

Problem-ID på Kattis: [kth.progp.f2](#)

Observera att i denna labb är det du och en eventuell labbpartners uppgift att välja vilka datatyper som ska lagras, att namnge och utforma eventuella hjälpfunktioner, att komma på lösningsmetoder och till sist implementera den. Det är ett brott mot hederskodexen att be personer utanför labblaget med hjälp om detta, eller att använda artificiella intelligenser. Listan med förbjudna artificiella intelligenser inkluderar men är inte begränsad till ChatGPT och GitHub Copilot.

för $\alpha \leq 0.94$ och $d_{a,b} = 3.7$ annars. Parametrarna är alltså ändrade för att reflektera det större alfabetet hos proteinsekvenser.

Uppgifter

1. Skapa en datatyp `MolSeq` för molekylära sekvenser som anger sekvensnamn, sekvens (en sträng), och om det är DNA eller protein som sekvensen beskriver. Du behöver inte begränsa vilka bokstäver som får finnas i en DNA/protein-sträng.
2. Skriv en funktion `string2seq` med typsignaturen `String -> String -> MolSeq`. Dess första argument är ett namn och andra argument är en sekvens. Denna funktion ska automatiskt skilja på DNA och protein, genom att kontrollera om en sekvens bara innehåller A, C, G, samt T och då utgå ifrån att det är DNA.
3. Skriv tre funktioner `seqName`, `seqSequence`, `seqLength` som tar en `MolSeq` och returnerar namn, sekvens, respektive sekvenslängd. Du ska inte behöva duplicera din kod beroende på om det är DNA eller protein!
4. Implementera `seqDistance :: MolSeq -> MolSeq -> Double` som jämför två DNA-sekvenser eller två proteinsekvenser och returnerar deras evolutionära avstånd.

Om man försöker jämföra DNA med protein ska det signaleras ett fel med hjälp av funktionen `error`.

Du kan anta att de två sekvenserna har samma längd, och behöver inte hantera fallet att de har olika längd.

3 Profiler och sekvenser

Profiler används för att sammanfatta utseendet hos en mängd relaterade sekvenser. De är intressanta därför att man har funnit att om man vill söka efter likheter så är det bättre att söka med en profil, som sammanfattar liknande gener/proteiner, än att söka enskilda sekvenser. Vanligen används profiler för att sammanfatta viktiga delar av sekvenser, men i den här programmeringsövningen förenklar vi uppgiften till att arbeta med hela sekvenser.

En profil för en uppsättning DNA- eller protein-sekvenser är en matris $M = (m_{i,j})$ där element $m_{i,j}$ är frekvensen av bokstaven i på position j . Om alla sekvenser man studerar börjar med "A", då ska vi ha att $m_{A,0} = 1$. Om hälften av sekvenserna har "A" i position 1, och den andra hälften har "C", då ska vi ha $m_{A,1} = m_{C,1} = 0.5$. Figur 1 har ett exempel på hur man går från sekvenser till profil och exemplets data finns i `molbio.hs`.

$$\begin{array}{|c|} \hline \text{ACATAA} \\ \text{AAGTCA} \\ \text{ACGTGC} \\ \text{AAGTTC} \\ \text{ACGTAA} \\ \hline \end{array} \longrightarrow C = \begin{array}{c} \begin{array}{ccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \begin{array}{c} A \\ C \\ G \\ T \end{array} & \begin{pmatrix} 5 & 2 & 1 & 0 & 2 & 3 \\ 0 & 3 & 0 & 0 & 1 & 2 \\ 0 & 0 & 4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 5 & 1 & 0 \end{pmatrix} \end{array} \longrightarrow M = \begin{array}{c} \begin{array}{ccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \begin{array}{c} A \\ C \\ G \\ T \end{array} & \begin{pmatrix} 1 & 0.4 & 0.2 & 0 & 0.4 & 0.6 \\ 0 & 0.6 & 0 & 0 & 0.2 & 0.4 \\ 0 & 0 & 0.8 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 1 & 0.2 & 0 \end{pmatrix} \end{array}
 \end{array}$$

Figur 1: Ett exempel på hur fem DNA-sekvenser av längd sex omvandlas till en profil. Matrisen C räknar hur många gånger varje bokstav används i varje position. Matrisen M skapas från C genom att dela varje element i C med antalet sekvenser.

