Versjon 22. mars 2022

Obligatorisk oppgave 5, 2022

Om å lete etter og behandle mønstre i immunrepertoarer.

Introduksjon.

I denne oppgaven skal du skrive et program som ved å analysere blodprøver finner sekvensmønstre som indikerer en infeksjon av et bestemt virus. For å få til dette vil vi analysere immunrepertoarene - DNA-sekvenser av immunceller i blodet - til personer som vi vet er blitt smittet eller ikke er blitt smittet av et bestemt virus.

Et immunrepertoar består av mange immunreseptorer som er proteiner som gjenkjenner viruset. Immunrepertoaret til én person er representert av én fil, og hver immunreseptor er representert som en sekvens av (store) bokstaver, en sekvens per linje i filen. Vi ønsker å analysere og finne mønstre i disse sekvensene fra folk som har hatt og som ikke har hatt viruset. Mønstrene vi ser etter er substrenger (typisk 3 eller fire bokstaver) av reseptorsekvensene. Til slutt, etter å ha funnet antall forekomster av alle subsekvensene, vil vi avgjøre hvilke mønstre som i sterkest grad forekommer fortrinnsvis hos personer som vi vet har hatt viruset. Disse dominante mønstrene kan deretter brukes til å diagnostisere nye personer mistenkt for å være smittet av det bestemte viruset. Oppgaven gjenspeiler en tilnærming som i prinsippet kan brukes til å diagnostisere infeksjon av virus som SARS-CoV-2, samt potensielt forbedre diagnostiseringen av autoimmune sykdommer og kreft.

Mønstrene vi er på jakt etter i immunrepertoarene er altså korte like substrenger eller *subsekvenser*. En subsekvens er en String av en gitt lengde, typisk 3 eller 4 (en konstant i programmet ditt), f.eks. «ABC». Når vi analyserer blodet til én person (en fil) finner vi svært mange slike subsekvenser. Vi er interessert i å finne mønstre (subsekvenser) som forkommer i mange personer. Om en subsekvens forkommmer bare én gang eller mange ganger i samme person (samme fil) vil dette ikke ha noen betydning. Det er bare når samme subsekvens forekommer i flere personer (flere filer) at funnet blir interessant.

Alle subsekvensene til en person (typisk mange hundre) lagrer vi én HashMap. Når vi analyserer blodet til N personer (leser N filer) får vi N HashMap-er. For å finne subsekvensene som forekommer hos mange personer skal vi slå sammen eller flette HashMap-er. Dette blir nærmere forklart nedenfor.

Del 1: Om å finne antall like subsekvenser uten parallellitet

Oppgave 1.

Skriv en klasse, kall den gjerne Subsekvens, som kan ta vare på en subsekvens (en String) og et antall (en int som angir antall forekomster av denne subsekvensen hos flere personer). La instansvariablen antall være private og la subsekvens være public men gjør den til en konstant ved bruk av 'final'. Lag metoder for å hente hente og endre antall forekomster.

Lag en toString-metode som oppfyller følgende krav:
Hvis verdien av subsekvensen i et Subsekvens-objekt f.eks. er ABC og den forekommer hos 4 personer, vil vi i denne oppgaven beskrive dette objektet og skrive det ut slik: (ABC,4).

