Stat243 PS3

Meng Wang, SID: 21706745

Septembre 27, 2015

1 Problem 2

1.1 (a)

There are two problems we need to solve here: the first problem is to extract all the links from the downloaded file; the second problem is to find out among the links that it is the first debate of a specific years we are interested in. Let us deal with these two problems one by one.
The first problem is straightforward. Using the content on the material of unit3, we could download the data and parse, and exact the urls in the parsed file. The links are saved in *links*

```r
library(XML)
library(curl)

# download all content on the page
URL <- "http://www.debates.org/index.php?page=debate-transcripts";
page <- htmlParse(URL);

# extract the all the links on page for future use
allSpeech <- getNodeSet(page, "//a[@title]")
links <- sapply(allSpeech, xmlGetAttr, "href")
```

Since the titles on the website debate contain the key word 'First'. What we need to do for the second problem is, to transfer the data download to string type and find the index of the strings which matches the keyword 'First'and the number of year. The position are stored in the *index*

```r
# save the XML as string for keyword matching
stringSpeech <- xpathSApply(page, "//a[@title]",saveXML)

# find the index that contains key word "First" and
years <- c("1996", "2000", "2004", "2008", "2012")

# function that returns the index of the year
findIndex <- function(year, speechstr=stringSpeech){
  intersect(grep(year, speechstr), grep("First", speechstr))
}
# find the index for every years' first speech
index <-sapply(years, findIndex)
```

The results are stored in 'index'and the element is names integer. So we could find the link and the content by using the year directly. For example, we want the first speech in year 2006, here is the results,

```r
links[index["1996"]]

## [1] "http://www.debates.org/index.php?page=october-6-1996-debate-transcript"
```

1.2 (b)

This part is very straight forward. Exact the debate part first and then replace the nextline in HTML to the corresponding ones in the text file. Here is the code.

```
# function that returns the debate in txt file format
# input: string year = "2000"
# output: string contains the debate content
extractTXT <- function(year = "2012"){
  # Parse the document
  debate <- htmlParse(links[index[year]])
 # extract the debate part
 content  <- xpathSApply(debate, '//div[@id="content-sm"]',saveXML)
 # change format from HTML to txt
 content <- gsub("<br>|<br/>","\n", content)
 content <- gsub('<(.*?)>', "", content)
 return(content)
}
```

Extract all the spoken part for each year and put them into 'spokenDebate'.

```
spokenDebate <- sapply(years, extractTXT)
# test
# spokenDebate["1996"]
```

1.3 (c)

In this part, we are asked to split the who speech and merge them together, so that we have the speech for each candidate for future analysis. The good news is that for lots of the paragraph, it starts with the speaker's name. So we could easily split the whole speech to paragraph, and scan the paragraphs to see whether there are any speaker's name included or not. This is very straight-forward. The tricky part of the work is however, how to know which speaker those paragraphs which do not have any speaker's name belongs to. Naturally we know it belongs to the speaker of the last paragraph with a speaker's name. In summary, the algorithm to this problem is, first, split the speech to paragraphs; Secondly, loop through the paragraphs. For each paragraph, check whether contains the name of the speaker or not. If it contains the speaker's name, we should save it. If it does not, also it does not belongs to any speaker's, we should check who the last speaker is. If the last speaker is the one we are looking for, then we should save this paragraph too. If it is not, we should continue looping. Here is the code.
First, we need a function to find out all the speakers of that year,

```
## function used to find the candidate by splitting the string to words
## input: year = "1996", data = spokenDebate
## output: name of the speakers
findSpeaker <- function(year , data=spokenDebate){
 # change the '\n' to space for word splitting
 text_no_newline <- gsub("\n", " ", data[year])
 # split the word
 split_word <- strsplit(text_no_newline," ")[[1]]
 # find the names which has the pattern: all capital words followed by a :
 speaker <- split_word[grep("[[:upper:]]:", split_word)]
 speaker <- unique(speaker)
 return(speaker)
 }
```

These are the speakers for 1996, as a test for the function

```r
# test
findSpeaker("1996")

## [1] "LEHRER:"  "CLINTON:" "DOLE:"
```

