INF-1101: Datastrukturer og algoritmer

# Obligatorisk oppgave

Jesper Rødstrøm

Februar 23th, 2018

## 1 Introduksjon

I denne oppgaven skal man lære hvordan man implementerer en sortert ADT og hvordan man kan utføre logiske operasjoner på flere datasett. Manipulasjon av eksterne filer er også brukt i oppgaven. Ved å fullføre oppgaven danner man kunnskap innenfor abstrakte data typer og man får kjennskap til sortering og manipulering av data. Dette er grunnleggende kunnskap for all programmering og kan brukes som et verktøy i fremtidige prosjekter.

### 1.1 Krav

Set ADT

* Legge til elementer i settet.
* Hente den nåværende størrelsen av settet.
* Sjekke om settet inneholder ett spesifikt element.
* Regne ut unionen mellom to sett.
* Regne ut snittet mellom to sett.
* Regne ut differansen mellom to sett.
* Iterere gjennom settet i sortert rekkefølge.

Spamfilter

* Beregne om en mail inneholder spam eller ikke basert på en regel.
* Kommando-linjen skal trenge tre mapper som inneholder informasjon om hvor programmet kan finne filene som skal bli brukt i programmet.

## 2 Teknisk Bakgrunn

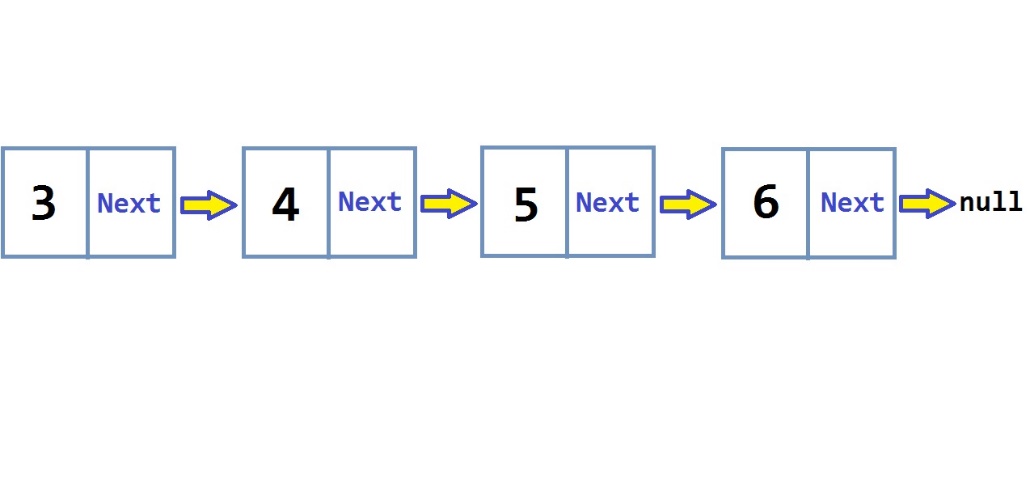
FILE I/O: Input og output mellom eksterne filer og et program. Dekker flere funksjoner som kan manipulere filer av flere format.

Mergesort: Sorterer data ved å dele den opp i flere biter og så setter den sammen igjen i sortert rekkefølge. Denne algoritmen er veldig flink til å håndtere massive mengder med data og har veldig god ytelse (n log(n)).

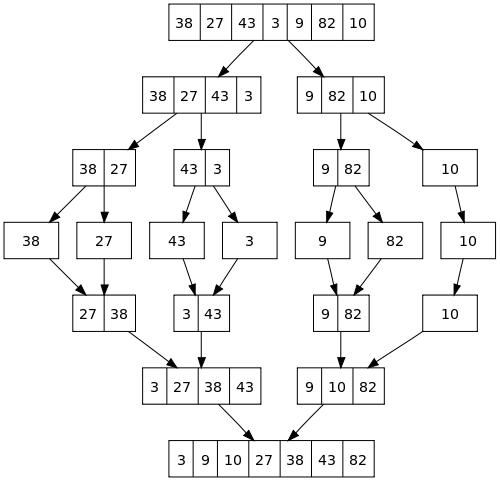
ADT: Står for "abstract data type" og består av en gruppe med data som man kan utføre et sett med operasjoner på. ADT-er brukes blant annet til å designe/analysere algoritmer og datastrukturer. Et eksempel på en ADT er int.