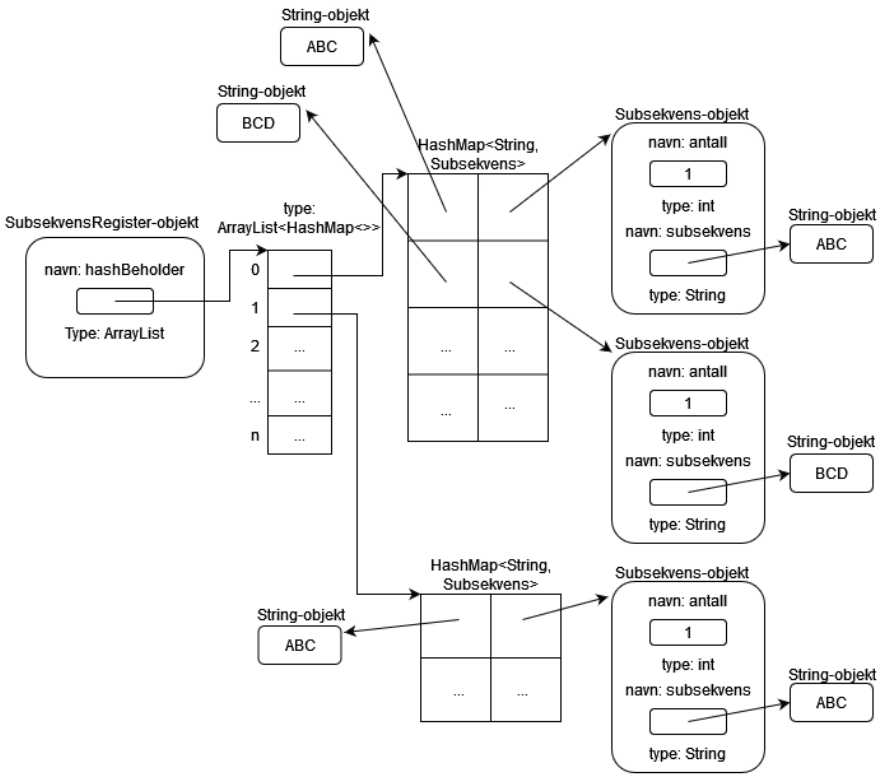
Oppgave 2.

HashMapene som subsekvensene lagres i skal være av typen HashMap<String,Subsekvens>. Lag en enkel beholder, kalt SubsekvensRegister, som kan ta vare på mange slike HashMap-er. Det du skal kunne gjøre med denne beholderen er:

- Sette inn en slik HashMap
- Ta ut en slik HashMap
- Spørre hvor mange HashMap-er den innholder

Når du tar ut en HashMap av beholderen kan du ta ut en vilkårlig en. Hint: Lag en beholder der elementene i beholderen (som er HashMap-er) er lagret i en ArrayList.

Datastrukturtegning-eksempel av SubsekvensRegister:



Oppgave 3.

I SubsekvensRegister-klassen du skrev i oppgave 2 skal du lage en statisk metode som leser én fil med én persons immunrepertoar og lager en HashMap av subsekvensene i filen. Metoden skal returnere en referanse til den nye HashMap-en.

Hvor mange ganger en subsekvens forekommer i én persons immunrepertoar bryr vi oss ikke om, vi skal bare finne hvilke (forskjellige) substrenger som finnes hos denne personen (i denne filen). Alle Subsekvens-objektene som lages i denne metoden vil derfor ha verdien 1 i instansvariablen antall. Metoden leser én og én linje, finner subsekvensene i linjen og legger disse inn i HahMap-en. Hvis hver subsekvens består av tre tegn vil det alltid være to færre subsekvenser enn det er tegn på en linje.

Hvis en subsekvens som leses allerede forekommer i HashMap-en skal den ikke lagres på nytt (og antallet skal forbli 1). Stopp programmet om en linje er kortere enn 3 tegn.

Test metoden din ved å bruke dataene i de to eksemplene under og dataene som finnes på oblig-siden . Skriv ut innholdet av den returnerte HashMap-en med en for-each-løkke og toString metoden i Subsekvens.

I tillegg til de to eksemplene nedenfor er det et eksempel helt på slutten av oppgaven i Appendikset.

Fil1:
ABCDBCD
EFGH

Denne filen gir disse subsekvensene av lengde 3:
ABCDBCD gir ABC BCD CDB DBC BCD
EFGH gir EFG FGH

Og vi får en HashMap med 6 elementer:
(ABC,1) (BCD,1) (CDB,1) (DBC,1) (EFG,1) (FGH,1)

Fil 2:
ABCDABCDBCD
DEF
BABCABC

Denne filen gir disse subsekvensene av lengde 3:
ABCDABCDBCD gir ABC BCD CDA DAB ABC BCD CDB DBC BCD

DEF gir DEF
BABCABC gir BAB ABC BCA CAB ABC

Og vi får en HashMap med 10 elementer:
(ABC,1) (BCD, 1) (CDA,1) (DAB,1) (CDB,1) (DBC, 1) (DEF,1) (BAB,1) (BCA,1) (CAB,1)

Oppgave 4.