Next, we need a function that can do the loop we discussed previously and merge all the paragraphs for a speaker at a specific year. Another little function is required to check whether this paragraph belongs to any speaker of this year or not. Here are the codes,

```r
## function: find whether this string belongs to any speaker or not
checkOwner <- function(allSpeakers, str){
  check = 0;
  for (i in 1:length(allSpeakers)){
    if (grepl(allSpeakers[i], str)) check = check+1
  }
  return(check)
}

## function: exact the speech given by each candidate
## Input: speaker = "CLINTON:", yera = "2006", data = spokenDebate
## Output: speech of the speaker
speakerSpeech <- function(speaker, year, data=spokenDebate){
 # fisrst get rid of "Laughter" and "Applause"
 spokenData <- gsub("Laughter", "", data[year])
 spokenData <- gsub("Applause", "", spokenData)
 # split the string into paragraphs
 split_para <- strsplit(spokenData[year],"\n")[[1]]
 # remove the blank paragraph
 split_para <- split_para[! split_para == ""]
 # for loop to go throught all the paragraphs
 speak_talk = 0 # 1 means this paragraph belong to the speaker
 k = 1
 speak_speech <- c("")
 for (para_num in 1:length(split_para)){
   if (grepl(speaker, split_para[para_num])){
     # this paragrah does belongs to the speaker
     speak_talk = 1;
     speak_speech[k] <- split_para[para_num]
     k = k+1
   }
   else if(checkOwner(findSpeaker(year), split_para[para_num]) == 0){
     if (speak_talk == 1)
     { # the paragrah is a continuing of the last paragraph
       speak_speech[k] <- split_para[para_num]
       k = k+1
     }
   }
   else{ # this paragraph belongs to someone else
     speak_talk = 0
   }
 }
 # put all paragraph together
 speak_speech <- paste(speak_speech, set="", collapse="")
 # delete the speaker's name due to duplication
 speak_speech <- gsub(speaker, "", speak_speech)
 return(speak_speech)
}
```

The code has been tested as following. The result is too long so please run the corresponding R code for the results.

```
# test
#speakerSpeech("CLINTON:", "1996")
```

Finally, we could loop through all the speakers with a year and split the debate of that year to several speakers. For efficiency, we use 'sapply'.

```
# function find all the speeches for one year only
OneYear_SpeakerSpeech <- function(one_year, data = spokenDebate){
  return (sapply(findSpeaker(one_year), speakerSpeech, year = one_year))
}
```

Since the result are stored characteristically, we could directly check the speech of a speaker by his/her name, for example

```
 # test
 speech1996 <- OneYear_SpeakerSpeech("1996")
 speech1996["CLINTON:"]
```

Then loop through the years we are interested with 'sapply', and we should have the results.

```
# all the years, all the speeches by the candidates
 AllSpeach <- sapply(years, OneYear_SpeakerSpeech)
```

1.4 (d)

This part is very straight forward. To split the sentences, we need to replace dot with nextline character. To split the text to words, we replace nextline character to space. Here are the functions and test case.

```
## function: exact a string to sentences
splitToSentence <- function(str_data = spokenDebate["1996"]){
  # change the '.' to "\n" for sentence splitting
  text_data <- gsub("\\.", "\n", str_data)
  sentence <- strsplit(text_data, "\n")[[1]]
  # remove empty lines
  sentence <- sentence[!sentence == ""]
  return(sentence)
}
# test
# sentence1996 <- splitToSentence(spokenDebate["1996"])

## function: extract the words from string
splitToWord <- function (str_data = spokenDebate["1996"]){
  text_noline <- gsub("\n", " ", str_data)
  # split the word
  split_word <- strsplit(text_noline," ")[[1]]
  # remove empty elements
  split_word <- split_word[! split_word == ""]
  return(split_word)
}

# test
#word1996 <- splitToWord(spokenDebate["1996"])
```

1.5 (e)

With the function in part(d), we just to count the length of the words splitted from the speech of each candidate, for the number of the words used. For the number of characters, we could use the function ‘nchar’in R to compute the length of each word and then sum the length of each word together. Average word length shall be the ratio of the two quantities we calculated. The results are stored in list format. Here is the function,

```
## function: do the word statistics and return a list of the data
wordStat <- function (candidate, years, data = spokenDebate){
 # get the speech for the candidate of that year
 candSpeech <- speakerSpeech(candidate,  years, data)
 # split the speech to words
 word <- splitToWord(candSpeech)
 # count the length of the words
 word_number <- length(word)
 # count the length of each word and sum up
 char_length <- sum(nchar(word))
 # get the average
 ave_word_length = char_length / word_number
 # put the statistics to a list
 result <- list(WordNumber = word_number, CharLength=char_length, %AveWordLength=ave_word_length)
 return(result)
}
```