Char \*: Datatypen char tilsvarer en bokstav, men når du deklarer en char pointer kan dette kalles en string. En string består av flere characters satt sammen. Denne datatypen brukes for å holde ord og setninger.

## 3 Design & Implementasjon

Den sorterte ADT-en som skal implementeres kan bli gjort dette på mange måter. I denne løsningen blir det brukt linked-lists for å løse problemet. Når man skal bruke lists med ADT-en gir man strukturen en peker til en list som skal inneholde elementene. Ved å bruke lists har man gode muligheter til å iterere gjennom elementene i datasettet.

For å sortere datasettet brukes en listefunksjon som er implementert. Den sorteringsfunksjonen bruker merge-sort. Merge-sort er en veldig effektiv sorteringsalgoritme som kan takle massive mengder data med lav kostnad innenfor program fart. Algoritmen tar listen og deler den i to kontinuert helt til den består av flere lister med bare et element i seg. Når en liste bare har et element i seg er listen sortert, merge-sort fortsetter da med å sette sammen lister to og to i sortert rekkefølge. Når to lister settes sammen starter merge-sort med å lage en ny liste, så itererer den gjennom de to sorterte listene, imens den samtidig legger til det minste elementet mellom listene i den nye listen. Tilslutt så slettes de gamle listene og en ny sortert liste av nesten doblet størrelse har blitt dannet. Dette gjøres helt til algoritmen ender opp med en fullstendig liste hvor alle elementer er sortert. Ytelsen på denne algoritmen er veldig effektiv (n log(n)) sammenlignet med andre sorteringsalgoritmer.

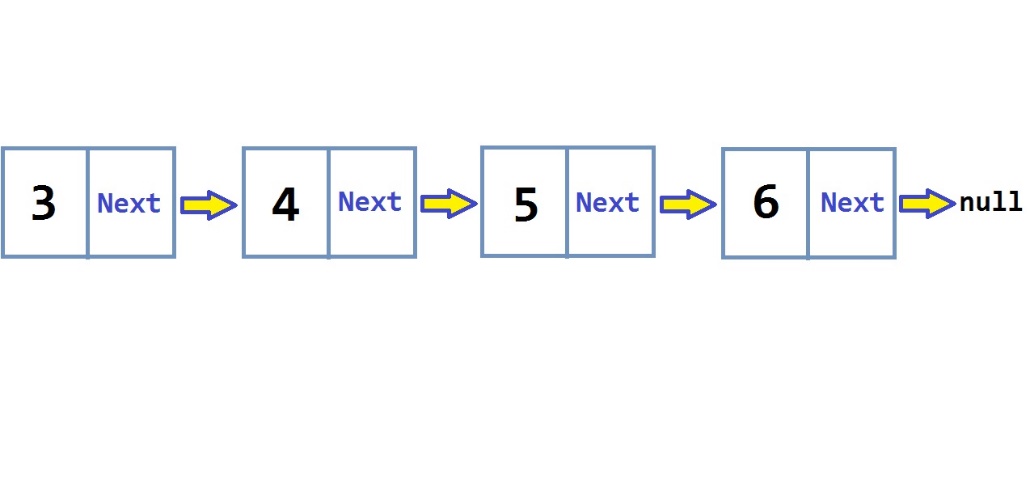
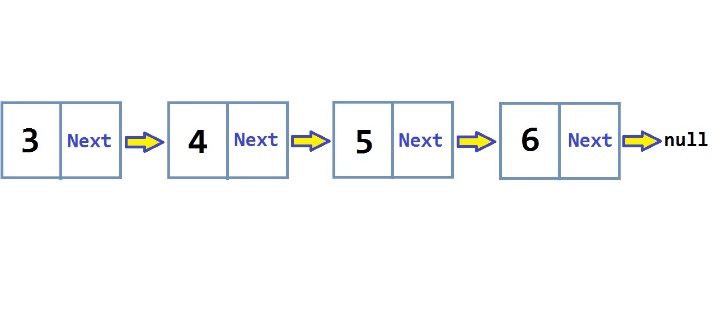


Når man skal implementere funksjonene for å foreta logiske operasjoner på flere sett kan man bruke en liste funksjon som itererer gjennom en liste for å sjekke om et spesifikt element er i listen. Denne funksjonen er ikke særlig effektiv (se diskusjon), men gjør det mer beleilig å implementere funksjonene for union, snitt og differanse. Man kan bruke den funksjonen i alle operasjonene for å sjekke at man ikke legger til samme element flere ganger i den nye listen. Dette er alt man trenger å gjøres for å implementere union når du iterer gjennom begge settene. Når man skal implementere snitt bruker man funksjonen for å sjekke om elementene du itererer over eksiterer også i det andre settet. For differanse gjør man det motsatte av dette, istedenfor å legge til elementet i et nytt sett hvis det finnes i begge sett iterer man isteden videre og bare legger til de som ikke finnes i begge sett.