Det finns flera sätt man kan mäta avståndet (eller skillnaden) mellan två profiler. Ett sätt är att räkna ut den totala elementvisa skillnaden. Låt $M = (m_{i,j})$ och $M' = (m'_{i,j})$ vara två profiler över n positioner.

```

nucleotides = "ACGT"
aminoacids = sort "ARNDCEQGHILKMFPSTWYV"

makeProfileMatrix :: [MolSeq] -> ???
makeProfileMatrix [] = error "Empty_sequence_list"
makeProfileMatrix sl = res
  where
    t = seqType (head sl)
    defaults =
      if (t == DNA) then
        zip nucleotides (replicate (length nucleotides) 0) -- Rad (ii)
      else
        zip aminoacids (replicate (length aminoacids) 0) -- Rad (iii)
    strs = map seqSequence sl -- Rad (iii)
    tmp1 = map (map (\x -> ((head x), (length x))) . group . sort)
            (transpose strs) -- Rad (iv)
    equalFst a b = (fst a) == (fst b)
    res = map sort (map (\l -> unionBy equalFst l defaults) tmp1)

```

Figur 2: Hjälpkod för att konstruera profilmatrix

Deras avstånd kan då skrivas

$$d(M, M') = \sum_{i \in \{A, C, G, T\}} \sum_{j=0}^{n-1} |m_{i,j} - m'_{i,j}|$$

Man summerar alltså över såväl alfabetet samt positionerna.

Om man skapar en profil för protein-sekvenser arbetar man med matriser som har 20 rader istället för 4, en rad för var och en av de tjugo aminosyrorerna (ARNDCEQGHILKMFPSTWYV).

Uppgifter

1. Skapa en datatyp `Profile` för att lagra profiler. Datatypen ska lagra information om den profil som lagras med hjälp av matrisen M (enligt beskrivningen ovan), det är en profil för DNA eller protein, hur många sekvenser profilen är byggd ifrån, och ett namn på profilen.
2. Skriv en funktion `molseqs2profile :: String -> [MolSeq] -> Profile` som returnerar en profil från de givna sekvenserna med den givna strängen som namn. Som hjälp för att skapa profil-matrisen har du koden i figur 2. Vid redovisning ska du kunna förklara exakt hur den fungerar, speciellt raderna (i)-(iv). Skriv gärna kommentarer direkt in i koden inför redovisningen, för så här kryptiskt ska det ju inte se ut!
3. Skriv en funktion `profileName :: Profile -> String` som returnerar en profils namn, och en funktion `profileFrequency :: Profile -> Int -> Char -> Double` som tar en profil p , en heltalsposition i , och ett tecken c , och returnerar den relativa frekvensen för tecken c på position i i profilen p (med andra ord, värdet på elementet $m_{c,i}$ i profilens matris M).
4. Skriv `profileDistance :: Profile -> Profile -> Double`. Avståndet mellan två profiler M och M' mäts med hjälp av funktionen $d(M, M')$ beskriven ovan.

4 Generell beräkning av avståndsmatriser

Du har nu definierat två relaterade datatyper, `MolSeq` och `Profile`. De är i grunden olika, men en operation som att beräkna avståndet mellan två objekt, till exempel, förenar dem även om de två imple-

mentationerna är olika. Eftersom vi har två skilda datatyper men med liknande funktioner, kan det vara praktiskt att skapa en typklass för att samla dem.

Vid studier av såväl molekylära sekvenser som profiler vill man ibland räkna ut alla parvisa avstånd och sammanfatta dessa i en *avståndsmatrix*. Eftersom en typklass kan samla generella metoder kan man skriva en sådan funktion i typklassen istället för att implementera den särskilt för de två datatyperna.

En avståndsmatrix kan representeras på många sätt, men i ett funktionellt språk är det ofta bra att ha en listrepresentation. Den representation du ska använda här är en lista av tripplar på formen (namn1, namn2, avstånd).

Uppgifter

1. Implementera typklassen `Evol` och låt `MolSeq` och `Profile` bli instanser av `Evol`. Alla instanser av `Evol` ska implementera en funktion `distance` som mäter avstånd mellan två `Evol`, och en funktion `name` som ger namnet på en `Evol`. Finns det någon mer funktion som man bör implementera i `Evol`?
2. Implementera funktionen `distanceMatrix` i `Evol` som tar en lista av någon typ som tillhör klassen `Evol`, och returnerar alla par av avstånd. Den här funktionen ska sedan automatiskt vara definierad för både listor av `MolSeq` och listor av `Profile`.

Som nämndes ska avståndsmatrisen som returneras representeras som en lista av tripplar på formen (namn1, namn2, avstånd). Denna ska komma i följande ordning: först kommer avstånden från det första elementet till alla andra. Sedan kommer avstånden från det andra elementet till alla andra utom det första (eftersom det redan angetts). Och så vidare. e.ex.: om vi har fyra `MolSeq`-objekt `A`, `B`, `C`, `D` och skickar in listan `[A, B, C, D]`, så ska `distanceMatrix` returnera listan

`[(A, A, ·), (A, B, ·), (A, C, ·), (A, D, ·), (B, B, ·), (B, C, ·), (B, D, ·), (C, C, ·), (C, D, ·), (D, D, ·)]`

(fast med samtliga “·” utbytta mot avståndet mellan respektive objekt).