Versjon 1.1, 2024-04-16

Obligatorisk oppgave 2, IN1010, 2024 Tråder

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 i én fil, og hver immunreseptor er representert som en sekvens av (store) bokstaver, én 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 tre 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 vi mistenker er smitter 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 leter 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 som du i din besvarelse kan sette til 3), 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 forekommer hos mange personer. Om en subsekvens forekommer bare én gang eller mange ganger hos samme person (samme fil), vil det ikke ha noen betydning. Det er bare når samme subsekvens forekommer hos flere personer (flere filer) at funnet blir interessant.

Alle subsekvensene til en person (typisk mange hundre) lagrer vi i én HashMap. Når vi analyserer blodet til N personer (leser N filer) får vi N HashMap-er. Det å lese N filer og lage N HashMap-er kaller vi steg 1 i algoritmen. For å finne subsekvensene som forekommer hos mange personer skal vi slå sammen eller flette HashMap-er. Dette kaller vi steg 2 i algoritmen, og det blir nærmere forklart nedenfor. Oppgaveteksten har to deler: Del 1 (oppgave 1-5) er uten tråder, mens del 2 (oppgave 6-12) er med tråder.

Datafiler, mapper og format

Etter hvert skal programmet ditt lese inn data fra filer som ligger i ulike mapper, med navn som angir om det er det endelige datasettet (mappen *Data*), små filer med testdata (f.eks. *TestDataLiten* og *TestDataLitenLike*) eller større mengder testdata (*TestData* og *TestDataLike*). I hver mappe ligger det en fil med navn *metadata.csv*; denne inneholder navn på filene som skal leses, ett filnavn på hver linje. Filen *metadata.csv* i mappene med navn som slutter på "*Like*" inneholder kun et filnavn på hver linje, mens metadata-filene i mappene *Data*, *TestData* og *TestDataLiten* inneholder et ekstra felt per linje med TRUE eller FALSE - disse trenger du ikke bruke før i oppgave 12. Bruk gjerne split() for å hente ut det du trenger fra hver linje.

Alle mappene ligger som .zip-filer i en felles mappe *Data* (samme sted som denne oppgaveteksten) på semestersiden, og må lastes ned og pakkes ut.

Det du leverer

Det du leverer skal kompilere feilfritt og kjøre i terminalen - det er *ikke* tilstrekkelig at for eksempel Visual Studio Code klarer å kjøre testprogrammer. Det er bedre at det du leverer inneholder mangler enn at det ikke kompilerer og kjører. Om innleveringene dine ikke kompilerer og kjører, vil de ikke bli vurderte.

Det endelige og fullstendige programmet du skal lage er spesifisert i oppgave 12 (Oblig2Hele). De andre oppgavene bygger opp løsningen skritt for skritt. Når du programmerer disse oppgavene, skal du også bygge dem opp skritt for skritt og passe på at de alltid kompilerer og kjører. Ta vare på kjørende versjoner av programmene dine *som om du skal levere* etter oppgave 8 (Oblig2Del2A) og 11(Oblig2Del2B), selv om du jobber videre med å løse senere oppgaver. Da kan du alltid levere et kjørende program og få et nytt forsøk om det du leverer er et seriøst forsøk på å løse obligen.

Del 1: Om å finne antall like subsekvenser uten parallellitet

Oppgave 1

Skriv en klasse Subsekvens som kan ta vare på en subsekvens (en String) og et antall (et heltall 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 og endre antall forekomster.