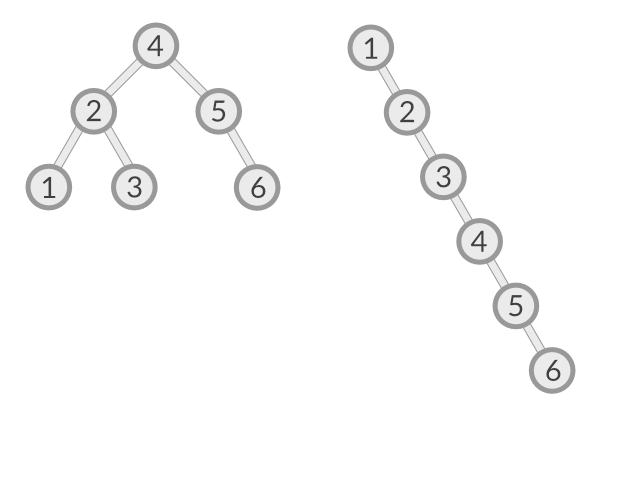
For å implementere spamfilter må man bruke en funksjon tokenize for å kunne manipulere ordene i eksterne filer. I denne løsningen bruker man en funksjon findfiles og lister. I listene setter man filene som skal bli analysert. Hver liste er representert som en mappe som inneholder filer av samme type. Når man skal bruke set funksjonene sine for å løse betingelsen for spam iterer man over disse listene. Man lagrer alle nøklene som man får gjennom tokenize i et set som videre er lagret i listene. Ved å bruke to set og en while loop kan man iterere gjennom listen og bruke set funksjonene på alle filene. Når man har gjort dette for å løse kravet kan man printe resultatet på kommandovinduet.

## 4 Diskusjon

Denne implementasjonen kunne vært mer effektiv. Et eksempel på noe som sakker programmet er bruken av list\_contains funksjonen. Denne funksjonen iterer over et sett hver gang loopen starter på nytt. Dette illustreres i bildet under. I funksjonen som regner snitt gjøres dette på to sett for hvert element som blir iterert over. Hvis mengden data er lav er ikke konsekvensene store, men for store mengder data vil dette redusere ytelsen til programmet. For å fikse dette problemet kunne man funnet en løsning som ikke tar denne funksjonen i bruk og heller bruke flere iteratorer en bare én.



Iterator

Datasettet kunne også blitt implementert på flere måter. Man kunne brukt binære søketre. Fordelen med å bruke den implementasjonen hadde hvert at søking hadde tatt mindre tid. Linked-lists starter alltid fra samme punkt og går i en retning fram til den finner elementet. Binære søketre har derimot alltid to valg om hvilken vei den skal ta. Dette gjør at et søketre hadde ikke trengt å besøke like mange noder som en linked-list for å komme til samme element og dermed øker ytelsen. En av ulempene ved å bruke et binært søketre er at treet kan bli ubalansert avhengig av hvilken rekkefølge elementene blir satt inn. Når treet er ubalansert kan det redusere den potensielle ytelsen kraftig.

Ubalansert

Balansert

## 5 Konklusjon

Alt i alt gikk oppgaven ut på at man skulle implementere et sortert ADT og bruke det for å videre implementere spamfilteret. Her brukte jeg linked-lists for holde på elementene og mergesort for å sortere datasettet. Implementasjonen av de logiske operasjonene som datasettet skulle kunne utføre ble hovedsakelig implementert med list contains. Spamfilteret ble implementert ved å bruke lister og tokenize. Av oppgaven lærte jeg meg en hel del om sortering og abstrakte data typer. Hvordan man manipulerer sett og håndterer eksterne filer lærte jeg meg også. Denne erfaringen kommer til å være nyttig i fremtidige utfordringer som blir møtt.