Rosalind problems 12-22

Erik Sundberg

6/12/2019

Contents

---

12. DNA strings Overlap graph

A graph whose nodes have all been labeled can be represented by an adjacency list, in which each row of the list contains the two node labels corresponding to a unique edge.

A directed graph (or digraph) is a graph containing directed edges, each of which has an orientation. That is, a directed edge is represented by an arrow instead of a line segment; the starting and ending nodes of an edge form its tail and head, respectively. The directed edge with tail v and head w is represented by (v,w) (but not by (w,v)). A directed loop is a directed edge of the form (v,v).

For a collection of strings and a positive integer k, the overlap graph for the strings is a directed graph O_k in which each string is represented by a node, and string s is connected to string t with a directed edge when there is a length k suffix of s that matches a length k prefix of t, as long as s t; we demand s t to prevent directed loops in the overlap graph (although directed cycles may be present).

Given: A collection of DNA strings in FASTA format having total length at most 10 kbp.

Return: The adjacency list corresponding to O_3. You may return edges in any order.

Solution:

```r
options(tinytex.verbose = TRUE)
#Scan the given text (fasta format)
OGd <- scan("rosalind_grph.txt", what = character())
#Complete sequences separated by \n in given text
OGd <- gsub(">", "", OGd)
```

```r
ixn <- grep("Rosa",OGd) #Indices for seq names
OGn <- OGd[ixn] #Seq names vector

OGd[ixn] <- "_" #Replace seq names
dna_list <- unlist(strsplit(paste(OGd, collapse = ""), "_"))
dna_list <- dna_list[-1]

#Name the sequences
names(dna_list) <- OGn

#Construct the O3 oadjacency list
ol <- NULL
for (i in 1:length(dna_list)) {
    for (j in 1:length(dna_list)) {
        if(substr(dna_list[i], nchar(dna_list[i])-2, nchar(dna_list[i])) == substr(dna_list[j],1,3) && 
           ol <- rbind(ol, cbind(names(dna_list[i]), names(dna_list[j])))
        }
    }
}

#Write the the answer to output file.
write.table(ol, "output.txt", quote = FALSE, row.names = FALSE, col.names = FALSE, sep = " ")
```

Alternatively

```r
library(seqinr)
library(stringr)
data <- read.fasta(file = "rosalind_grph.txt", as.string = TRUE)
data_name <-names(data)
for(i in 1:(length(data))) {
  data[i]= unlist(data[i],use.names=FALSE)
}

result <- character()
for(i in 1:length(data)) {
  data[i][-1:-3]
  for(j in 1:length(data)) {
    if(i!=j) {
      if(str_sub(data[i],-3,-1)==str_sub(data[j],1,3)) {
      result <- rbind(result, c(data_name[i],data_name[j]))
      }
    }
  }
}
write.table(result, file="mymatrix.txt", row.names=FALSE, col.names=FALSE,quote=FALSE)
```

---

13. Calculating expected offspring

Given: Six nonnegative integers, each of which does not exceed 20,000. The integers correspond to the number of couples in a population possessing each genotype pairing for a given factor. In order, the six given integers represent the number of couples having the following genotypes:

1. AA-AA
2. AA-Aa
3. AA-aa
4. Aa-Aa
5. Aa-aa
6. aa-aa

Return: The expected number of offspring displaying the dominant phenotype in the next generation, under the assumption that every couple has exactly two offspring.

Solution:

```
#Numbers of each genotype pairing
np <- scan("rosalind_iev.txt")

#Vector of probabilities for offsp w dom phenotype
vP <- c(1, 1, 1, 0.75, 0.5, 0)

#Multiply pairs w probabilities, each pair gets 2 offsp
os <- sum(2*np*vP)

#Total nr of offsp w dom phenotype, write to file
write.table(os, file = "os_output.txt",quote = FALSE, row.names = FALSE, col.names = FALSE)
```

---

14. Finding a shared motif

A common substring of a collection of strings is a substring of every member of the collection. We say that a common substring is a longest common substring if there does not exist a longer common substring. For example, "CG" is a common substring of "ACGTACGT" and "AACCGTATA", but it is not as long as possible; in this case, "CGTA" is a longest common substring of "ACGTACGT" and "AACCGTATA".

