STAT243: PS 3

Xinyue Zhou

September 30, 2015

1.

This is a brief but very comprehensive paper trying to solve the scientific programming problem met by scientists, including adopting good coding manner, using tools for collaborating, and debbuging tips.

Programmer should adopt good coding manners for people to understand, read, modify and reused. It's also helpful if the programmer want to use that piece of code later. The specific designs for those issues are friendly varible(object) types and names, small pieces of explanation documentations inserting in the code, and separate code intro chunks properly. There are lot of interesting things in talking about ”manners”. In my opinion, the most fantastic part is how to elegently separate the code into chunks or functions, so that chunks can be manipulate and modified easily later. Based on my own experience, I prefer relatively small chunks, targeting only one task for each. How to define a task is another question. Definitely there is no obsolute rule for this. For me, things in a loop can be a task. I would extract this part out and write it as a function. Analysis around one object can be a task, I would write it separately and list the results. Those work pretty well until now, which are really good for checking errors and debugging.

Working on a project with a group of people is another general situation in real life, thus how to handle conflict and to keep up the newest version as well as to save the older versions are meaningful topics. I do realized there are problems using Dropbox and Google Doc for group project. The Version control System (VCS) is good for handling such group works. The VCS highlights the differences and requires them to resolve any conflicts before accepting the changes. It can also stores the entire history of those files, allowing arbitrary versions to be retrieved and compared. I am really surprising this paper recommand us to put everything that manually created in version control in practice. Would that be too redundant? Would that occupy a lot of space in the computer if it is repeated again and again? (I kind of know that it is just a snapshot, but it should have some information stored in the computer anyway.)

Debugging part in this paper is enlightening for me. I used some of them unconciously before, while now I can implement them all in order to consider all the possible cases. I will summarize a little bit here.
1. Use defensive programming techeque: stop the program once error occurs and explanation for the code as well.
2. Write and run tests: test just one part each time; write tests in a chunk so that it can be run over and over again.
3. How to chose cases: use the previous bug (fix or not) as the case.
4. Use oracles (it seems like I never use this technique before).
5. Use a symbolic debugger: keep track of the execution of the program to find and fix bugs.

2.
a) Read links into R
step1: Load Library needed

```
##load libraries
require(XML)

## Loading required package:  XML

require(curl)

## Loading required package:  curl

require(RCurl)

## Loading required package:  RCurl
## Loading required package:  bitops
```

```r
require(stringr)

## Loading required package:  stringr
```

step2: parse the link to find what we need

```r
##extract URL from the main
mainPage <-"http://www.debates.org/index.php?page=debate-transcripts"
parseLink = htmlParse(mainPage)
###find candidate namelist
First=getNodeSet(parseLink, "//a[contains(text(),'First')]")
findnames = unique(grep ("(1996|2000|2004|2008|2012)",
                         lapply(First,toString.XMLNode),value=TRUE))
```

step2.1: I add a namelist here, which make it easy to extract the statement of candidate. Here is a special case. In the debate, it uses name ”McCain” instead of ”Cain”, therefore I change the element of namelist.

Very Important: I use namelist as a global variable in the whole problem.

```r
namelist = unlist(str_extract_all(findnames,"([A-Z][a-z]+)-[A-Z][a-z]+"))
#it actually use name "McCain" instead of "Cain"
namelist[2] = "McCain-Obama"

##find out datasets needed
datasets= sapply(First, xmlGetAttr, "href")
repp = unique(grep ("(1996|2000|2004|2008|2012)",datasets,value=TRUE))
repp

## [1] "http://www.debates.org/index.php?page=october-3-2012-debate-transcript"
## [2] "http://www.debates.org/index.php?page=2008-debate-transcript"
## [3] "http://www.debates.org/index.php?page=september-30-2004-debate-transcript"
## [4] "http://www.debates.org/index.php?page=october-3-2000-transcript"
## [5] "http://www.debates.org/index.php?page=october-6-1996-debate-transcript"
```

b) Write dataset on txt file to see the format
step1:extract text

```r
##EXTRACT TEXTS
extract.txt= function(html){
  html = getURL(html)
  doc <- htmlParse(html, asText=TRUE)
  plain.text <- xpathSApply(doc, "//p/text()", xmlValue)
  return(plain.text)
}
```

step2:write

```r
##write onto txt
for (i in 1:length(repp)){
  a = extract.txt(repp[i])
  years = str_extract(repp[i],"[:digit:]{4}" ) ##EXTRACT YEAR
  write(a,file=paste(years, "debate.txt", sep = ""))
}
```

