SDGB 7844 HW 1: R Syntax and Control Structures

Name: Jiayin Hu     Class Time: Thu. 1:15-3:15PM

2018-09-26

## Question 1

The vectors state.name, state.area, and state.region are pre-loaded in R and contain US state names, area (in square miles), and region respectively.

First, have a look at the content of these 3 vectors.

state.name

## [1] "Alabama" "Alaska" "Arizona" "Arkansas"   
## [5] "California" "Colorado" "Connecticut" "Delaware"   
## [9] "Florida" "Georgia" "Hawaii" "Idaho"   
## [13] "Illinois" "Indiana" "Iowa" "Kansas"   
## [17] "Kentucky" "Louisiana" "Maine" "Maryland"   
## [21] "Massachusetts" "Michigan" "Minnesota" "Mississippi"   
## [25] "Missouri" "Montana" "Nebraska" "Nevada"   
## [29] "New Hampshire" "New Jersey" "New Mexico" "New York"   
## [33] "North Carolina" "North Dakota" "Ohio" "Oklahoma"   
## [37] "Oregon" "Pennsylvania" "Rhode Island" "South Carolina"  
## [41] "South Dakota" "Tennessee" "Texas" "Utah"   
## [45] "Vermont" "Virginia" "Washington" "West Virginia"   
## [49] "Wisconsin" "Wyoming"

state.area

## [1] 51609 589757 113909 53104 158693 104247 5009 2057 58560 58876  
## [11] 6450 83557 56400 36291 56290 82264 40395 48523 33215 10577  
## [21] 8257 58216 84068 47716 69686 147138 77227 110540 9304 7836  
## [31] 121666 49576 52586 70665 41222 69919 96981 45333 1214 31055  
## [41] 77047 42244 267339 84916 9609 40815 68192 24181 56154 97914

state.region

## [1] South West West South West   
## [6] West Northeast South South South   
## [11] West West North Central North Central North Central  
## [16] North Central South South Northeast South   
## [21] Northeast North Central North Central South North Central  
## [26] West North Central West Northeast Northeast   
## [31] West Northeast South North Central North Central  
## [36] South West Northeast Northeast South   
## [41] North Central South South West Northeast   
## [46] South West South North Central West   
## Levels: Northeast South North Central West

1. Identify the data type for state.name, state.area, and state.region.

typeof(state.name)

## [1] "character"

typeof(state.area)

## [1] "double"

typeof(state.region)

## [1] "integer"

The data type of these three vectors are character, double and integer respectively.

1. What is the longest state name (including spaces)? How long is it?

#Using if statement  
state.name.len = 0 # store the longest length  
y <- c() #store the order number of the state with longest name  
  
for (i in 1:length(state.name)){  
 x <- nchar(state.name[i]) #the length of each state  
 if (state.name.len < x){   
 state.name.len = x  
 y <- i  
 }  
 else{  
 if (state.name.len == x) y <- c(y, i)  
 }  
}  
  
print(state.name[y]) #output the state name

## [1] "North Carolina" "South Carolina"

state.name.len

## [1] 14

The state “North Carolina” and “South Carolina” have the longest length. They both contains 14 bytes.

1. Compute the average area of the states which contain the word “New” at the start of the state name. Use the function substr().

state.area.total <- 0 #initial sum of area  
new.state <- c() #store the state beginning with "New"  
  
for (i in 1:length(state.name)){  
 if(substr(state.name[i], start = 1, stop = 3) == "New"){  
 new.state <- c(new.state, state.name[i])  
 state.area.total <- state.area.total + state.area[i]  
 }  
}  
  
print(new.state)

## [1] "New Hampshire" "New Jersey" "New Mexico" "New York"

state.area.average <- state.area.total/length(new.state)  
print(paste("The average area of the states which contain the word “New” at the start of the state name is", state.area.average))

## [1] "The average area of the states which contain the word “New” at the start of the state name is 47095.5"

1. Use the function table() to determine how many states are in each region. Use the function kable() to include the table in your solutions. (Notes: you will need the R package *knitr* to be able to use kable(). See the RMarkdown example in the Assignments folder on Blackboard for an example.)

library(knitr)  
state.table <- table(state.region)  
kable(state.table, caption = "State Region Table", align = "c" )

State Region Table

|  |  |
| --- | --- |
| state.region | Freq |
| Northeast | 9 |
| South | 16 |
| North Central | 12 |
| West | 13 |

## Quesetion 2

Perfect numbers are those where the sum of the proper divisors (i.e., divisors other than the number itself) add up to the number. For example, 6 is a perfect number because its divisors, 1, 2, and 3, when summed, equal 6.