I SubsekvensRegister-klassen du skrev i oppgave 2 skal du lage en statisk metode som slår sammen to HashMap-er som tar vare på Subsekvens-objekter. Parametrene til metoden er referanser til de to HashMap-ene som skal slås sammen, mens returverdien skal være en referanse til en HashMap som er en sammenslåing (en fletting) av de to parametrene. Resultatet består av alle objektene i de to HashMap-ene som ble tatt inn som parametere, men der subsekvensene er like skal det kun lagres ett objekt i resultatet. Antall forekomster i dette objektet skal være summen av forekomstene av subsekvensen i de to HashMap-ene som slås sammen.

Om vi fletter de to hashMapene vi fikk fra de to filene over får vi:
(ABC,2) (BCD,2) (CDA,1) (CDB,2) (DBC,2) (EFG,1) (FGH,1) (BAB,1) (DAB,1) (BCA,1) (CAB,1)

Oppgave 5.

Skriv et testprogram som tester filinnlesingen fra oppgave 3 sammen med flettingen i oppgave 4. Bruk filene i de to mappene *TestDataLike og TestDataLitenLike*. I hele oblig5 skal navnet på (eventuelt inkludert stien til) mappen der datafilene ligger være en parameter til programmet. I alle mapper er det en fil ved navn *metadata.csv,* som inneholder navnene på immunrepertoar-filene som skal leses.

Testprogrammet ditt skal opprette et SubsekvensRegister der alle HashMap-er som lages av metoden du skrev i oppgave 3 legges inn. Bruk deretter metoden du skrev i oppgave 4 til å flette alle HashMap-ene i beholderen. Når det er én HashMap igjen skriver du ut subsekvensen som har flest forekomster (antall). I TestDataLitenLike bør dette være ASS med 3 forekomster og i TestDataLike, QYF med 5 forekomster.

Lever alle klassene i en zippet mappe kalt Oblig5Del1, men ikke .class-filer og ikke datafiler. Kall hovedprogrammet Oblig5Del1.java

Del 2: Finne mønstre med tråder

I denne delen av oppgaven skal du gjenbruke alle klassene du laget i del 1 men du skal utvide oppgaven litt samtidig som du skal gjøre deler i parallell (med tråder). Vi skal introdusere tråder i to steg: Først skal du lage et program med tråder bare for å lese og behandle filene, senere skal vi også bruker tråder til å flette.

I første omgang skal vi gjøre analysen av immunrepertoarene til mange personer raskere ved at hvert immunrepertoar (hver fil) behandles av en tråd. Ønsker du en større utfordring kan du anta at denne jobben skal gjøres av K tråder, der K er mindre enn antall filer. Når en tråd har fullført behandling av en fil og opprettet en hashmap, så behandler den en ny fil og lager en ny hashmap, etc. Bare når alle filer er ferdig behandlet kan disse trådene avsluttes. For å få til dette må du først løse oppgave 6.

Oppgave 6

Ved hjelp av komposisjon, bygg en monitor rundt SubsekvensRegisteret du skrev i oppgave 2, slik at bare én tråd kan sette inn en HashMap om gangen. Kall monitoren Monitor1. Monitor1 skal tilby akkurat de samme metodene som SubsekvensRegister.

Oppgave 7

Skriv en trådklasse, kalt LeseTrad, som leser en fil og legger den resulterende HashMap-en inn i en beholder av klassen Monitor1. En referanse til filnavnet og en referanse til monitoren skal være parametre til konstruktøren til klassen.

Oppgave 8

Skriv om hovedprogrammet ditt fra del 1 skal at main-metoden starter opp mange tråder for å lese alle filene og legge de resulterende HashMap-ene inn i ett (og det samme) objekt av klassen Monitor1. Main-metoden skal deretter flette de resulterende HashMap-ene på samme måte som i del1, eneste forskjell er at nå ligger disse inne i Monitor1-objektet. Men siden bare én tråd (main-tråden) foretar fletting behøver ikke metodene som henter ut HashMap-er fra Monitor1 beskyttes mot bruk fra flere tråder samtidig.

Legg denne main-metoden i en klasse du kaller Oblig5Del2A.java og sørg for at dette programmet kan kjøres i den mappen du leverer. Kall mappen Oblig5Del2A og zip den. Bruk de samme test-filene og utskrift som i oppgave 5 (men ikke lever data- eller class.-filer). Husk at mappenavnet til filene som skal leses skal være en parameter til programmet.

