Rosalind problems 3-11

Erik Sundberg

6/6/2019

Contents

3. Complementing a strand of DNA

In DNA strings, symbols 'A' and 'T' are complements of each other, as are 'C' and 'G'.

The reverse complement of a DNA string s is the string sc formed by reversing the symbols of s, then taking the complement of each symbol (e.g., the reverse complement of "GTCA" is "TGAC").

Given: A DNA string s of length at most 1000 bp.

Return: The reverse complement sc of s.

```
s <- "AAAACCCGGT"#aseq#

rc <- function(x) {
#Split a character string
ss <- unlist(strsplit(x, NULL))

#Reverse the split character string
ssrev <- rev(ss)

#Replace characters with complementary letters
for(i in 1:length(ssrev)) {
  if(ssrev[i] == "T") {
    ssrev[i] <- "A"
  } else if(ssrev[i] == "A") {
      ssrev[i] <- "T"
  } else if(ssrev[i] == "G") {
      ssrev[i] <- "C"
  } else if(ssrev[i] == "C") {
      ssrev[i] <- "G"
    }
```

```
}
#Paste the complementary string
sc <- paste(ssrev, collapse = "")
sc }
```

4. Rabbits and recurrence relations

Given: Positive integers n 40 and k 5.

Return: The total number of rabbit pairs that will be present after n months, if we begin with 1 pair and in each generation, every pair of reproduction-age rabbits produces a litter of k rabbit pairs (instead of only 1 pair).

```
rF <- c()
n = 32
k = 3
i = 0
for(i in 1:n) {
  if(i == 1) {
    rF[i] <- 1
  } else if(i == 2) {
    rF[i] <- 1
  } else rF[i] = rF[i-1] + k*rF[i-2]
}
rF[length(rF)]
```

5. Calculate percentage GC-content in a DNA sequence

Given: At most 10 DNA strings in FASTA format (of length at most 1 kbp each).

Return: The ID of the string having the highest GC-content, followed by the GC-content of that string. Rosalind allows for a default error of 0.001 in all decimal answers unless otherwise stated; please see the note on absolute error below.

```
library(seqinr)
#Function for calculating GC-content from text in fasta format
#x is the filename, eg 'rosalind_gc.txt'
gcc <- function(x) {
  rs <- read_file(x)
  rs2 <- unlist(strsplit(rs, ">"))
  rs2 <- rs2[-1]
  rs3 <- gsub("\n", "", rs2)
  i = 0
  v <- c()
  nv <- c()
  for(i in 1:length(rs3)) {
    s1 <- unlist(strsplit(rs3[i], NULL))
    sn <- paste(s1[1:13], collapse = "")
    s1 <- s1[14:length(s1)]
    s1 <- as.factor(s1)
    s1s <- summary(s1)
    gc1 <- unname(s1s["G"]) + unname(s1s["C"])
    gc1c <- gc1/length(s1) * 100
    v[i] <- gc1c
```

```
    nv[i] <- sn
  }
  names(v) <- nv
  print(noquote(names(v[which(v == max(v))])))
  return(max(v))
}


#Alternatively, using seqinr package
fasta <- read.fasta('rosalind_gc.txt')
gc_content <- apply(matrix(names(fasta)), 1, function(x){GC(fasta[[x]])})
most_gc <- which(gc_content==max(gc_content))

#Result
rbind(names(fasta)[most_gc], paste(signif(gc_content[most_gc], 7) * 100, "%", sep=""))
```

6. Counting points mutations

Given two strings s and t of equal length, the Hamming distance between s and t, denoted $d_H(s,t)$, is the number of corresponding symbols that differ in s and t.

Given: Two DNA strings s and t of equal length (not exceeding 1 kbp).

Return: The Hamming distance $d_H(s,t)$.

```
a <- c("")
a2 <- unlist(strsplit(a, "\n"))

b1 <- unlist(strsplit(a2[1], ""))
b2 <- unlist(strsplit(a2[2], ""))
length(which(b1 != b2))
```

7. Mendel's first law

Given: Three positive integers k, m, and n, representing a population containing k+m+n organisms: k individuals are homozygous dominant for a factor, m are heterozygous, and n are homozygous recessive.