1. The following code was written to find the first 2 perfect numbers: 6 and 28; however, there are some errors in the code and the programmer forgot to add comments for readability. Debug and add comments to the following:

num.perfect <- 2 #to find 2 perfect numbers  
count <- 0 #the number of the perfect number we've found  
iter <- 2 #the number used to check whether it is a perfect number or not. The initial value is 2.   
  
while(count < num.perfect){ # modify <= to <. When there's less than 2 numbers, keep going  
   
 divisor <- 1 #1 is a divisorof all numbers.  
   
#Use for circulation to find all the divisors except the number itself.  
 for(i in 2:(iter-1)){  
   
 if(iter%%i==0) divisor <- c(divisor, i)  
   
 } # end for loop  
  
#Determine whether the sum of the dicisors equals to the number.  
#If true, then output the number is a perfect number and add 1 to count.  
 if(sum(divisor)==iter){ # modify = to ==  
   
 print(paste(iter, " is a perfect number", sep=""))  
 count <- count + 1   
   
 } # end if  
   
 iter <- iter + 1 #check the next number in the next circulation  
   
} # end while loop

## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"

1. Use the function date() at the start and at the end of your amended code. Then compute how long the program approximately takes to run (you can do this sub- traction by hand). Find the run time when you set num.perfect to 1, 2, 3, and 4. Create a table of your results (Note: use the first table format in the RMarkdown Example file in the Assignments folder on Blackboard) . What are the first four perfect numbers?

for (t in 1:4){  
 print(date())  
 num.perfect <- t #to find t perfect numbers  
 count <- 0 #the number of the perfect number we've found  
 iter <- 2 #the number used to check whether it is a perfect number or not. The initial value is 2.   
   
 while(count < num.perfect){ #When there's less than t numbers, keep going  
   
 divisor <- 1 #1 is a divisorof all numbers.  
   
 #Use for circulation to find all the divisors except the number itself.  
 for(i in 2:(iter-1)){  
   
 if(iter%%i==0) divisor <- c(divisor, i)  
   
 } # end for loop  
   
 #Determine whether the sum of the dicisors equals to the number.  
 #If true, then output the number is a perfect number and add 1 to count.  
 if(sum(divisor)==iter){   
   
 print(paste(iter, " is a perfect number", sep=""))  
 count <- count + 1   
   
 } # end if  
   
 iter <- iter + 1 #check the next number in the next circulation  
   
 } # end while loop  
 print(date())  
}

## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "6 is a perfect number"  
## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"  
## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"  
## [1] "496 is a perfect number"  
## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "Wed Sep 26 12:58:54 2018"  
## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"  
## [1] "496 is a perfect number"  
## [1] "8128 is a perfect number"  
## [1] "Wed Sep 26 12:59:01 2018"

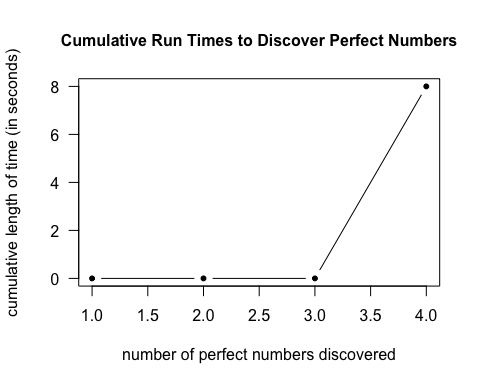
So the first 4 perfect number is 6, 28, 496, 8128. The rough run time is shown as follow.

Run Time Table

|  |  |
| --- | --- |
| No.of.Perfect.Numbers | Run.Time |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 8 |

1. Let x <- 1:4 and define y to be the vector of run times. Plot y vs x using the code below. Is the relationship between the discovery of perfect numbers and run times on your computer linear? Justify your answer.

x <- 1:4  
y <- c(0, 0, 0, 8)  
plot(x, y, pch=20, type="b", xlab="number of perfect numbers discovered",  
 ylab="cumulative length of time (in seconds)",   
 main="Cumulative Run Times to Discover Perfect Numbers", las=TRUE, cex.main = 1.0)



The relationship between the discovery of perfect numbers and run times on your computer is not linear. Run time increases significantly when the number we aim to find is larger than 3, because we have to run more than 8000 loops.