Note: I am not going to output txt here. It’s too long. From txt, I notice that all the candidates’ state followed with an uppercase of string following with ”:”. This can be the criterion later for extracting text.
c) Write dataset on txt file to see the format
Helper function1: select names see the detailed explanation in code.

```r
#after taking a look at the txt file we got from #b, it is not that clean at
# the first glance. Now we first clean it and then put them in a better manner

#helper func1---select names:
#input- a namelist from property of files, it's
  #a list with elements like"Obama-Romney"
  #year of speech
#output- a list candidate that year
select_name<-function(namelist,year){
  years = rev(c(1996,2000,2004,2008,2012))
  idx = seq(1,5)
  my_idx = idx[years==year]
  candidate = str_split(namelist,"-")
  return(candidate[[my_idx]])
}
```

Helper function2: split chunks see the detailed explanation in code.

```r
#helper func2---split chunks: make candidate text in turn
#input- candidates namelist and corresponding text
#output- a table that candidate statement in turn
split_chunks<-function(candi_names,candi_text_list){
  if (length(candi_names)!=length(candi_text_list)) {
    print("invalid input")
    return
  }
  #create a mask, including idx that share the same candidate
  #name with it successor
  idx=which(candi_names[1:(length(candi_names)-1)]
            ==candi_names[2:(length(candi_names))])
  #print(idx)
  candi_names2 = candi_names[-idx]
  #note I reverse the idx here to avoid a candidate
  #continuous speak several times.
  for (i in rev(idx)){
    candi_text_list[i] = paste (candi_text_list[i],candi_text_list[i+1])
  }
  candi_text_list2 = candi_text_list[-(idx+1)]
  debate_tbl= cbind(candi_names2,candi_text_list2)
  colnames(debate_tbl) = c("name","text")
  return(debate_tbl)
}
```

Helper function3: clean the txt and output see the detailed explanation in code.

```r
##check
clean_text <-function(namelist,year){
  filename = paste(year,"debate.txt",sep="")
  textfile =  readLines(filename)
  textfile<-paste(textfile,collapse=" ")

  #print out the number of laughter and applause each year
  n_laugter = str_count(textfile,"\\(LAUGHTER\\)")
  n_applause = str_count(textfile,"\\(APPLAUSE\\)")
  cat("Count laugther and applause in ", year,":","\n")
  print(c(count_laugter=n_laugter,count_applause=n_applause))
  #clean all the tags
  textfile<-gsub("\\(LAUGHTER\\)|\\(APPLAUSE\\)|\\(CROSSTALK\\)","",
                 textfile, ignore.case=TRUE)

  ##notice there always just title in the first line, to match names and text,
  ##select out the names
  names = unlist(str_extract_all(textfile, "[A-Z]+: "))
  cand1 = paste0(toupper(select_name(namelist,year)[1]),": ")#;print(cand1)
  cand2 = paste0(toupper(select_name(namelist,year)[2]),": ")#;print(cand2)
  #which((names==cand1)|(names==cand2))
  candi_names = names[(names==cand1)|(names==cand2)]
  ##we just keep the debates between candidates
  text_list = str_split(textfile,pattern = "[A-Z]+: ")
  text_list = unlist(text_list)[-1]
  candi_text_list = text_list[(names==cand1)|(names==cand2)]
  ##split chunks
  debate_tbl = split_chunks(candi_names,candi_text_list)
  return(debate_tbl)
}
```