Note that the longest common substring is not necessarily unique; for a simple example, "AA" and "CC" are both longest common substrings of "AACC" and "CCAA".

Given: A collection of k (k 100) DNA strings of length at most 1 kbp each in FASTA format.

Return: A longest common substring of the collection. (If multiple solutions exist, you may return any single solution.)

Solution:

```
library(seqinr)
#Read all fasta sequences
fasL <- read.fasta("rosalind_lcsm.txt")
#Convert to a vector of seq-strings
allS <- toupper(unname(sapply(fasL, paste, collapse = "")))
#Order so that the shortest seq is nr 1
allS <- allS[order(nchar(allS), allS)]

#Check if a motif in one seq exists in all seqs
mot <- ""  #motif-string
j = 1  #start of substring
i = 0  #increased length of substring
```

```r
while(j <= nchar(allS[1])-nchar(mot))
    {
  for(i in nchar(mot):nchar(allS[1]))
      {
      if(names(summary(grepl(substr(allS[1], j, j+i), allS)))[2] != "FALSE" & nchar(substr(allS[1], j,
        {
    mot <- substr(allS[1], j, j+i)  #longest motif
      } else break
    }
  j <- j+1
}

print(mot, quote = FALSE)
writeChar(mot, "mot_output.txt")
```

---

15. Independent alleles

Two events A and B are independent if Pr(A and B) is equal to Pr(A)×Pr(B). In other words, the events do not influence each other, so that we may simply calculate each of the individual probabilities separately and then multiply.

More generally, random variables X and Y are independent if whenever A and B are respective events for X and Y, A and B are independent (i.e., Pr(A and B)=Pr(A)×Pr(B)).

Given: Two positive integers k (k 7) and N (N 2k). In this problem, we begin with Tom, who in the 0th generation has genotype Aa Bb. Tom has two children in the 1st generation, each of whom has two children, and so on. Each organism always mates with an organism having genotype Aa Bb.

Return: The probability that at least N Aa Bb organisms will belong to the k-th generation of Tom's family tree (don't count the Aa Bb mates at each level). Assume that Mendel's second law holds for the factors.

Solution:

```r
#Aa and Bb alleles are independent

k = 7 #Final generation
N = 37 #Least number of Aa Bb in gen k
P = 2^k #Gen k population (nr of 'trials')

#Probability that a parent mating with Aa Bb has an offspring with Aa Bb is always 0.25 (0.5*0.5)

#Perform binomial distribution to get Pr that at least N offspring have the Aa Bb genotype
#quantiles = 2^k-N (91), nr of 'trials' = 2^k (128)
prob <- pbinom(P-N, P, 0.75)
#or sum(dbinom(N:P, P, 0.25))
print(signif(prob, 4))
```

---

16. Finding a protein motif

To allow for the presence of its varying forms, a protein motif is represented by a shorthand as follows: [XY] means “either X or Y” and {X} means “any amino acid except X.” For example, the N-glycosylation motif is written as N{P}[ST]{P}.

You can see the complete description and features of a particular protein by its access ID “uniprot_id” in the UniProt database, by inserting the ID number into http://www.uniprot.org/uniprot/uniprot_id.

Alternatively, you can obtain a protein sequence in FASTA format by following http://www.uniprot.org/uniprot/uniprot_id.fasta.

For example, the data for protein B5ZC00 can be found at http://www.uniprot.org/uniprot/B5ZC00.

Given: At most 15 UniProt Protein Database access IDs.

Return: For each protein possessing the N-glycosylation motif, output its given access ID followed by a list of locations in the protein string where the motif can be found.

