# Dominante mønstre

## Introduksjon

Denne oppgaven er inspirert av en rekke studier gjort ved gruppa for Biomedisinsk Informatikk (BMI) ved Institutt for informatikk (IFI), Universitetet i Oslo (UiO). Datasettet er hentet via denne gruppa fra en studie som knytter en rekke immunreseptorer til cytomegalovirus, heretter CMV (Emerson et al., 2017).

Vi skal klargjøre data som kan brukes til å finne dominante mønstre i en rekke immunrepertoarer ved hjelp av parallellprogrammering. Immunrepertoaret er en del av menneskets immunforsvar, og består av mange ulike immunreseptorer (Scheffer, 2024). Immunreseptorer er spesialiserte proteiner som gjør det mulig for immunforsvaret å oppdage og respondere på fremmede, potensielt skadelige, stoffer. Hver immunreseptor har en unik sekvens av DNA-segmenter som binder seg til bestemte stoffer (antigener).

Når en person blir smittet med en infeksjon, vil immunforsvaret (blant annet) generere og utvide immunreseptorer som kan binde seg til stoffene i den aktuelle infeksjonen. Disse immunreseptorene blir en del av immunrepertoaret, og på denne måten lagres informasjonen om hvilke infeksjoner en person har vært smittet med. I denne oppgaven skal du skrive et program som klargjør data som kan brukes til å analysere slike immunrepertoarer.

## Nyttige lenker

Dokumentasjon:

- `File`
- `Map<K, V>`
- `TreeMap<K, V>`
- `ArrayList<E>`

Notater:

- Lesing og skriving i Java
- Notat om tråder

## Datasett

Vi deler immunrepertoarene inn i to grupper: De som tilhører personer vi vet har vært smittet med CMV, og de som tilhører personer vi vet ikke har vært smittet med CMV.

En person sitt immunrepertoar er representert som en fil. Hver linje i filen består av en sekvens av store bokstaver som utgjør DNA-sekvensen til én immunreseptor. En subsekvens er en sammenhengende delstreng av en DNA-sekvens.

Filene ligger i mappen `Data`, som er publisert på semestersiden. Der ligger også en fil `metadata.csv`, som inneholder navnene til alle filene som representerer immunrepertoarer i mappen, etterfulgt av en boolsk verdi `True` eller `False`. Verdien angir hvorvidt immunrepertoaret tilhører en person som har vært smittet med CMV (`True`), eller en person som ikke har vært smittet med CMV (`False`). I tillegg ligger det to filer, `smittet` og `ikke_smittet`, som inneholder det forventede resultatet av programmet.

Filene i mappen `TestData` (som også er publisert på semestersiden) følger akkurat samme struktur som filene i mappen `Data`, men inneholder et veldig lite datasett. Bruk dette for å teste programmet ditt.

## Oppgave 1 – `Frekvenstabell`

En frekvenstabell holder rede på en samling subsekvenser, og deres frekvenser (altså, hvor ofte de forekommer). Skriv en klasse med navn `Frekvenstabell` som arver fra `TreeMap<String, Integer>` fra Javas standardbibliotek.[1]

```
class Frekvenstabell extends TreeMap<String, Integer> {

    @Override
    public String toString() { /* ··· */ }

    public static Frekvenstabell flett(Frekvenstabell f1,
                                       Frekvenstabell f2) {
        Frekvenstabell flettet = new Frekvenstabell();
        /* ··· */
        return flettet;
    }

    public void skrivTilFil(String filnavn) { /* ··· */ }
}
```

En nøkkel (av typen `String`) i ordboka er en unik streng som representerer en subsekvens. Verdien (av typen `Integer`), som nøkkelen assosieres med, er et heltall som representerer subsekvensens frekvens.

[1] Merk at `TreeMap<K, V>` kan brukes akkurat som et `HashMap<K, V>`, siden begge implementerer grensesnittet `Map<K, V>`. Grunnen til at vi bruker `TreeMap<K, V>` her er at vi er garantert å få ut nøklene *ordnet alfabetisk* når vi itererer over dem.