Let's see the result of 2012 here to get an idea, like see the 11's row. It can also give us the number of tags in 2012.

```
##see the result of 2012 here
test1 = clean_text(namelist,2012)

## Count laugther and applause in  2012 :
##  count_laugter count_applause
##              4              1

test1[11,]

##          name          text
##     "OBAMA: " "I like it. "
```

d) Two spliters
spliter1: split sentenses Split sentenses using punctuation marks like ".","...","?"

```
sentence_spliter<-function(table){
  sentences=str_split(table[,2],pattern = "\\. |\\? |\\.\\.\\. ")
  return(sentences)
}
```

spliter2: split words the pattern include type like "I'm"

```
word_spliter<-function(table){
  words = str_extract_all(pattern="([\\w][\\w\\']*\\w|\\w)", table[,2])
  return(words)
}
```

e) Count table of Words, Characters and average length of words
Helper Function1: reconstruct table Reconstruct table to be just two rows for two candidates in the debate. The second column is long string of each candidate speech in this debate.

```
reconstruct<-function(table){
  # wordoflist=word_spliter(table)
  candidate = table[1:2,1]
  cand_speech1=NULL
  cand_speech2=NULL
  for (i in 1:dim(table)[1]){
    #statement of first candidate
    if (i%%2){
      cand_speech1 = paste(cand_speech1,table[i,2])
    }
    #statment of second candidate
    else{
      cand_speech2 =  paste(cand_speech2,table[i,2])
    }
  }
  rectr_tbl = cbind(candidate,c(cand_speech1,cand_speech2))
  return(rectr_tbl)
}
```

Helper Function2: Count Function Restructed dataset and split it using $word_spliter$ function written previously. $wordoflist$ gives us a list of list, correspond to list of word of differenct candidate. At last, calculate average using nword and nchar.

```
  count_word_char<-function(table){
    rectr_tbl = reconstruct(table)
    wordoflist=word_spliter(rectr_tbl)
    count_word1 = length(wordoflist[[1]])
    count_word2 = length(wordoflist[[2]])#;print(count_word2)
    count_char1 = sum(nchar(wordoflist[[1]]))
    count_char2 = sum(nchar(wordoflist[[2]]))
    count_word = c(count_word1,count_word2)
    count_char = c(count_char1,count_char2)
    count_table = data.frame(matrix(nrow=2, ncol=4))
    count_table[,1] = rectr_tbl[,1]
    count_table[,2] = count_word
    count_table[,3] = count_char
    count_table[,4] = count_char/count_word
    colnames(count_table) = c("candidate","count_word","cound_char","average")
    return(count_table)
  }
count_word_char(test1)

##   candidate count_word cound_char  average
## 1    OBAMA:        7315      32523 4.446070
## 2  ROMNEY:        7835      33812 4.315507
```

Construct the table

```
  ##get tables
years = c(1996,2000,2004,2008,2012)
summary_table = data.frame(matrix(nrow = length(years)*2, ncol = 4))
for (i in 1:length(years)){
  table=clean_text(namelist,years[i])
  count_table = count_word_char(table)
  summary_table[(2*i-1):(i*2),] = count_table
}

## Count laugther and applause in  1996 :
##  count_laugter count_applause
##              0              0
## Count laugther and applause in  2000 :
##  count_laugter count_applause
##              0              1
## Count laugther and applause in  2004 :
##  count_laugter count_applause
##              3              2
## Count laugther and applause in  2008 :
##  count_laugter count_applause
##              4              4
## Count laugther and applause in  2012 :
##  count_laugter count_applause
##              4              1
```

```
colnames(summary_table) = c("candidate","count_word","count_char","average")
summary_table
```

```
##    candidate count_word count_char  average
## 1  CLINTON:        7422      32472 4.375101
## 2     DOLE:        8151      35106 4.306956
## 3     GORE:        7251      31424 4.333747
## 4     BUSH:        7499      32202 4.294173
## 5    KERRY:        7142      30614 4.286474
## 6     BUSH:        6355      27436 4.317231
## 7    OBAMA:       15294      66742 4.363934
## 8   MCCAIN:       14338      63154 4.404659
## 9    OBAMA:        7315      32523 4.446070
## 10  ROMNEY:        7835      33812 4.315507
```