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

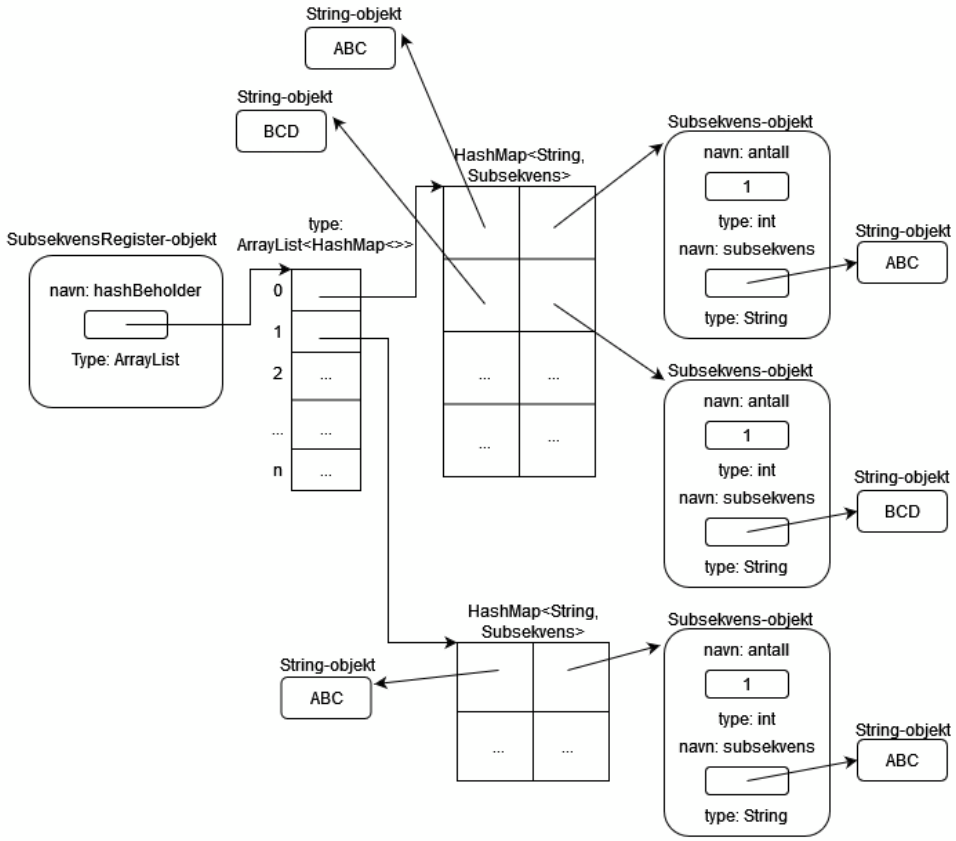
Oppgave 2

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, for eksempel i en ArrayList. Det du skal kunne gjøre med beholderen SubsekvensRegister er:

- Sette inn en HashMap med subsekvenser
- Ta ut en vilkårlig HashMap med subsekvenser
- Spørre hvor mange HashMap-er den innholder

Skriv klassen SubsekvensRegister.

Datastrukturtegning som viser et eksempel på 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 HashMap-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 tre – 3 – tegn (lengden av subsekvensene).

Nedenfor ser du to eksempler (disse finner du i *fil1* og *fil2* i mappen *TestOppgaveEks*). Hvis du vil kan du teste metoden din på disse dataene og se om du får samme svar. Du kan også teste ved å bruke datafilene i de andre mappene (se avsnitt om datafiler og formater i starten av denne oppgaven). Skriv for eksempel 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 denne oppgaveteksten (i appendikset).

Fil 1:
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 (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.

I dette tilfellet skal de gamle HashMap-ene og Subsekvens-objektene ikke brukes videre, så du kan gjerne oppdatere disse til bruk i den sammenslåtte HashMap-en.

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) (DEF,1) (BAB,1)
(DAB,1) (BCA,1) (CAB,1)

Oppgave 5

Skriv et testprogram, Oblig2Del1.java, som tester filinnlesingen fra oppgave 3 sammen med flettingen i oppgave 4. Bruk filene i de to mappene *TestDataLike og TestDataLitenLike*. I hele Oblig2 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 ett SubsekvensRegister, der alle HashMap-er som lages av metoden du skrev i oppgave 3 legges inn (steg 1 i algoritmen). Bruk deretter metoden du skrev i oppgave 4 til å flette alle HashMap-ene i beholderen (steg 2). Dette gjør du ved å ta ut to og to HasMap-er, flette dem og legge resultatet tilbake i SubsekvensRegisteret. Når det bare 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 bør dette være QYF med 9 forekomster.

Ta gjerne vare på det du har laget så langt i en mappe kalt Oblig2Del1, før du videreutvikler programmet ditt ved bruk av tråder i del 2.

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 først introdusere tråder ved å parallellisere steg 1. Senere skal vi også bruker tråder til å flette (steg 2).

Tips: Programmeringen av flettingen i steg 2 blir enklere om du lar alle LeseTrad-ene (steg 1) bli ferdige før du starter å flette i steg 2. Husk også at du må starte alle LeseTrad-ene (med kall på start()) før du begynner å vente på at de skal være ferdige (f.eks. med join() eller en annen barriere).

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.

Tips: Husk at komposisjon betyr *sammensetning*. Monitor1 skal derfor ikke implementere metodene fra SubsekvensRegister på nytt, men inneholde en peker til et SubsekvensRegister-objekt og kalle metodene i dette objektet (*delegere*).