(a) Overskriv metoden `toString()` slik at hver linje er et nøkkel- og verdipar, der nøkkelen og verdien er separert med mellomrom. Linjene skal være sortert etter nøkkel. For eksempel, en frekvenstabell der strengen `"en"` forekommer én gang, `"to"` forekommer to ganger og `"tre"` forekommer tre ganger skal resultere i en streng som vil skrives ut slik:

```
en 1
to 2
tre 3
```

(b) Skriv en statisk metode `flett(Frekvenstabell f1, Frekvenstabell f2)` som tar to frekvenstabeller `f1` og `f2` som argument. Metoden skal slå frekvenstabellen `f1` sammen med frekvenstabellen `f2` i en ny frekvenstabell. Frekvensene til en subsekvens som forekommer i begge frekvenstabeller skal summeres.

**Eksempel**

Under ser du et eksempler på to frekvenstabeller `f1` og `f2` og resultatet av å flette dem `Frekvenstabell.flett(f1, f2)`.

`f1`

```
ABC 1
BCD 1
CDB 1
DBC 1
EFG 1
FGH 1
```

`f2`

```
ABC 1
BAB 1
BCA 1
BCD 1
CAB 1
CDA 1
CDB 1
DAB 1
DBC 1
DEF 1
```

`flett(f1, f2)`

```
ABC 2
BAB 1
BCA 1
BCD 2
CAB 1
CDA 1
CDB 2
DAB 1
DBC 2
DEF 1
EFG 1
FGH 1
```

(c) Implementer metoden `skrivTilFil(String filnavn)`. Denne skal skrive resultatet av `toString()`-metoden til en fil med navnet gitt av `filnavn`.

## Oppgave 2 – `Subsekvensregister`

I deloppgavene nedenfor skal du implementere klassen `Subsekvensregister`. Klassen skal representere et register som holder rede på, og lar deg behandle, en samling av frekvenstabeller.

La `register` være en instansvariabel som referer til en dynamisk liste i klassen. Du kan eksempelvis bruke `ArrayList<E>` fra Javas standardbibliotek. Registeret skal inneholde frekvenstabeller av typen `Frekvenstabell`. I denne oppgaven skal vi kun

behandle subsekvenser av lengde `3`. La derfor et heltall `SUBSEKVENSLENGDE` være en statisk konstant i klassen, og sett den til verdien `3`. Ingen av instansvariablene skal kunne aksesseres direkte utenfor klassen de tilhører.

(a) Skriv følgende metoder i klassen:

- `settInn(Frekvenstabell f)` – setter inn en gitt frekvenstabell i registeret.
- `taUt()` – fjerner og returnerer en vilkårlig frekvenstabell fra registeret.
- `antall()` – returnerer antall frekvenstabeller i registeret.

Metodene skal kunne kalles på fra en hvilken som helst klasse.

(b) Skriv en statisk metode `les(String filnavn)` som leser inn data fra en fil med det spesifiserte filnavnet, og returnerer en frekvenstabell av typen `Frekvenstabell`. Metoden skal opprette en `Frekvenstabell f`, og fylle denne med alle unike subsekvenser av lengde `3`. Du kan anta at alle linjer inneholder minst tre tegn. Vi er ikke interessert i hvor ofte en subsekvens forekommer hos én og samme person. En subsekvens sin frekvens settes derfor til `1`.

Metoden skal kunne aksesseres fra en hvilken som helst klasse.

**Eksempel**

For en fil som innehodler følgende:

```
ABCDBCD
EFGH
```

skal den resulterende frekvenstabellen se slik ut:

```
ABC 1
BCD 1
CDB 1
DBC 1
EFG 1
FGH 1
```

## Oppgave 3 – `Monitor`

Skriv en klasse `Monitor` som koordinerer operasjoner på et subsekvensregister, og sørger for at disse utføres trådsikkert.

Initialisér en instans av `Subsekvensregister` i klassens konstruktør. Instansvariabelen skal kun være mulig å aksessere innenfor denne klassen. Bruk relevante klasser fra Javas bibliotek `java.util.concurrent`, og initialisér andre nødvendige instansvariabler.

`Monitor` skal inneholde de samme metodene som `Subsekvensregister`. Du skal ikke implementere stegene i disse metodene på nytt, men heller bruke monitorens

subsekvensregister til å kalle på dem.[2] I en senere oppgave skal du utvide denne klassen med flere metoder. Metodene skal kunne aksesseres fra en hvilken som helst klasse.