Comment:
1)The average length of words candidate used is almost the same, which is around 4.3.
2) Obama and McCain have the longest debate in this five first-round debates. **f) Count Keywords Helper Function1: convert keys** Convert keys to be regular expression, so that we can count it later

```
#key has been clean here
keys = c("I", "we", "America{,n}",
         "democra{cy,tic}", "republic", "Democrat{,ic}", "Republican",
         "free{,dom}", "war", "God", "God Bless",
         "{Jesus, Christ, Christian}")
convert_key<-function(key){
  if (key=="I") {key="I\\b"}
  else if (key=="we") {key="we\\b|We\\b"}
  else if (key=="America{,n}") {key="American?\\b"}
  else if (key=="democra{cy,tic}") {key="democracy\\b|democratic\\b"}
  else if (key=="republic") {key="republic"}
  else if (key=="Democrat{,ic}") {key="Democrat(ic)?\\b"}
  else if (key=="Republican") {key="Republicans?\\b"}
  else if (key=="free{,dom}") {key="(F|f)ree(dom)?\\b"}
  else if (key=="war") {key="(W|w)ars?\\b"}
  else if (key=="God") {key="(G|g)od (?!bless)"}
  else if (key=="{Jesus, Christ, Christian}")
          {key="Jesus|Christs?\\b|Christians?"}
  else if (key=="God Bless") {key="(G|g)od bless"}
  else {print("I don't care this word")}
  return(key)
}
```

Helper Function2: Count Keys Given a debate table and the key we are searching, return the number of appearance of keywords for candidates in the table.

```
count_key<- function(table,key){
  key = convert_key(key)
  rectr_tbl = reconstruct(table)
  #remove punctuation marks to get a pure string of word
  clean_punc1 = str_replace_all(rectr_tbl[1,2],pattern="[:punct:]"," ")
  clean_punc2 = str_replace_all(rectr_tbl[2,2],pattern="[:punct:]"," ")
  count_k1 = str_count(clean_punc1,key)
  count_k2 = str_count(clean_punc2,key)
  #return(list(candidates=candidate,count=c(count_k1,count_k2)))
  return(c(count_k1,count_k2))
}
```

Helper Function3: Count Keys by Year Given a year, return a table of appearance of all targeted keywords for candidates.

```
count_byyear<-function(year){
  table = clean_text(namelist,year)
  candidates = table[1:2,1]
  countit=NULL
  for (i in 1:length(keys)){
    temp = count_key(table,keys[i])
    countit=cbind(countit,temp)
  }
  rownames(countit)=candidates
  colnames(countit)=keys
  return(countit)
}
```

Merge the table

```
keywords_count_table = NULL
for (i in years){
  temp = count_byyear(i)
  keywords_count_table=rbind(keywords_count_table,temp)
}

## Count laugther and applause in  1996 :
##  count_laugter count_applause
##              0              0
## Count laugther and applause in  2000 :
##  count_laugter count_applause
##              0              1
## Count laugther and applause in  2004 :
##  count_laugter count_applause
##              3              2
## Count laugther and applause in  2008 :
##  count_laugter count_applause
##              4              4
## Count laugther and applause in  2012 :
##  count_laugter count_applause
##              4              1

print(keywords_count_table)
```

```
##             I  we America{,n} democra{cy,tic} republic Democrat{,ic}
## CLINTON:  236 150          33               4        0            1
## DOLE:     275 166          42               0        0            7
## GORE:     229  96          13               1        0            1
## BUSH:     213 109          19               1        0            2
## KERRY:    197 150          43               2        0            0
## BUSH:     179 170          24               4        0            0
## OBAMA:    290 534          26               2        0            0
## MCCAIN:   426 336          36               2        0            2
## OBAMA:    119 182          18               0        0            4
## ROMNEY:   217 124          34               1        0            4
##           Republican free{,dom} war God God Bless
## CLINTON:          10          8   7   0         0
## DOLE:             12          1   8   0         1
## GORE:              2          1   7   0         0
## BUSH:              9          3   4   0         0
## KERRY:             1          4  37   0         1
## BUSH:              0         38  24   1         0
## OBAMA:             6          4  28   0         0
## MCCAIN:           14          6  20   0         0
## OBAMA:             9          3   5   0         0
## ROMNEY:            9          7   0   1         0
##           {Jesus, Christ, Christian}
## CLINTON:                           0
## DOLE:                              0
## GORE:                              0
## BUSH:                              0
## KERRY:                             0
## BUSH:                              0
## OBAMA:                             0
## MCCAIN:                            0
## OBAMA:                             0
## ROMNEY:                            0
```