Solution:

```r
library(seqinr)

write("", "output.txt", append = FALSE)
nm <- readLines("rosalind_mprt.txt") #UniProt IDs

tmp <- read.fasta("testPmotif.txt") #Protein sequences
names(tmp) <- nm
pmt <- toupper(sapply(tmp, paste, collapse=""))

vecn <- c() #names vector
veci <- c() #indices vector
k = 0 #shortened length of vectors
#Find all (N-glycosylation) motif's starting w N, followed by not P, then S or T, then not P.
motf <- gregexpr("N(?=[^P][ST][^P])", pmt[1:length(tmp)], perl = TRUE)

for(i in 1:length(motf)){
  if(motf[[i]][1] != -1){
    vecn[i-k] <- paste(names(tmp[i]))
    veci[i-k] <- paste(as.numeric(motf[[i]]), collapse = " ")
    write(vecn[i-k], "output.txt", append = TRUE)
    write( veci[i-k], "output.txt", append = TRUE, sep = " ")
  } else k = k+1
}
```

---

17. Inferring mRNA from protein

For positive integers a and n, a modulo n (written a mod n in shorthand) is the remainder when a is divided by n. For example, 29 mod 11 = 7 because 29 = 11 × 2 + 7.

Modular arithmetic is the study of addition, subtraction, multiplication, and division with respect to the modulo operation. We say that a and b are congruent modulo n if a mod n = b mod n; in this case, we use the notation a bmodn.

Two useful facts in modular arithmetic are that if a bmodn and c dmodn, then a + c   b + d mod n and a × c   b × d mod n. To check your understanding of these rules, you may wish to verify these relationships for a = 29, b = 73, c = 10, d = 32, and n = 11.

As you will see in this exercise, some Rosalind problems will ask for a (very large) integer solution modulo a smaller number to avoid the computational pitfalls that arise with storing such large numbers.

Given: A protein string of length at most 1000 aa.

Return: The total number of different RNA strings from which the protein could have been translated, modulo 1,000,000. (Don't neglect the importance of the stop codon in protein translation.)

Solution:

```r
#Construct an 'amino-acid'-vector w elements corresponding to each codon
cdns <- seqinr::words(alphabet = s2c("tcag"))
cdns <- as.list(cdns)
cdns <- sapply(cdns, strsplit, "")
cdnsAA <- sapply(cdns, translate)

#Read the amino-acid seq
AAseq <- readLines("rosalind_mrna.txt")
AAseq <- unlist(strsplit(AAseq, ""))
AAseq <- c(AAseq, "*") #add the stop-codon
n = 10^6
b = 1
for(i in 1:length(AAseq)) {
a = length(which(cdnsAA == AAseq[i]))
b = (b*a) %% n
}
b
```

---

18. Open reading frames

Either strand of a DNA double helix can serve as the coding strand for RNA transcription. Hence, a given DNA string implies six total reading frames, or ways in which the same region of DNA can be translated into amino acids: three reading frames result from reading the string itself, whereas three more result from reading its reverse complement.

An open reading frame (ORF) is one which starts from the start codon and ends by stop codon, without any other stop codons in between. Thus, a candidate protein string is derived by translating an open reading frame into amino acids until a stop codon is reached.

Given: A DNA string s of length at most 1 kbp in FASTA format.

Return: Every distinct candidate protein string that can be translated from ORFs of s. Strings can be returned in any order.