Return: The probability that two randomly selected mating organisms will produce an individual possessing a dominant allele (and thus displaying the dominant phenotype). Assume that any two organisms can mate.

```
vec <- c("19 18 28")
vec <- as.double(unlist(strsplit(vec, " ")))

#Total nr of organisms
norg = sum(vec)

#Creat a matrix for probabilities
ofal <- c("AA","Aa","aa")
mmat <- matrix(nrow = 3, ncol = 3)
rownames(mmat) <- ofal
colnames(mmat) <- ofal
#for loop for the probability of mating combinations
for(i in 1:nrow(mmat)){
  for(j in 1:ncol(mmat)) {
```

```
    if(vec[i] == vec[j]) {
      mmat[i,j] = vec[i]/norg*(vec[j]-1)/(norg-1)
      } else if(vec[i] != vec[j]) {
      mmat[i,j] = (vec[i]/(norg))*(vec[j]/(norg-1))
    }
  }
}

dPr = sum(mmat[1,1:3])+sum(mmat[2:3,1])+0.75*sum(mmat[2,2])+sum(mmat[2,3])
dPr

#Alternative solutions:
gt.counts <- c(19,18,28)
gt.probs <- gt.counts / sum(gt.counts);
gt.subcounts <- t(matrix(gt.counts,3,3)) - diag(3);
gt.subprobs <- gt.subcounts / rowSums(gt.subcounts);

dominant.mat <- matrix(c(1,1,1,1,0.75,0.5,1,0.5,0),3,3);
sum(rowSums(gt.subprobs * dominant.mat) * gt.probs);

#or
domi <- rep(NA,1000000)
pop <- c(rep("AA",16), rep("Aa",17),rep("aa",15))
for(x in 1:length(domi)) {
  pair <- sample(pop,2)
  domi[x] <- sample(strsplit(pair[1],NULL)[[1]],1) == "A" | sample(strsplit(pair[2],NULL)[[1]],1) == "A
}
sum(domi)/length(domi)
```

8. Translating RNA into Protein

The 20 commonly occurring amino acids are abbreviated by using 20 letters from the English alphabet (all letters except for B, J, O, U, X, and Z). Protein strings are constructed from these 20 symbols. Henceforth, the term genetic string will incorporate protein strings along with DNA strings and RNA strings.

The RNA codon table dictates the details regarding the encoding of specific codons into the amino acid alphabet.

Given: An RNA string s corresponding to a strand of mRNA (of length at most 10 kbp).

Return: The protein string encoded by s.

```
library(readr)
#Given file w RNA string, in working directory
s <- readLines("rosalind_prot.txt")
s2 <- gsub("\n", "", s)
#RNA codon table
RC <- read_table('RNAcodonTable.txt', col_names = FALSE)
RCvec <- unlist(c(RC[,c(1,3,5,7)]))
RCvec <- unname(RCvec)
names(RCvec) <- unname(unlist(c(RC[,c(2,4,6,8)])))

ss <- c()
i = 1
j = 1
```

```
while(i < nchar(s) && j <= nchar(s)/3) {
  ss[j] <- substr(s, i,i+2)
  i <- i+3
  j <- j+1
}
#Amino-acid vector
pvec <- c()
k=1
for(k in 1:length(ss)) {
  if(ss[k] %in% RCvec) {
    pvec[k] <- names(which(RCvec == ss[k]))
  }
}
pvec <- pvec[-k]
paste(pvec, collapse = "")
```