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, slik at main-metoden starter opp mange tråder for å lese alle filene og legge de resulterende HashMap-ene inn i ett (og det samme) objektet av klassen Monitor1 (steg 1 i algoritmen). Det enkleste er å starte opp en tråd for hver fil. Når alle disse trådene er ferdige, skal main-metoden fortsette, og flette alle de resulterende HashMap-ene (del 2 i algoritmen) på samme måte som i oppgavene 4 og 5. Eneste forskjell er at nå ligger HashMap-ene som skal flettes 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 main-metoden i en klasse du kaller Oblig2Del2A.java og sørg for at dette programmet kan kjøres i den mappen du leverer. Kall mappen Oblig2Del2A og zip den. Bruk de samme test-filene og utskrift som i oppgave 5 (men ikke lever data- eller .class-filer). Retteren din skal kunne kompilere og kjøre programmet ditt *og bruke stien (mappe og filnavn) til sine datafiler som parameter til programmet*.

Oppgave 9, 10 og 11

Vi skal nå gjøre ferdig det fulle programmet som bruker tråder både til steg 1 og til steg 2 i algoritmen; det vil si til både lesing fra fil og 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 lenger nede i oppgaveteksten.

Oppgave 9

Skriv trådklassen FletteTrad, som går i løkke, og for hvert gjennomløp henter ut to HashMap-er fra et monitor-objekt av klassen Monitor2, slår HashMap-ene sammen og legger resultatet tilbake i monitoren. Først når alle HashMap-ene er flettet til én HashMap, terminerer disse trådene. Antall tråder av trådklassen FletteTrad skal være en konstant i programmet ditt; den kan du kan sette til 8.

Oppgave 10

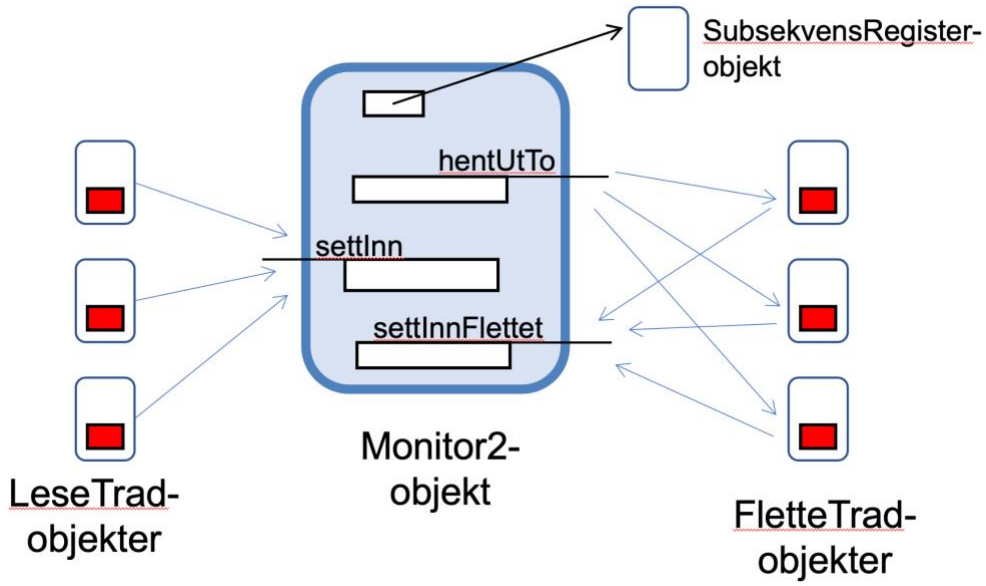
Skriv monitoren Monitor2 (se figuren). Bygg videre på Monitor1, men du får nå to hovedutfordringer: Du må lage én metode som henter ut to HashMap-er som skal flettes. Det vil si at metoden må vente inne i monitoren hvis den ikke inneholder to HashMap-er som kan hentes ut. Til slutt vil det bare være 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. Ikke kopier andres kode, og om du har funnet på løsningen sammen med andre, så skriv hvem du har samarbeidet med.

Relevant Trix-oppgave: 13.05

Oppgave 11

Bruk også det du har programmert i oppgavene 9 og 10, og skriv en main()-metode som gjør det samme som i oppgave 8. Bruk de samme testdataene og samme testutskrift. Du bør få de samme svarene på alle testene du har gjort inntil nå i denne obligatoriske oppgaven. Kall hovedklassen Oblig2Del2B.java og lever alle klassene i en mappe som heter Oblig2Del2B og zip den (uten data- og .class-filer). Retteren din skal kunne kompilere og kjøre programmet ditt *og bruke stien (folder og filnavn) til sine datafiler som 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 (viruset) vi er på jakt etter å analysere, og filene som representerer immunreseptoarne til personer vi regner med ikke har hatt sykdommen. I metadatafilen i mappene TestDataLiten og TestData (og i mappen Data med de virkelige dataene), står det «True» på slutten av hver linje (etter filnavnet) for personene/filene som har hatt viruset, og «False» for de som ikke har hatt det.