Comments:
1)All the candidates used a lot of first person words like ”I” and ”we”
2)War is another keyword which is mentioned a lot.
3)Religious issues are rarely mentioned.
4)The appearance of ”republic/republican” is less than ”Democrat,ic/democracy,tic”

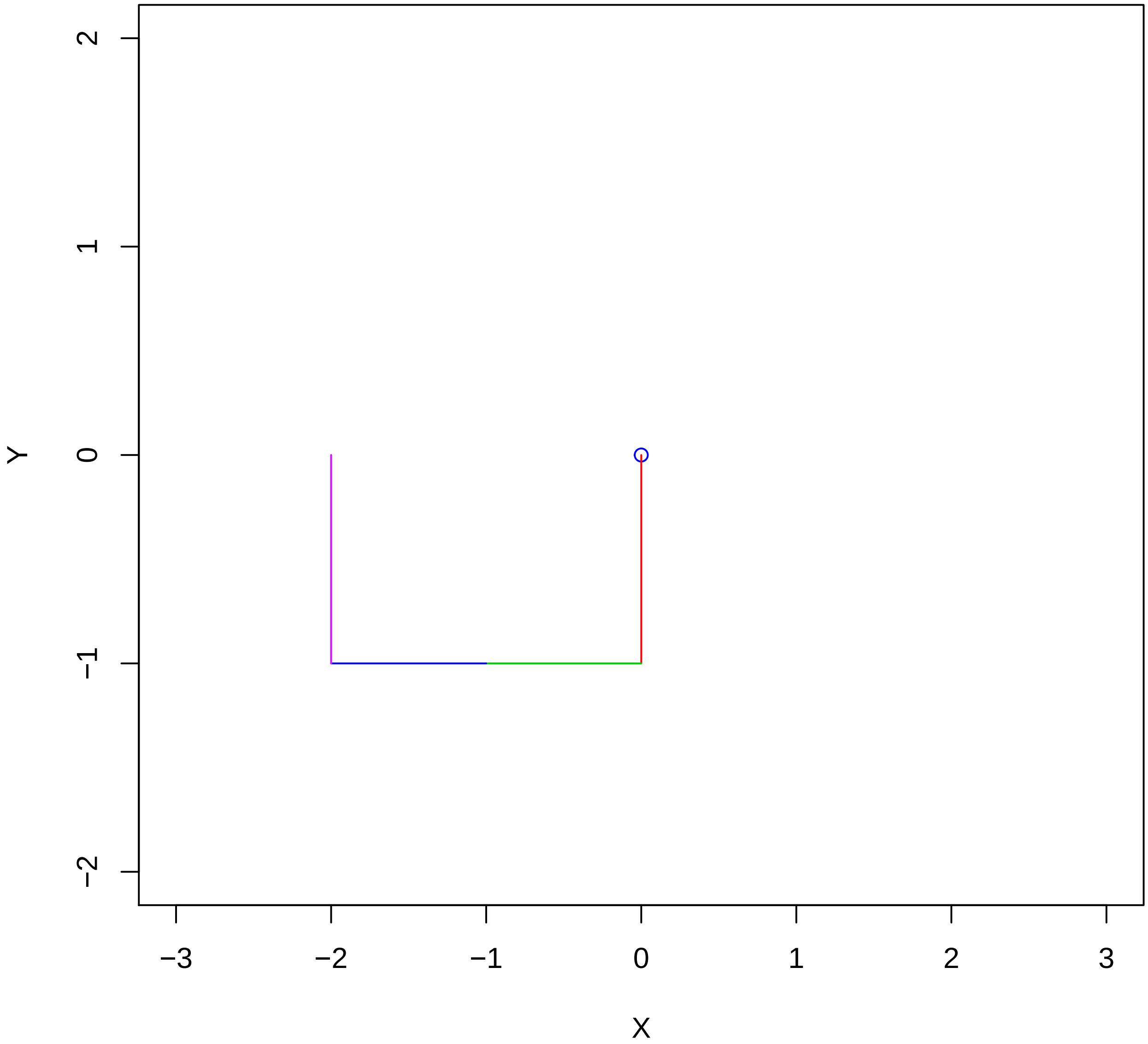
3.
a) and b) Vectorized Method

See the explanation in the code.

```
RandomWalk2= function(step,categ){
  if((step%%1!=0)|(step<0)|(step==0)) {
    print("Parameter step is not valid")
    return
  }
  if((categ!="position")&&(categ!="path")){
    print("Parameter category is not valid")
    return
  }
  x_path=numeric(step+1)
  y_path=numeric(step+1)
  set.seed(0) ##don't use set.seed in practice. This is just for debugging
  ##select direction
  #1:go up; 2:go down; 3:do right; 4:go left.
  #Those directions are with equal probability
  #then transfer it to the operation of each steps
  path = sample(1:4,step,replace=TRUE)