Alternatively

```
## The translation table
codon_trans <- scan( what=character(), textConnection("
UUU F      CUU L      AUU I      GUU V
UUC F      CUC L      AUC I      GUC V
UUA L      CUA L      AUA I      GUA V
UUG L      CUG L      AUG M      GUG V
UCU S      CCU P      ACU T      GCU A
UCC S      CCC P      ACC T      GCC A
UCA S      CCA P      ACA T      GCA A
UCG S      CCG P      ACG T      GCG A
UAU Y      CAU H      AAU N      GAU D
UAC Y      CAC H      AAC N      GAC D
UAA Stop   CAA Q      AAA K      GAA E
UAG Stop   CAG Q      AAG K      GAG E
UGU C      CGU R      AGU S      GGU G
UGC C      CGC R      AGC S      GGC G
UGA Stop   CGA R      AGA R      GGA G
UGG W      CGG R      AGG R      GGG G
"
))

lc <- length( codon_trans )
codon_trans_table <- codon_trans[ seq(1, lc, by=2) ]
names( codon_trans_table ) <- codon_trans[ seq(2, lc, by=2) ]

## Handy replace function
kReplace <- function( vec, orig, out=names(orig) ) {
  tmp <- out[ match( vec, orig ) ]
  tmp[ is.na(tmp) ] <- vec[ is.na(tmp) ]
  tmp
}

## Slice function
strslice <- function(x, n) {
  x <- as.data.frame( stringsAsFactors=FALSE,
```

```
                    matrix( unlist( strsplit( x, "" ) ), ncol=n, byrow=T )
  )

  do.call( paste0, x )

}

dat <- scan( "rosalind_prot.txt", what=character(), quiet=TRUE )

codons <- strslice(dat, 3)
cat(paste(collapse="",
      kReplace( codons, codon_trans_table )
))
```

9. Finding a motif in DNA

Given: Two DNA strings s and t (each of length at most 1 kbp).

Return: All locations of t as a substring of s.

```
#Read given DNA string and motif, from file in wd
#s <- readLines("rosalind_subs.txt")
#s1 <- s[1]
#s2 <- s[2]
#unlist(gregexpr(pattern = s2, text = s1, perl=T))

#or...
s2 <- unlist(strsplit("CTTCGTTTCCTGCTCGTTTCTCGTTTCTCAATCGTTTCATCGTGTCGTTTCTGTCGTTTCAGGTCGTTTCTCGTTTCTTC
t <- unlist(strsplit("TCGTTTCTC", ""))
i = 1
ix = c()
nix = c()
for( i in 1:length(s2) ) {
  end <- length(t) - 1 + i
  if(paste(t, collapse = "") == paste(s2[i:end], collapse = "")) {
    ix <- c(ix, i)
  } else nix <- c(nix, i)
}

ix
```

10. Consensus and Profile (most likely common ancestor)

Say that we have a collection of DNA strings, all having the same length n. Their profile matrix is a 4×n matrix P in which $P_{1,j}$ represents the number of times that 'A' occurs in the jth position of one of the strings, $P_{2,j}$ represents the number of times that C occurs in the j th position, and so on.

A consensus string c is a string of length n formed from our collection by taking the most common symbol at each position; the jth symbol of c therefore corresponds to the symbol having the maximum value in the j-th column of the profile matrix. Of course, there may be more than one most common symbol, leading to multiple possible consensus strings.

Given: A collection of at most 10 DNA strings of equal length (at most 1 kbp) in FASTA format.

Return: A consensus string and profile matrix for the collection. (If several possible consensus strings exist, then you may return any one of them.)

```
DNAstr <- readLines("/Users/eriksundberg/Downloads/rosalind_cons.txt")
DNApst <- paste(DNAstr, collapse = "")
DNAsub <- gsub(">Rosalind_[0-9][0-9][0-9][0-9]", " ", DNApst)
DNAspl <- unlist(strsplit(DNAsub, " "))
DNAspl2 <- DNAspl[2:11]
DNAmat <- sapply(DNAspl2, strsplit, "", USE.NAMES = FALSE)
DNAmatr <- unname(t(as.data.frame(DNAmat[1:10])))

Prof <- matrix(nrow = 4, ncol = ncol(DNAmatr))
rownames(Prof) <- levels(factor(DNAmatr))

for(i in 1:ncol(DNAmatr)) {
  Prof[,i] <- summary(factor(DNAmatr[,i], levels = c("A","C","G","T")))
}

Cons <- c()
for(i in 1:ncol(DNAmatr)) {
  Cons <- c(Cons, names(which.max(Prof[,i])))
}

paste(Cons, collapse = "")
Prof

A <- c(paste("A:"), paste(Prof["A",]))
C <- c(paste("C:"), paste(Prof["C",]))
G <- c(paste("G:"), paste(Prof["G",]))
TT <- c(paste("T:"), paste(Prof["T",]))
write(paste(Cons, collapse = ""), file = "DNAmat.txt", ncolumns = length(A), append = FALSE)
write(paste(A), file = "DNAmat.txt", ncolumns = length(A), append = TRUE)
write(paste(C), file = "DNAmat.txt", ncolumns = length(C), append = TRUE)
write(paste(G), file = "DNAmat.txt", ncolumns = length(G), append = TRUE)
write(paste(TT), file = "DNAmat.txt", ncolumns = length(TT), append = TRUE)
```