And here is the result of a test case,

```
 wordStat("CLINTON:", "1996")

## $WordNumber
## [1] 7380
##
## $CharLength
## [1] 33323
##
## $AveWordLength
## [1] 4.515312
```

Then we loop the function with each candidate of a year, for every year, with ‘sapply’, like what we did in last part. Here is the code,

```
## function: do the word statistics for each candidate for a year
wordStat_Year <- function (one_year, data = spokenDebate){
  # get the speaker of that year
  candidate <- findSpeaker(one_year)
  # find all the word statistics of the candidates that year
  word_stat_year <-sapply(candidate, wordStat, years = one_year)
  return (word_stat_year)
}

# test
# wordStat1996 <- wordStat_Year("1996")
# wordStatAll <- sapply(years, wordStat_Year)
```

Here is the result (I copied it directly from Rstudio, please run the R code)

```
> wordStatAll
$`1996`
              LEHRER:  CLINTON: DOLE:
WordNumber    1218     7380     8114
CharLength    5800     33323    36305
AveWordLength 4.761905 4.515312 4.474365

$`2000`
              MODERATOR: GORE:    BUSH:
WordNumber    1688       7208     7451
CharLength    8056       32310    33132
AveWordLength 4.772512   4.482519 4.446651

$`2004`
              SPEAKERS: LEHRER: KERRY:   BUSH:
WordNumber    26        1362    7150     6345
CharLength    148       6855    31708    28395
AveWordLength 5.692308  5.03304 4.434685 4.475177

$`2008`
              SPEAKERS: LEHRER:  OBAMA:   MCCAIN:
WordNumber    52        2797     15358    14392
CharLength    316       12779    69128    65516
AveWordLength 6.076923  4.568824 4.501107 4.552251

$`2012`
             SPEAKERS: LEHRER:  OBAMA:   ROMNEY:
WordNumber    11        1962     7341     7860
CharLength    67        9174     33647    35161
AveWordLength 6.090909  4.675841 4.583435 4.47341
```

Here are some comment: First, it seems that the number of words for each year and each candidate does not change that much, except year 2008 debate between Obama and McCain, which almost twice as long as the other years. While the word length does not change that much at all, for any year and any candidate. Secondly, the number of words for each speaker at the same year, are almost the same. Speaking more words is not directly related with whether he or she will win the election or not.

1.6 (f)

For the last part, we just need to use the function in the previous part to split the speech into words, and count the words we are interested in. You may notice I decided to write the code for each word, instead of writing a function to look for each word. Since the code looking for number of words are very short, it does not save me any efforts to make it a function. So I directly write it. Here is the function to count the words for one speaker of a year.

```r
## function: do the words counts for a speaker for a specific year
wordsCount <- function (speaker, year, data = spokenDebate){
 # first find the speech by that speaker
 speech <- speakerSpeech(speaker, year, data=spokenDebate)
 words <- splitToWord(speech)
 num_I = length(words[words == "I"])
 num_we = length(words[words == "we"])
 # number of words contains America/American
 num_America = length(words[grep("America", words)])
 # number of words contains democra
 num_democra = length(words[grep("democra", words)])
 num_republic = length(words[grep("republic", words)])
 num_Democrat = length(words[grep("Democrat", words)])
 num_Republican = length(words[grep("Republican", words)])
 num_free = length(words[(words == "free") | (words == "freedom")])
 num_war = length(words[words == "war"])
 num_God = length(words[words == "God"])
 num_GodBless = length(words[words == "God bless"])
 num_Jesus = length(words[words == "Jesus" | words == "Christ" | words == "Christian"])
 num_words <- list(num_I = num_I, num_we = num_we, num_America = num_we, num_democra = num_democra,
                   num_republic = num_republic, num_Democrat=num_Democrat, num_Republican=num_Republican
                   num_free = num_free, num_war = num_war, num_God = num_God, num_GodBless= num_GodBles
 return(num_words)
}
```

This is the test

```r
# test
clinton1996 <- wordsCount("CLINTON:","1996")
```

```
> clinton1996
$num_I
[1] 203

$num_we
[1] 77

$num_America
[1] 77

$num_democra
[1] 6

$num_republic
[1] 0

$num_Democrat
[1] 1

$num_Republican
[1] 10

$num_free
```

```
[1] 2

$num_war
[1] 2

$num_God
[1] 0

$num_GodBless
[1] 0

$num_Jesus
[1] 0
```