## Oppgave 4 – `Lesetråd`

Skriv en trådklasse `Lesetråd` som implementerer grensesnittet `Runnable`. Klassen skal ha to instansvariabler, `filnavn` og `monitor`, som refererer til en streng og en instans av klassen `Monitor`. Disse skal tas inn som argumenter og settes i klassens konstruktør. Tråden skal bruke monitoren til å lese fra filen med det gitte filnavnet, og sette den resulterende frekvenstabellen tilbake inn i monitorens register.

## Oppgave 5 – En utvidelse av `Monitor`

Du skal nå utvide klassen `Monitor` med flere metoder.

Skriv en metode `taUtTo()` i `Monitor` som tar ut og returnerer to frekvenstabeller til fletting. Legg merke til at det kan oppstå en midlertidig tilstand i programmet hvor antall frekvenstabeller i subsekvensregisteret er under to, selv om det ikke er tilfellet at alle frekvenstabeller har blitt flettet sammen til én. I så fall må du sørge for at trådene som forsøker å eksekvere denne metoden venter til tilstanden endrer seg, og det igjen er mulig å ta ut to frekvenstabeller.

Du kan velge å implementere flere metoder og/eller justere eksisterende metoder for å få til en slik programflyt. Det kan for eksempel være lurt å definere en ny metode `settInnFlettet(Frekvenstabell f)` som setter inn resultatet fra flettingen, men du står fritt til å velge den implementasjonen som faller mest naturlig for deg.

## Oppgave 6 – `Flettetråd`

Skriv en trådklasse `Flettetråd` som implementerer grensesenittet `Runnable`. Klassens konstruktør tar en referanse til et `Monitor`-objekt inn som argument, og denne skal tilordnes en instansvariabel. Tråden skal bruke monitoren til å hente ut to frekvenstabeller, flette disse sammen og sette resultatet fra flettingen tilbake inn i monitorens register. Trådens oppgave skal utføres i en løkke som først terminerer når alle frekvenstabeller i subsekvensregisteret har blitt flettet sammen til én.

## Oppgave 7 – `KlargjørData`

Du skal nå bruke klassene du har skrevet til å sette sammen et hovedprogram som identifiserer unike subsekvenser, og teller frekvensen av disse hos de to gruppene i datasettet. Resultatene skal skrives til fil.

[2] For statiske metoder bør kallet gjøres på klassen, og ikke instansen.

(a) Skriv en hovedklasse `KlargjørData` som tar en filsti som kommandolinjeargument. Klassen skal opprette to monitorer, én for de som har vært smittet med CMV, og én for de som ikke har vært smittet med CMV. Programmet skal opprette og starte én lesetråd for hvert filnavn i metadata-filen. Sørg for at lesetråden oppretter og setter inn frekvenstabellen i riktig monitor. Programmet skal kunne testes med filstien til metadata.csv som kommandolinjeargument.

Merk at det kan være lurt å opprette et `File`-objekt for filnavnet som er gitt som kommandolinjeargument. Det gjør det enklere å hente ut mappenavnet, som vist i følgende lille kodesnutt:

```
String filnavn = args[0];
File fil = new File(filnavn);
String mappe = fil.getParent() + "/";
```

(b) La et heltall `ANTALL_TRÅDER` være en statisk konstant i hovedklassen, og sett den til verdien 8. Denne skal brukes når du oppretter flettetråder. Du må gjerne eksperimentere med ulike verdier for antall tråder.

Når alle dataene er lest inn, skal programmet starte flettingen av frekvenstabeller. Du skal opprette og starte åtte flettetråder for hver monitor. Trådene skal flette frekvenstabeller til du står igjen med én frekvenstabell per monitor. Det resulterer i én frekvenstabell med data fra personer som har vært smittet med CMV, og én med data fra de som ikke har vært smittet med CMV.

(c) De to resulterende frekvenstabellene skal skrives til fil; den ene med navn `"smittet"` og den andre med navn `"ikke_smittet"`.

## Referanser


Emerson, R., DeWitt, W., Vignali, M., et al. (2017). Immunosequencing identifies signatures of cytomegalovirus exposure history and HLA-mediated effects on the T cell repertoire. Nature Genetics, 49(5), 659–665. https://doi.org/10.1038/ng.3822

Scheffer, L. (2024). Machine learning and computational analyses of adaptive immune receptors (Doktorgradsavhandling). Universitetet i Oslo. https://www.duo.uio.no/handle/10852/107194