Alternatively

```
alphabet = c("A","C","G","T")
mat = read.table("rosalind_cons.txt")

smat <- paste(mat$V1, collapse = "")
smat <- strsplit(smat, ">")
smat <- unlist(smat)
smat <- smat[-1]
smat <- gsub("Rosalind_[0-9]*","",smat)
smat <- cbind(smat)

mat2 <- t(apply(smat, 1, function(x){ strsplit(x, "")[[1]] }))

res=apply(mat2,2,function(x){  a=sum(x=="A"); c=sum(x=="C"); g=sum(x=="G"); t=sum(x=="T"); return(c(a,c
rownames(res)=c("A:","C:","G:","T:")
consensus = apply(res,2,function(x){  alphabet[which.max(x)] })
consensus = paste(consensus,collapse="")
```

11. Mortal Fibonacci rabbits

Recall the definition of the Fibonacci numbers from "Rabbits and Recurrence Relations", which followed the recurrence relation $F_n=F_{n-1}+F_{n-2}$ and assumed that each pair of rabbits reaches maturity in one month and produces a single pair of offspring (one male, one female) each subsequent month.

Our aim is to somehow modify this recurrence relation to achieve a dynamic programming solution in the case that all rabbits die out after a fixed number of months. See Figure 4 for a depiction of a rabbit tree in which rabbits live for three months (meaning that they reproduce only twice before dying).

Given: Positive integers n 100 and m 20.

Return: The total number of pairs of rabbits that will remain after the n-th month if all rabbits live for m months.

```
library(gmp)
rF <- c(1,1) #Vector of rabbits alive each month
rF <- as.bigz(rF)
n <- 86 #Number of i total
m <- 17 #Number of i alive
k = 1
i = 1
for(i in 3:n) {
    if(i > 2 && m >= i) { #3 to 4, two elem (m=4)
    rF[i] = sum(as.bigz(rF[i-1]), as.bigz(rF[i-2]))
  } else if(i == m+1) { #5
    rF[i] = sum(as.bigz(rF[i-1]),as.bigz(rF[i-2]),-1)
  } else if(i >= m+2 && i <m+4) { #6 to 7
    rF[i] = sum(as.bigz(rF[i-1]),as.bigz(rF[i-2]),-1)
    } else rF[i] = sum(as.bigz(rF[i-1]),as.bigz(rF[i-2]),as.bigz(-rF[i-m-1]))
  }
as.bigz(rF[length(rF)])
```

Alternatively

```
#Correctly:
library('gmp')

rabbitpairs <- function(n,m) {

adult   <- as.bigz(vector(length = n))
adult[1:2] <- c(as.bigz("0"),as.bigz("1"))
offsp   <- as.bigz(vector(length = n))
offsp[1:2] <- c(as.bigz("1"),as.bigz("0"))

seq <- numeric(length = n)

for(i in 3:n){

    adult[i] <- adult[i-1] + offsp[i-1]

    if(i > m ){
        adult[i] <- adult[i] - offsp[i-m]
    }

    offsp[i] <- adult[i-1]
    }
```

```
seq <- adult + offsp
seq[n]
}
rabbitpairs(86,17)
```

Problem descriptions sourced from rosalind.info