Then loop the speaker and the years with ‘sapply’, we have the final results

```
## function: do the words count for all the speakers for that year
wordsCount_Year <- function(one_year, data = spokenDebate){
  # get the speaker of that year
  candidate <- findSpeaker(one_year)
  # find all the word statistics of the candidates that year
  word_count_year <-sapply(candidate, wordsCount, year = one_year)
  return (word_count_year)
}
# test
wordCountALl <- sapply(years, wordsCount_Year)
```

Here are the results, copied from Rstudio. They are stored in list format. Please run the R code for this result

```
> wordCountALl
$`1996`
               LEHRER: CLINTON: DOLE:
num_I          12      203      213
num_we         9       77       78
num_America    9       77       78
num_democra    0       6        0
num_republic   0       0        0
num_Democrat   1       1        12
num_Republican 2       10       12
num_free       0       2        0
num_war        0       2        0
num_God        0       0        1
num_GodBless   0       0        0
num_Jesus      0       0        0

$`2000`
               MODERATOR: GORE: BUSH:
num_I          11         195   171
num_we         7          61    64
num_America    7          61    64
num_democra    1          1     1
num_republic   1          0     0
num_Democrat   1          2     12
```

```
num_Republican 1          2     9
num_free       0          1     3
num_war        0          3     4
num_God        0          0     0
num_GodBless   0          0     0
num_Jesus      0          0     0

$`2004`
               SPEAKERS: LEHRER: KERRY: BUSH:
num_I          0         7       143    150
num_we         0         2       92     89
num_America    0         2       92     89
num_democra    0         1       2      4
num_republic   0         0       0      0
num_Democrat   0         1       0      0
num_Republican 0         1       1      0
num_free       0         0       1      21
num_war        0         1       26     20
num_God        0         0       1      1
num_GodBless   0         0       0      0
num_Jesus      0         0       0      0

$`2008`
               SPEAKERS: LEHRER: OBAMA: MCCAIN:
num_I          0         18      234    332
num_we         0         8       338    226
num_America    0         8       338    226
num_democra    0         0       6      8
num_republic   0         0       0      0
num_Democrat   0         2       0      6
num_Republican 0         2       6      14
num_free       0         0       4      4
num_war        0         0       20     6
num_God        0         0       0      0
num_GodBless   0         0       0      0
num_Jesus      0         0       0      0

$`2012`
               SPEAKERS: LEHRER: OBAMA: ROMNEY:
num_I          0         18      91     145
num_we         0         10      110    71
num_America    0         10      110    71
num_democra    0         0       0      1
num_republic   0         0       0      0
num_Democrat   0         1       8      7
num_Republican 0         1       9      9
num_free       0         0       2      7
num_war        0         0       2      0
num_God        0         0       0      0
num_GodBless   0         0       0      0
num_Jesus      0         0       0      0
```

Here is the comment: It seems the candidates never say religious words(Jesus, Christ). It may be because most of the debate is focused on the policy of economy, diplomacy and social affairs. Keywords like freedom and war are definitely mentioned, but not very frequently. American and democracy tend to be popular words in the debate.

2 Problem 3

2.1 (a)

This part should be straight forward. We a 2d array to store the position (x, y) for each step. For each step, we generate a random number between 0 and 1. Based on the number, we move the position left, right, up and down for $[0, 0.25), (0.25, 0.5], (0.5, 0.75], (0.75, 1]$. We set another parameter path_ return, if it is true, we return the path. If it is false, we only return the last position. It is false as default.

```r
set.seed(0)
randomWalkGen <- function(step_num, path_return=FALSE){
  if(is.numeric(step_num) && floor(step_num) == step_num && step_num > 0){
    pos_x = 0
    pos_y = 0
    path <- matrix(0, nrow=step_num+1, ncol=2)
    for(i in 1:step_num){
      rand <- runif(1)
      if(rand <= 0.25){
        pos_x = pos_x -1 # left
      }else if(rand <= 0.5){
        pos_x = pos_x + 1 # right
      }else if(rand <= 0.75){
        pos_y = pos_y + 1 # up
      }else{
        pos_y = pos_y - 1 # down
      }
      path[i+1,] <- c(pos_x, pos_y)
    }
    if (path_return == FALSE) return(path[step_num + 1,])
    else return(path)
  }else{
    cat("Invalid Input!")
  }
}
```