Solution:

```r
library(seqinr)
#Given sequence as character vector
aseq <- unlist(lapply(read.fasta("rosalind_orf.txt"), toupper))
#Complementary sequence (made w rc(x) from chunk above)
raseq <- unlist(strsplit(rc(toupper(aseq)), ""))

#Translate all possible reading frames, store as vectors of strings:
fRF <- c() #Translated forward reading frames
rRF <- c() #Translated reverse reading frames
```

```
for(i in 1:3) {
 fRF[i] <- paste(translate(aseq[i:length(aseq)]), collapse = "")
 rRF[i] <- paste(translate(raseq[i:length(raseq)]), collapse = "")
}

#Extract all strings containing "M", ending w a stop-codon ("*"), store them in a vector
prtn <- c()
for(i in 1:length(fRF)) {
tmp = NULL
tmp <- unlist(strsplit(fRF[i], "\\*")) #split at "*"
tmp <- tmp[-length(tmp)] #remove last string
tmp <- tmp[grep("M", tmp)] #extract strings w "M"
tmp2 = NULL
tmp2 <- unlist(strsplit(rRF[i], "\\*"))
tmp2 <- tmp2[-length(tmp2)]
tmp2 <- tmp2[grep("M", tmp2)]
prtn <- c(prtn, tmp, tmp2)
}

#From each string, extract all possible substrings starting w an "M", store them in a vector
orf <- c()
for(i in 1:length(prtn)) {
  ix <- c()  #indices for "M"
  ix <- unlist(gregexpr("M", prtn[i]))
  for(j in 1:length(ix)) {
    orf <- c(orf, substr(prtn[i], ix[j], nchar(prtn[i])))
  }
}
print(unique(orf), quote = FALSE) #Print w/o duplicates
writeLines(unique(orf), "output.txt")
```

---

19. Enumerating gene orders

A permutation of length n is an ordering of the positive integers {1,2,…,n}. For example, =(5,3,2,1,4) is a permutation of length 5.

Given: A positive integer n 7.

Return: The total number of permutations of length n, followed by a list of all such permutations (in any order).

Solution:

```
n = 5 #given integer

perm <- function(x) { #permutation function

dig <- seq(1, x) #integer sequence

#Make vector w all 1st integers
digR = rep(seq(1, x), each = factorial(x-1))
#Make empty permutation matrix
pm <- matrix(ncol = x, nrow = length(digR))
#Fill 1st column w all 1st ints
pm[,1] <- digR

b <- c() #times to repeat each seq of ints... (dig[-ix])
for(i in 1:(length(dig)-1)) {
  b = c(b, factorial(x-(i+1)) )
}

d = 1
while(d < x) {
  tmp <- c()
  k = 1
  for(i in 1:(factorial(x)/factorial(x-d) ) ) {
    ix <- c() #indices for ints not in preceeding seq
    ix <- which(dig %in% pm[k,1:d])
    tmp <- c(tmp, rep(dig[-ix], each = b[d]) )

    k = k+b[d]*dig[length(dig)-d]
    if(i == (factorial(x)/factorial(x-d) ) ) {

      pm[,d+1] <- tmp

    }
  }
d = d+1
}

print(dim(pm)[1])
return(pm)
write.table(dim(pm)[1], file ="output.txt", quote = FALSE, row.names = FALSE, col.names = FALSE)
write.table(pm, file ="output.txt", quote = FALSE, row.names = FALSE, col.names = FALSE, append = TRUE)
}
```

Alternatively

---

20. Calculating protein mass

In a weighted alphabet, every symbol is assigned a positive real number called a weight. A string formed from a weighted alphabet is called a weighted string, and its weight is equal to the sum of the weights of its symbols.

The standard weight assigned to each member of the 20-symbol amino acid alphabet is the monoisotopic mass of the corresponding amino acid.

Given: A protein string P of length at most 1000 aa.

Return: The total weight of P. Consult the monoisotopic mass table.