Oppgave 9, 10 og 11

Vi skal nå gjøre ferdig det fulle programmet som bruker tråder både til lesing fra fil og til fletting. I oppgave 9 skal du skrive enda en trådklasse, kalt FletteTrad, som henter ut to og to HashMap-er, fletter disse og legger resultatet tilbake. I oppgave 10 skal du skrive den fulle monitoren (kalt Monitor2) som beskytter HashMap-en mot samtidig bruk både fra tråder som leser filer og tråder som fletter. En overordnet figur som viser hvordan dette virker finner du i figuren lenger nede i oppgaveteksten.

Oppgave 9

Skriv trådklassen FletteTrad som henter ut to HashMap-er fra et monitor-objekt av klassen Monitor2, slår HashMap-ene sammen og legger resultatet tilbake i monitoren. Først når det bare er én HashMap igjen i monitoren, terminerer disse trådene. Antall tråder av trådklassen FletteTrad skal være en konstant i programmet ditt.

Oppgave 10

Det du trenger i denne oppgaven foreleses først 29. mars.

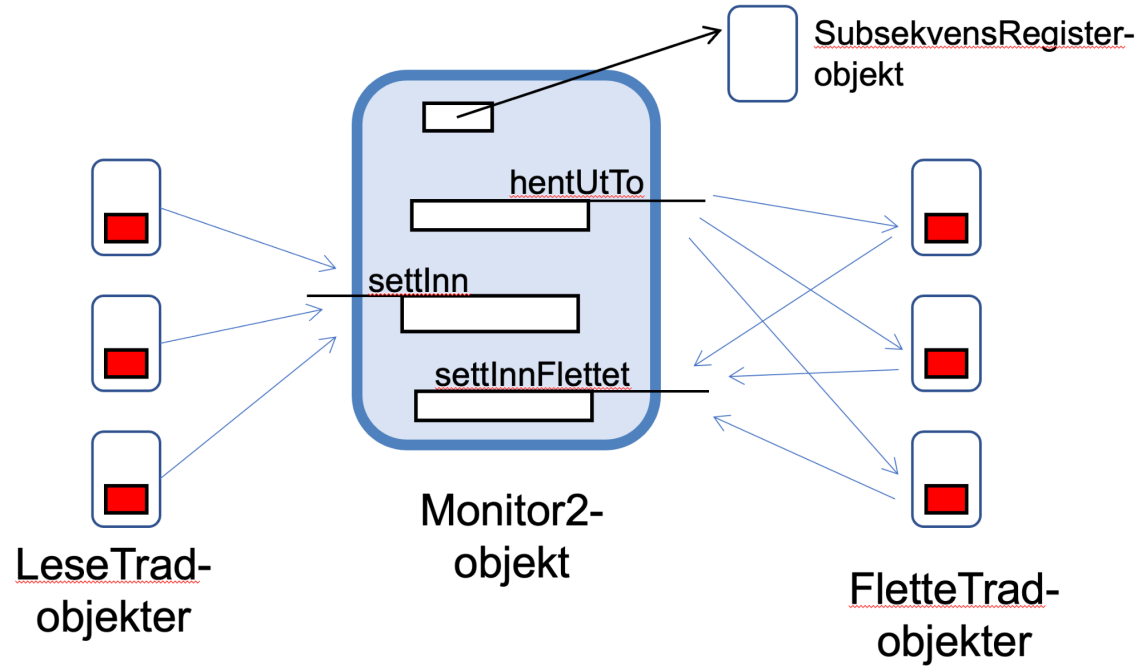
Skriv monitoren Monitor2. Bygg videre på Monitor1, men du får nå to hovedutfordringer: Du må lage én metode som henter ut to HashMap-er som skal flettes. Det vil si at metoden må vente inne i monitoren hvis den ikke inneholder to HashMap-er som kan hentes ut. Til slutt vil det bare være én HashMap igjen i beholderen – den HashMap-en som er resultatet av all flettingen. Å identifisere når det bare er én igjen og flettetrådene ikke har mer å gjøre er en fin utfordring for dine logiske evner.

Dette kan gjøres på svært mange måter og du må gjerne snakke med andre om denne problemstillingen. Men du må finne på din egen måte å gjøre dette på og ikke kopiere en måte fra andre. Det vil være svært enkelt å se om du har kopiert en annens løsning.