Here are some test cases,

```r
 randomWalkGen(3, TRUE)

##      [,1] [,2]
## [1,]    0    0
## [2,]    1    0
## [3,]    1    1
## [4,]    1    0

 randomWalkGen(3)

## [1] -1 -2

 randomWalkGen(0)

## Invalid Input!
```

```
randomWalkGen("Hello GSI!")

## Invalid Input!
```

2.2 (b)

Now we use the vector structure instead of for loop. We should just generate a vector contains the ramdom number and decide each movement for each step. Then use cumsum to sum all the steps between each step as the position. Here is the code

```
randomWalkGenVector <- function(step_num, path_return=FALSE){
  if(is.numeric(step_num) && floor(step_num) == step_num && step_num > 0){
    set.seed(0)
    numbers <- runif(step_num, 0, 1)
    pos_x <- matrix(0, nrow = step_num, ncol = 1)
    pos_y <- matrix(0, nrow = step_num, ncol = 1)
    pos_x[numbers <= 0.25] = -1 # move left
    pos_x[numbers >0.25 & numbers <=0.5] = 1 # move right
    pos_y[numbers > 0.5 & numbers <=0.75] = 1# move up
    pos_y[numbers > 0.75 & numbers <=1.0] = -1 # move down

    path <- matrix(0, nrow = step_num+1, ncol = 2)
    path[2:(step_num+1),1] = cumsum(pos_x)
    path[2:(step_num+1),2] = cumsum(pos_y)
    if (path_return == FALSE) return(path[step_num + 1,])
    else return(path)
  }else{
    cat("Invalid Input!")
  }
}
```

Here are some tests

```
randomWalkGenVector(3, TRUE)

##      [,1] [,2]
## [1,]    0    0
## [2,]    0   -1
## [3,]    1   -1
## [4,]    2   -1

randomWalkGenVector(3)

## [1]  2 -1

randomWalkGenVector(0)

## Invalid Input!

randomWalkGenVector("Hello GSI!")

## Invalid Input!
```

2.3 (c)

This part is very similar to unit 5 examples. First, define a S3 class as following,

```r
myWalk <- list(origin = c(0,0), step = 0, path = matrix(0, nrow =1,ncol = 2))
class(myWalk) <- 'rm'
```

The class rm contains those members, the random walk start point origin, the number of steps in this walk step, the path of the random walk path.
The constructor is build up so that we have a clear path for a object during the initialization process.

```r
# build up constructor
rm <- function(origin = NA, step = NA){
  if(is.numeric(step) && floor(step) == step && step > 0){
    set.seed(0)
    numbers <- runif(step, 0, 1)
    pos_x <- matrix(origin[1], nrow = step, ncol = 1)
    pos_y <- matrix(origin[2], nrow = step, ncol = 1)
    pos_x[numbers <= 0.25] = -1 # move left
    pos_x[numbers >0.25 & numbers <=0.5] = 1 # move right
    pos_y[numbers > 0.5 & numbers <=0.75] = 1# move up
    pos_y[numbers > 0.75 & numbers <=1.0] = -1 # move down

    path <- matrix(0, nrow = step+1, ncol = 2)
    path[2:(step+1),1] = cumsum(pos_x)
    path[2:(step+1),2] = cumsum(pos_y)
  }else{
    path = origin
  }

  obj <- list(origin = origin, step = step, path = path)
  class(obj) <- "rm"
 return (obj)
}
```

We can initialize an object

```r
 # initialize an object
 aWalk <- rm(c(0,0), 2)
```

Define method print,

```r
# method print
print.rm <- function(obj){
  return(cat("Random Walk, Start at (", obj$origin, "), after", obj$step, "steps,",
                          "arrive final position at(", obj$path[obj$step+1,], ")"))
}
```