  x_path[which(path==1)+1]=1
  x_path[which(path==2)+1]=-1
  y_path[which(path==3)+1]=1
  y_path[which(path==4)+1]=-1
  x_pos = cumsum(x_path)
  y_pos = cumsum(y_path)
  if (categ=="position"){
    print(paste0("The final position is: (", x_pos[step+1], ",",y_pos[step+1],")"))
  }
  else if(categ=="path"){
    x_lim = max(abs(x_pos))
    y_lim = max(abs(y_pos))
    plot(0,0,xlim=c(-x_lim-1,x_lim+1),ylim=c(-y_lim-1,y_lim+1),xlab="X",ylab="Y",col=4)
    print("The path is:")
    for (i in 1:length(x_pos)) {
      print(paste0("(", x_pos[i], ",",y_pos[i],")"))
      lines(c(x_pos[i],x_pos[i+1]),c(y_pos[i],y_pos[i+1]),col=i+1)
    }
  }
}
```

Test case when *step* = 5.

```
RandomWalk2(5,"position")

## [1] "The final position is: (-2,-1)"

RandomWalk2(5,"path")
```



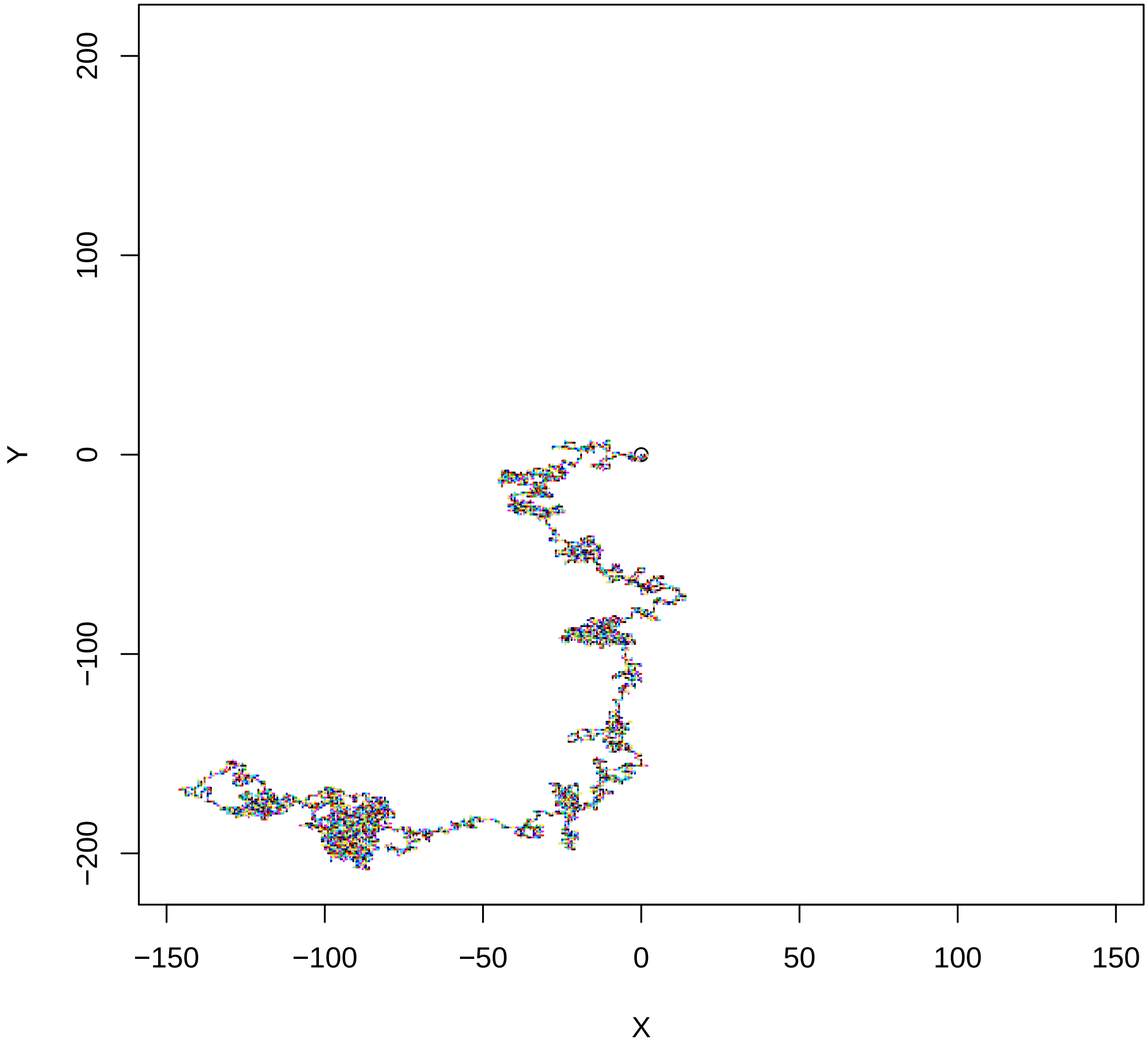
```
## [1] "The path is:"
## [1] "(0,0)"
## [1] "(0,-1)"
## [1] "(-1,-1)"
## [1] "(-2,-1)"
## [1] "(-2,0)"
## [1] "(-2,-1)"
```