Here the time is roughly calculated by hands. In another way, we use Sys.time() to record the time and repeat (b) and (c). Meanwhile, this function can subtract directly to get the run time.

y <- c()  
for (t in 1:4){  
 start <- Sys.time()  
 num.perfect <- t #to find n perfect number  
 count <- 0 #the number of the perfect number we've found  
 iter <- 2 #the number used to check whether it is a perfect number or not. The initial value is 2.   
   
 while(count < num.perfect){ #When there's less than n numbers, keep going  
   
 divisor <- 1 #1 is a divisorof all numbers.  
   
 #Use for circulation to find all the divisors except the number itself.  
 for(i in 2:(iter-1)){  
   
 if(iter%%i==0) divisor <- c(divisor, i)  
   
 } # end for loop  
   
 #Determine whether the sum of the dicisors equals to the number.  
 #If true, then output the number is a perfect number and add 1 to count.  
 if(sum(divisor)==iter){ # ==  
   
 print(paste(iter, " is a perfect number", sep=""))  
 count <- count + 1   
   
 } # end if  
   
 iter <- iter + 1 #check the next number in the next circulation  
   
 } # end while loop  
 end <- Sys.time()  
 time <- end - start  
 y <- c(y, time)  
 print(time)  
}

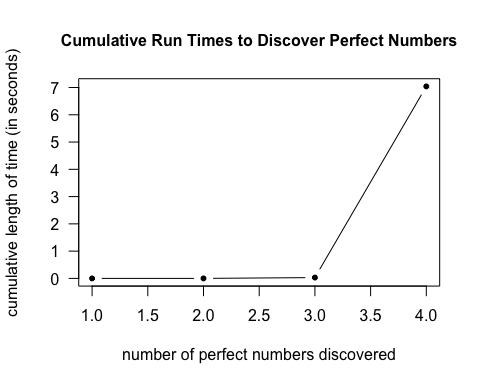
## [1] "6 is a perfect number"  
## Time difference of 6.41346e-05 secs  
## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"  
## Time difference of 0.0001699924 secs  
## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"  
## [1] "496 is a perfect number"  
## Time difference of 0.0304811 secs  
## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"  
## [1] "496 is a perfect number"  
## [1] "8128 is a perfect number"  
## Time difference of 7.040359 secs

x <- 1:4  
timeData <- data.frame("No of Perfect Numbers"= x, "Run Time"=y)  
kable(timeData, caption = "Run Time Table", align = "c" )

Run Time Table

|  |  |
| --- | --- |
| No.of.Perfect.Numbers | Run.Time |
| 1 | 0.0000641 |
| 2 | 0.0001700 |
| 3 | 0.0304811 |
| 4 | 7.0403590 |

plot(x, y, pch=20, type="b", xlab="number of perfect numbers discovered",  
 ylab="cumulative length of time (in seconds)",   
 main="Cumulative Run Times to Discover Perfect Numbers", las=TRUE, cex.main = 1.0)



## Question 3

1. Using a for loop, write code to compute the geometric mean of the numeric vector x < −c(4, 67, 3). Make sure your code (i) removes any NA values and (ii) prints an error message if there are any non-positive values in x.

geoMean <- function(x){  
   
 #remove any NA values  
 xWithoutNa <- x[!is.na(x)]  
 n <- length(xWithoutNa)  
 product <- 1  
   
 # print an error message if there are any non-positive values in x.  
 if (any(xWithoutNa<=0)) cat("Error: (", x, ") contains non-positive value \n")  
   
 else {  
 for (i in 1:n){  
 product <- product \* xWithoutNa[i]  
 geometricMean <- product^(1/n)  
 }  
 cat("The geometric mean of (", x, ") is", geometricMean, "\n")  
 }  
}  
  
x <- c(4, 67, 3)  
geoMean(x)

## The geometric mean of ( 4 67 3 ) is 9.298624

1. Test your code on the following cases and show the output: (i) {NA,4,67,3}, (ii) {0, NA, 6}, (iii) {67, 3, ∞}, and (iv) {−∞, 67, 3}.

x1 <- c(NA,4,67,3)  
x2 <- c(0, NA, 6)  
x3 <- c(67, 3, Inf)  
x4 <- c(-Inf, 67, 3)  
for (i in 1:4){  
 geoMean(eval(parse(text = paste0("x", i))))  
   
}

## The geometric mean of ( NA 4 67 3 ) is 9.298624   
## Error: ( 0 NA 6 ) contains non-positive value   
## The geometric mean of ( 67 3 Inf ) is Inf   
## Error: ( -Inf 67 3 ) contains non-positive value