjør den modifiserte algoritmen en gang for hver tunnelmunning (hvor startnoden er lik tunnelmunningen)
- Slå sammen listene for alle tunnelmunningene av samme tunnel.
    - Siden listene fra ulike tunnelmunninger kan inneholde samme brannstasjon, gå gjennom lista og fjern duplikater, behold innslaget med kortest avstand.
    - Gå gjennom lista for å finne de fem korteste avstandene totalt.

- Siden disse listene er bare 5 lange er det ikke så viktig med en optimal algoritme for dette. Kan bruke en haug for optimal ytelse.

Kode som kan erstatte «if node_index == til_index» blokken i Dijkstra's algoritme:

```
if graf.get_markering(node_index) == "brannstasjon":
    resultat.append((node_index, node_kostnad))
    if len(resultat) >= 5:
        return resultat
```

Om Dijkstra vs. Bellman Ford:

- Dijkstra's algoritme i denne formuleringen vil stoppe når den har funnet de nærmeste fem brannstasjonene. Bellman Ford stopper ikke før den har gått gjennom hele grafen og funnet korteste vei til alle brannstasjonene. Dijkstra er derfor mye mer effektiv til dette enn Bellman ford. Denne oppgaven kjører på et vei-nettverk. I et vei-nettverk har man ingen negative kanter. Derfor er Dijkstra'a algoritme velegnet her.