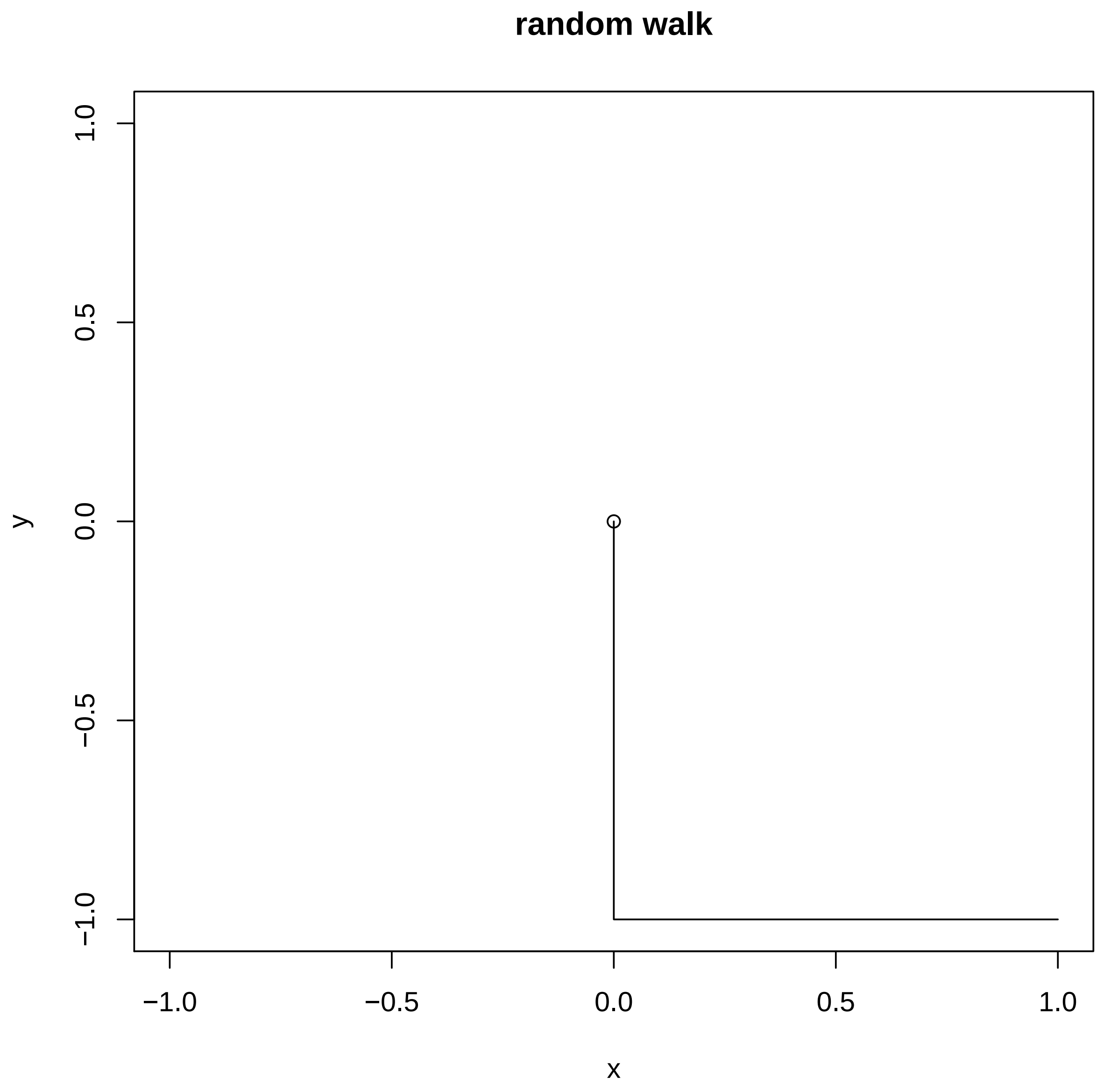
Here is the result

```r
 print(aWalk)

## Random Walk, Start at ( 0 0 ), after 2 steps, arrive final position at( 1 -1 )
```

Plot method is defined to plot the points and path of this random walk

```r
plot <- function(x, ...) UseMethod("plot")
plot.rm <- function(obj){
  plot(obj$origin[1],obj$origin[2], xlab="x", ylab="y", main = 'random walk')
  return(lines(x = obj$path[,1], y = obj$path[,2]))
}
plot(aWalk)
```



```
0 -1
```

At last, a start method to change the the origin

```r
'start<-' <- function(x, ...) UseMethod("start<-")
'start<-.rm' <- function(obj, value){
  obj$origin = value
  obj$path[1:(obj$step+1),] = obj$path[1:(obj$step+1),] + value
  return (obj)
}
start(aWalk) <- c(5,7)
aWalk

## Random Walk, Start at ( 5 7 ), after 2 steps, arrive final position at( 6 6 )
```

** R code saved in HW3.R; all codes have been run and tested*