Relevant Trix-oppgave: 10.05

Oppgave 11

Skriv en main()-metode i en klasse Oblig5Del2B.java. Denne main()-metoden skal gjøre det samme som i oppgave 8 og bruk de samme testdataene og samme testutskrift. Du bør jo få de sammen svarene på alle testene du har gjort inntil nå i denne obligatoriske oppgaven. Lever alle .java-filene du trenger i en zippet mappe kalt Oblig5Del2B (men ikke .class- eller datafiler). Husk at mappenavnet til filene som skal leses skal være en parameter til programmet.



Oppgave 12 Fullt program

For å finne mønstre som indikerer om en person har vært smittet, skal vi dele personene (og filene) inn i to grupper: De filene som representerer immunreseptoarne til personer vi vet har hatt den sykdommen (det viruset) vi er på jakt etter å analysere og de filene som representerer immunreseptoarne til personer vi regner med ikke har hatt denne sykdommen. I metadatafilen i mappene TestDataLite og TestData (og i mappen Data med de virkelige dataene), står det på slutten av hver linje (etter filnavnet) «True» for de personen/filene som har hatt viruset, og «False» for de som ikke har hatt det.

Skriv en ny klasse, Oblig5Hele.java der main()-metoden oppretter to Monitor2-objekter, dvs. et objekt med en beholder for HashMap-er fra personer som har hatt sykdommen, og et objekt med en beholder for de som ikke har hatt sykdommen. Deretter opprettes det tråder som leser filer og, som på samme måte som før, legger resultatet i den ene eller den andre beholderen, avhengig av om filen er fra en som har hatt sykdommen eller ikke.

Så opprettes det et antall tråder av klassen FletterTrad som har som oppgave å flette HashMap-er i den ene beholderen og like mange tråder som fletter i den andre beholderen. Dette antallet er fremdeles en parameter til programmet.

Resultatet er nå at vi har to Monitor2 objekter der vi kan hente ut en HashMap fra hver, én HashMap med subsekvenser fra personer som har vært smittet og én HashMap med subsekvenser fra friske personer.

Programmet ditt skal avslutte med å finne det vi kaller dominante subsekvenser. Dette kan du gjøre på en enkel (men ikke så sikker) eller på en mer avansert måte. Det er bare den enkle måten som er obligatorisk. Den gode, statistiske måten er beskrevet lenger nede. Den enkle metoden går ut på å skrive ut alle subsekvenser der antall forekomster hos personer med viruset er mye høyere enn antall forekomster hos personer som ikke har hatt viruset.

Bruk først testdataene i mappene TestDataLiten og TestData og skriv ut de subsekvensene som forekommer oftest blant de som har vært smittet i forhold til de som ikke har vært smittet. I TestDataLiten bør dette være ABC med 2 flere forekomster og i TestData, GAE med 5 flere forekomster. Kjør til slutt programmet på de virkelige dataene i mappen Data og skriv ut alle subsekvenser som forekommer 7 flere ganger eller ofteren blant de som har hatt viruset enn de som har vært friske.

Lever alle klassene inklusive Oblig5Hele.java som skal inneholder main() i en zippet mappe kalt Oblig5Hele. Ikke lever .class-filer og ikke data-filer. Retteren din skal kunne oversette og kjøre programmet ditt og bruke stien til sine datafiler som parameter til programmet.

Hvis du vil kan du programmere en bedre statistisk test på de virkelige dataene:

Bruk en binomial test for å beregne hvilke mønstre som er dominant hos personer som har hatt viruset. Den binomial testen skal utføres for hver subsekvens. Bruk følgende parameterverdier:

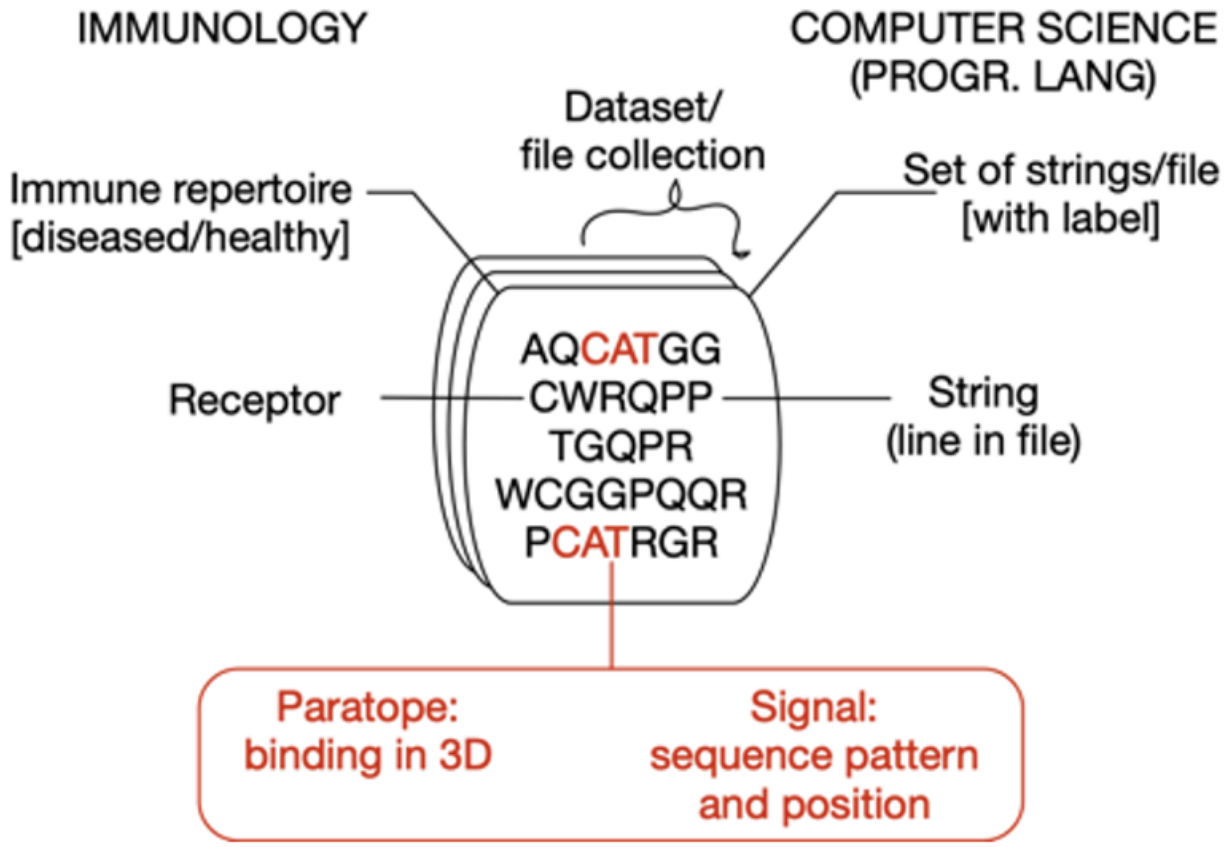
- Antall forsøk: Dette er det totale antallet repertoarer subsekvensen finnes i, som kommer fra både personer som hadde viruset og folk som ikke hadde.
- Antall suksesser: Dette er antallet i repertoarer som kommer fra personer som hadde viruset.
- Sannsynligheten for suksess: Hvor sannsynlig det er for subsekvensen å forekomme hos syke mennesker. Denne kan settes til 0,5, noe som betyr at det er like sannsynlig å forekomme som å ikke forekomme.
- Typen hypotese som evalueres: Her ser vi bare på subsekvenser som er mer sannsynlig å forekomme hos syke mennesker, så en ensidig test kan brukes her.

Testen gir en p-verdi, og bare subsekvenser med en p-verdi under en terskelverdi (som kan være 5%) vil bli valgt som dominante mønstre. Terskelen kan være en konstant i programmet.

En implementasjon av en binomial test er tilgjengelig i Apache Commons Math biblioteket, se denne linken for referanse.

Når alle subsekvenser er testet, skal programmet avslutte med å skrive ut de dominante mønstrene.

Appendiks



I dette eksemplet inneholder én fil 5 sekvenser. Vi bruker subsekvenser av lengde 3:

AQCATGG 7 bokstaver 5 subsekvenser

CWRQPP 6 bokstaver 4 subsekvenser

TGQPR 5 bokstaver 3 subsekvenser

WCGGPQQR 8 bokstaver 6 subsekvenser

PCATRGR 7 bokstaver 5 subsekvenser

Resultatet fra denne personen vil være disse 23 subsekvensene:

AQC QCA CAT ATG TGG CWR WRQ . . . QQR PCA CAT ATR TRG RGR

CAT er den eneste subsekvensen med flere forekomster, så dette vil resultere i en hashmap med 22 oppføringer, hver med antallet én.