c) write class
See the explanation in the code.

```
#the constructor
rw = function(step=0){
  if((step%%1!=0)|(step<0)) {
    print("Parameter step is not valid")
    return
  }
  x_path=numeric(step+1)
  y_path=numeric(step+1)
  set.seed(0)
  path = sample(1:4,step,replace=TRUE)
  #1:go up; 2:go down; 3:do right; 4:go left.
  #Those directions are with equal probability
  #then transfer it to the operation of each steps
  x_path[which(path==1)+1]=1
  x_path[which(path==2)+1]=-1
  y_path[which(path==3)+1]=1
  y_path[which(path==4)+1]=-1
  x_pos = cumsum(x_path)
  y_pos = cumsum(y_path)
  obj = list(length=step,x.path=x_pos,y.path=y_pos,
             x.origin = x_pos[1],y.origin=y_pos[1])
  class(obj)<-'rw'
  return(obj)
}
#the printer
#print out the origin, step and the final position
print.rw = function(object){
  cat("The walk starts from:(",object$x.origin,",",object$y.origin,").","\n")
  cat("After",object$length, "length of random walk,")
  cat("the final position of random walk is: (",object$x.path[length(object$x.path)]
      ,",",object$y.path[length(object$y.path)],")")
}
#plot method
plot.rw = function(object){
  x_lim = max(abs(object$x.path))
  y_lim = max(abs(object$y.path))
  plot(object$x.origin,object$y.origin,
       xlim=c(-1-x_lim,1+x_lim),
       ylim=c(-1-y_lim,1+y_lim),
       xlab="X",ylab="Y")
  for (i in 1:length(object$x.path))
    lines(c(object$x.path[i],object$x.path[i+1]),
          c(object$y.path[i],object$y.path[i+1]),col=i+1)
}
#operator []
'[.rw'=function(object,i){
  x = object$x.path[i+1]
  y = object$y.path[i+1]
  return(c("x"=x,"y"=y))
}
#start replacement
'start<-' <- function(object,...) UseMethod("start<-",object)
'start<-.rw' = function(object,value=c(0,0)){
  object$x.path= object$x.path+value[1]
  object$y.path= object$y.path+value[2]
  object$x.origin = value[1]
  object$y.origin = value[2]
  return(object)
}
```