Solution:

```r
#Given protein
amins <- unlist(strsplit(readLines("rosalind_prtm.txt"),""))
#Weighted alphabet
mimt <- read.table("Monoisotopic mass table.txt")

prtm = 0
for(i in 1:length(amins)) {
  ix <- which(mimt[,1] == amins[i])
  prtm = prtm + mimt[ix, 2]
}
print(format(prtm, nsmall = 3), quote = FALSE)
```

Reverse-complementary-sequence function

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

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 }
```

---

21. Locating restriction sites

A DNA string is a reverse palindrome if it is equal to its reverse complement. For instance, GCATGC is a reverse palindrome because its reverse complement is GCATGC. See Figure 2.

Given: A DNA string of length at most 1 kbp in FASTA format.

Return: The position and length of every reverse palindrome in the string having length between 4 and 12. You may return these pairs in any order.

Solution:

```r
dna = toupper(unlist(read.fasta('rosalind_revp.txt')))

locz <- c()
lenz <- c()
for (length in 3:12) {
    for (index in 1:(length(dna)-length+1)) {
        if (DNAString(paste(dna[index:(index+length)], collapse ="")) == reverseComplement(DNAString(pa

            locz <- c(locz, index)
            lenz <- c(lenz, length+1)

        }
    }
}

revp <- matrix(ncol = 2, nrow = length(lenz))
revp[,1] <- locz
revp[,2] <- lenz
revp <- revp[order(revp[,1]),]

write.table(revp, "output.txt", row.names = FALSE, col.names = FALSE)
```

Alternatively

```r
DnaComplements <- list(T='A', A='T', C='G', G='C')

complementDnaItem <- function(token) {
  DnaComplements[[token]]
}

complementDna <- function(str, reverse=FALSE) {
  complement <- sapply(unlist(strsplit(str, '')), complementDnaItem)
  if (reverse) complement <- rev(complement)
  paste(complement, collapse='')
}

isDnaReversePalindrome <- function(str) {
  str == complementDna(str, rev=TRUE)
}

findAllRevPalindromes <- function(str, lo=4, hi=8) {
  str_len <- nchar(str)
  for (i in 1:(str_len-4)) {
    for (j in lo:(min(str_len-i+1,hi))) {
      search_str <- substr(str, i, i+j-1)
      if (isDnaReversePalindrome(search_str)) cat(i,j,"\n")
    }
  }
}

data <- scan('rosalind_revp.txt', character(), quiet=TRUE)
findAllRevPalindromes(data)
```

Alternatively

```r
library(stringi)

seq<-"TCAATGCATGCGGGTCTATATGCAT"
comp_seq<-chartr("ATCG","TAGC", seq) #Translate characters

 for (i in 1:nchar(seq)) {
    for (j in c(3,5,7,9,11)) {
        string1<-substring(seq, i, j+i)
        string2<-substring(comp_seq, i, j+i)

        if (nchar(string1)<=j) {
            break
        }

        if(string1==stri_reverse(string2)) { #rev string function!
            cat(i, " ", j+1, "\n", sep="")
        }
    }
}
```

---

22. RNA Splicing

After identifying the exons and introns of an RNA string, we only need to delete the introns and concatenate the exons to form a new string ready for translation.

Given: A DNA string s (of length at most 1 kbp) and a collection of substrings of s acting as introns. All strings are given in FASTA format.

Return: A protein string resulting from transcribing and translating the exons of s. (Note: Only one solution will exist for the dataset provided.)

Solution:

```r
exin <- read.fasta("rosalind_splc.txt", as.string = TRUE, forceDNAtolower = TRUE)

exin <- sapply(exin, as.vector)
dna <- exin[1] #DNA/pre-mRNA to be spliced
intr <- exin[-1] #introns to splice away

spl <- dna
for(i in 1:length(intr)) {
  #remove patterns equal to introns
  spl <- base::gsub(intr[i], "", spl)
}

splc <- unlist(strsplit(spl, ""))
amin <- seqinr::translate(splc) #translate into aa-sequence
pept <- paste(amin[-length(amin)], collapse = "")

writeLines(pept, "output.txt")
```

Alternatively

```r
library(Biostrings)
x <- readDNAStringSet("rosalind_splc.txt")
dna <- as.character(x[[1]])
#
i<-2
while (i<=length(x))
{
  dna <- sub(x[[i]],"",dna)
  i <- i+1
}
p <- translate(DNAString(dna))
```

Problem descriptions sourced from rosalind.info