Dat102 Oblig 3

Thobias K. Høivik

March 11, 2025

Uke 10 d) Tidskompleksitetsanalyse

Vi skal analysere tidskompleksiteten for følgende metoder for `TabellMengde` (array-basert mengde) og `LenketMengde` (lenket liste-basert mengde). Vi bruker *O-notasjon* for å uttrykke kjøretiden, og tar med både beste og verste tilfelle der det er forskjell.

i. `boolean inneholder(T element)`

- *Sjekk om et element finnes i mengden*

TabellMengde (Array-basert mengde):

- **Beste tilfelle**: Elementet finnes i første posisjon, som tar konstant tid, $O(1)$.
- **Verste tilfelle**: Elementet er på siste posisjon eller ikke i mengden i det hele tatt, og vi må iterere gjennom hele arrayet, som tar lineær tid, $O(n)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle**: Elementet er i hodet av listen, og tar konstant tid, $O(1)$.
- **Verste tilfelle**: Elementet er på slutten av listen eller ikke i listen, og vi må traversere hele listen, som tar lineær tid, $O(n)$.

Så, for `LenketMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n)$$

ii. `boolean erDelmengdeAv(MengdeADT<T> annenMengde)`

- *Sjekk om mengden er en delmengde av en annen mengde*

TabellMengde (Array-basert mengde):

- **Beste tilfelle**: Hvis første element ikke finnes i den andre mengden, kan vi umiddelbart returnere `false`, som tar $O(1)$.
- **Verste tilfelle**: Vi må sjekke hvert element i den nåværende mengden for å se om det finnes i den andre mengden. For hvert element i den nåværende mengden, sjekker vi om det finnes i den andre mengden, som tar $O(n)$ for hvert av de n elementene, som gir en total tidskompleksitet på $O(n^2)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n^2)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle**: Hvis første element ikke finnes i den andre mengden, kan vi umiddelbart returnere `false`, som tar $O(1)$.
- **Verste tilfelle**: Vi må sjekke hvert element i den nåværende mengden, som innebærer å sjekke om hvert element er i den andre mengden. Dette krever å traversere begge listene, og derfor tar det $O(n^2)$ i verste tilfelle, der n er antall elementer.

Så, for `LenketMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n^2)$$

iii. `boolean erLik(MengdeADT<T> annenMengde)`

- *Sjekk om mengden er lik en annen mengde*
TabellMengde (Array-basert mengde):
For å se om to mengder er like, matematisk, innebærer å sjekke om de to mengdene er delmengder av hverandre. Så hvis de har forskjellig kardinalitet vil de ikke være like. Desverre ville dette involvert å se gjennom begge mengdene før vi gjør noe mer så vår metode kaller bare hver mengde sin er `delMengdeAv()` metode siden vi ikke kan spare noe meningsful tid.

- **Beste tilfelle:** `erDelMengdeAv` $\rightarrow O(1)$. Vi kaller metoden to ganger som gir oss, men dette simplifiseres til $O(1)$.
- textbfVerste tilfelle: Igjen for vi det dobbelte av `erDelMengdeAv`, men $2n^2$ simplifiseres til n^2 så vi sitter igjen med $O(n^2)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n^2)$$

LenketMengde (Lenket liste-basert mengde):
For akkurat samme grunner som i TabellMengde seksjonen får vi
`LenketMengde`:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n^2)$$

iv. `MengdeADT<T> union(MengdeADT<T> annenMengde)`

- *Union av to mengder*

TabellMengde (Array-basert mengde):

- **Beste tilfelle**: Hvis begge mengdene er tomme, lager vi og returnerer vi bare en mengde som tar $O(1)$.
- **Verste tilfelle**: Vi må traversere begge mengdene. I verste tilfelle, hvis begge mengdene har n elementer, tar det $O(n)$ for å traversere begge mengdene og $O(n)$ for å kopiere resultatet til en ny array. Så, tidskompleksiteten er $O(n)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle**: Samme igjen viss begge mengdene er tomme: $O(1)$.
- **Verste tilfelle**: Vi må traversere begge lenkede listene og sette inn alle elementene fra den andre listen i den første. Dette krever å besøke alle elementene i begge listene, og tar derfor $O(n)$.

Så, for `LenketMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n)$$

v. `T fjern(T element)`

- *Fjern et element fra mengden*

TabellMengde (Array-basert mengde):

- **Beste tilfelle**: Elementet er på første posisjon, og vi kan bare bytte dette elementet med det siste og fjerne det siste. Dette tar $O(1)$. Alternativt er mengden tom og det tar fortsatt $O(1)$
- **Verste tilfelle**: Elementet er på siste posisjon eller finnes ikke i det hele tatt, og vi må traversere hele arrayet før vi kan fjerne elementet. Dette tar$O(n)$.

Så, for `TabellMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n)$$

LenketMengde (Lenket liste-basert mengde):

- **Beste tilfelle**: Elementet er i hodet av listen eller hodet finnst ikke, og vi kan bare oppdatere pekerne/gjøre ingenting. Dette tar $O(1)$.
- **Verste tilfelle**: Vi må traversere hele listen for å finne elementet, som tar $O(n)$.

Så, for `LenketMengde`, er tidskompleksiteten:

$$\text{Beste tilfelle} : O(1), \quad \text{Verste tilfelle} : O(n)$$

Oppsummering av tidskompleksiteter

Metode	Tabell Beste	Tabell Verste	Lenket Beste	Lenket Verste
i. inneholder()	$O(1)$	$O(n)$	$O(1)$	$O(n)$
ii. erDelmengdeAv()	$O(1)$	$O(n^2)$	$O(1)$	$O(n^2)$
iii. erLik()	$O(1)$	$O(n^2)$	$O(1)$	$O(n^2)$
iv. union()	$O(1)$	$O(n)$	$O(1)$	$O(n)$
v. fjern()	$O(1)$	$O(n)$	$O(1)$	$O(n)$

Uke 10 e) HashSet & TreeSet vs. Tabell & Lenket

Generellt når vi gjorde praktiske hastighetstester mellom Lenket- og Tabellmengde vs. HashSet, som vi bruket i JavaSetToMengde implementasjonen, observerte vi en hastighetsforskjell på to størrelsesordener; en enorm forskjell. Når det gjelder sammenligning mellom Lenket- og Tabellmengde observerte vi i våre tester at tabellmengde som oftest var litt raskere. Med HashSet og TreeSet står det på nettet at HashSet som oftest har bedre kompleksitet, gjern $O(1)$ når TreeSet er $O()$. Den store fordelen for TreeSet er vistnok er sortert og kan da traverserest effektivt. TreeSet har også noen ekstra metoder som egner seg til noen bruksområder.

Figure 1: Kjøretider

Uke 10 f)

Figure 2: Match

Uke 11

Figure 3: Kjøretider

Vi observerer at HashSet er 5-6 ganger raskere for verdiene vi testet. Med en god hashing funksjon er tidskompleksiteten for å hente ut et gitt element $O(1)$ vs. binærsøk som er $O(\log n)$.