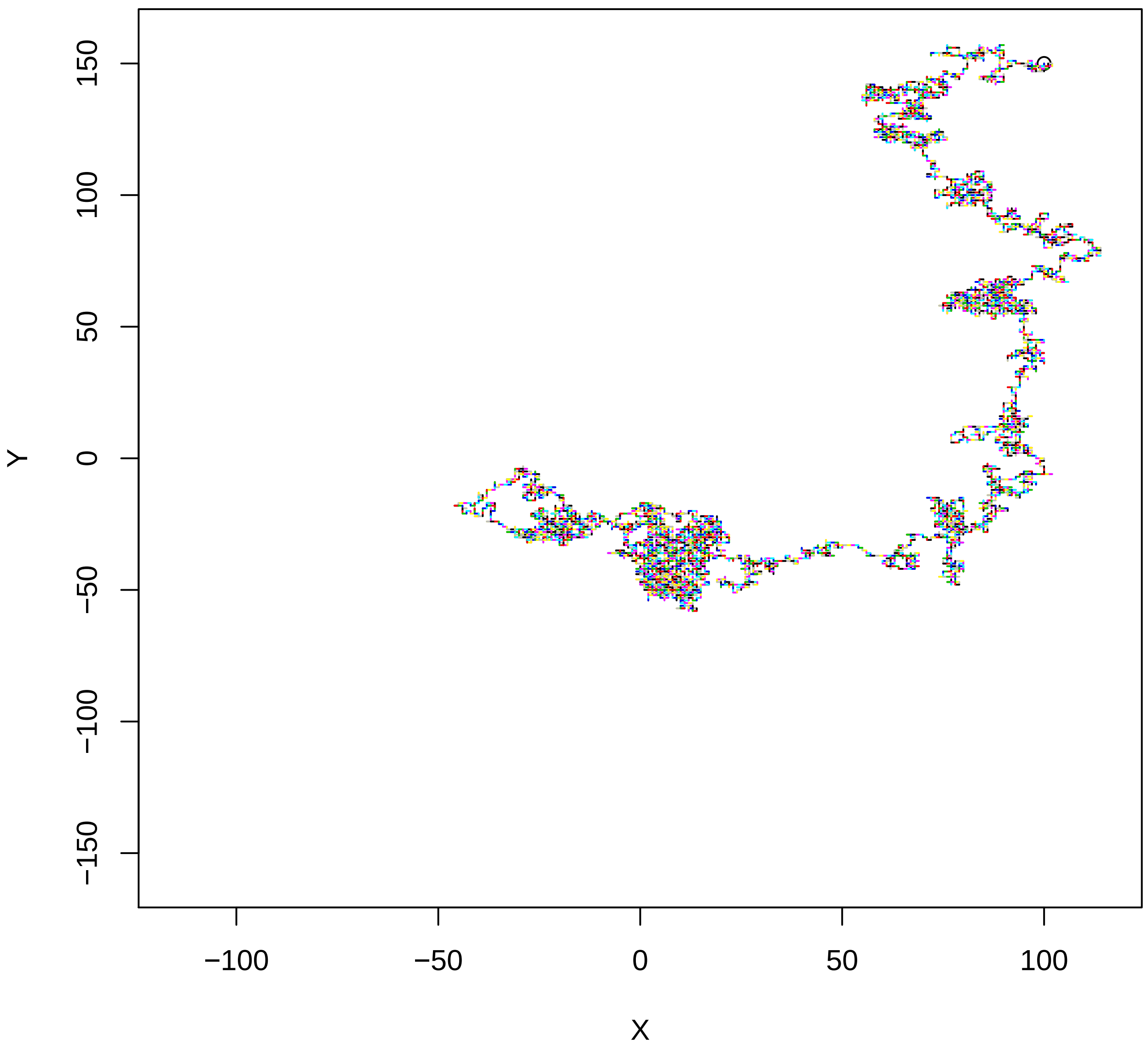
Try the test case $a = rw(5)$, and test its fiture.

```
a=rw(10000)
print(a)

## The walk starts from:( 0 , 0 ).
## After 10000 length of random walk,the final position of random walk is: ( -104 , -176 )

plot(a)
```



```
start(a)=c(100,150) #change its origin to (2,3)
print(a)
```

```
## The walk starts from:( 100 , 150 ).
## After 10000 length of random walk,the final position of random walk is: ( -4 , -26 )

plot(a)
```


150
100
50
0
−50
−100
−150
Y
−100
−50
0
50
100
X