Skriv en ny klasse, Oblig2Hele.java, der main()-metoden oppretter to Monitor2-objekter; det vil si 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 FletteTrad, som har som oppgave å flette HashMap-er i den ene beholderen og like mange tråder som fletter i den andre beholderen. Bruk den samme konstanten som tidligere for hvert antall (slik at du får 8 + 8 flettetråder).

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 eller på en mer avansert måte. Den enkle måten 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. Den mer statistisk korrekte måten er beskrevet lenger nede og er frivillig å implementere.

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 bør dette være GAE med 5 flere forekomster. Kjør til slutt programmet på de virkelige dataene i mappen Data, og skriv ut alle subsekvenser som forekommer minst 7 flere ganger blant de som har hatt viruset enn blant de som har vært friske.

Lever alle klassene, inklusive Oblig2Hele.java som skal inneholde main(), i en zippet mappe kalt Oblig2Hele. Ikke lever .class-filer og ikke datafiler (.txt-filer). Retteren din skal kunne kompilere og kjøre programmet ditt *og bruke stien (folder og filnavn) 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 binomiale 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.

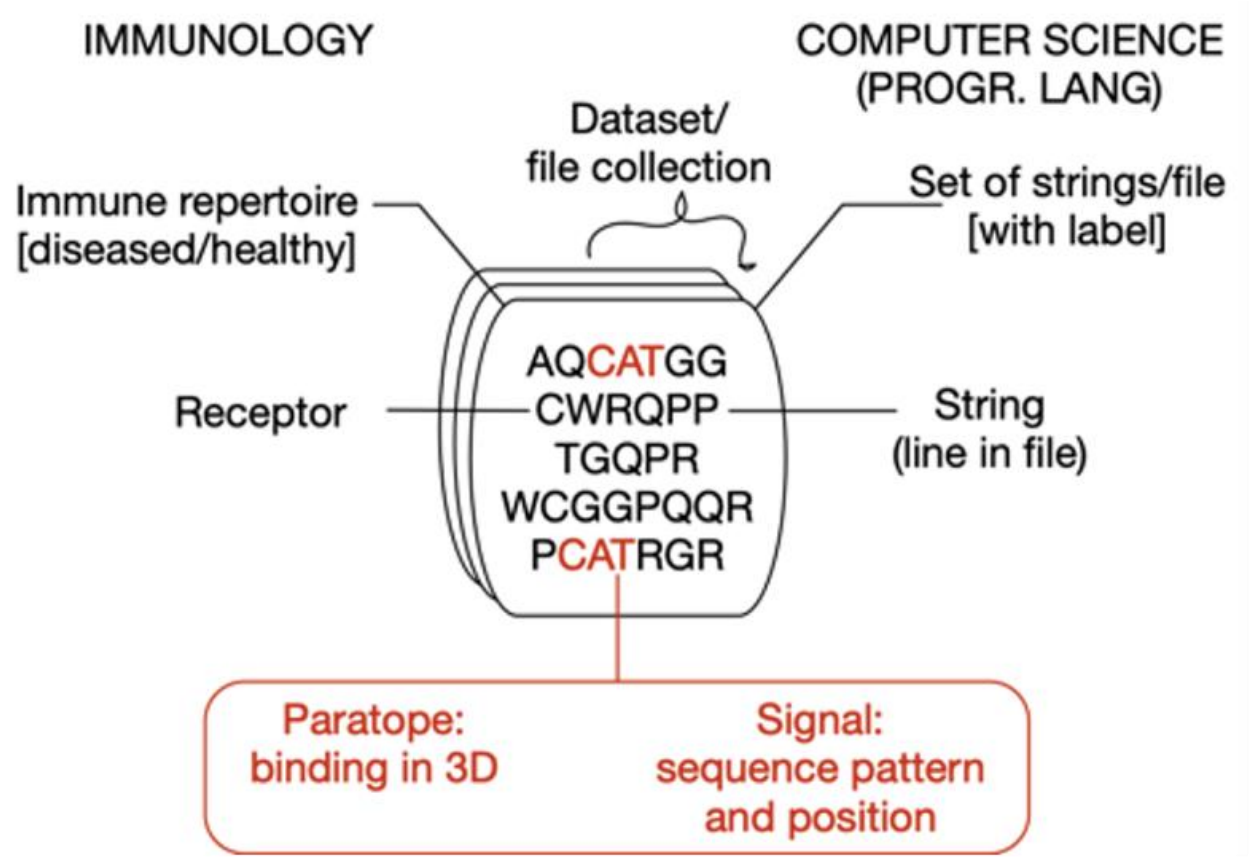
Når alle subsekvenser er testet, skal programmet avslutte med å skrive ut de dominante mønstrene.

Guide binomialtest fra apache

Enkel binomialtest kode

Appendiks

7



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.