R Assignment MAS6024

Registration Number:- 210116270

Part 1

We first create a 8x8 letter grid as mentioned.

```
lgrid <- matrix(NA, nrow = 8, ncol = 8)
lgrid[1,] <- c("r", "l", "q", "s", "t", "z", "c", "a")
lgrid[2,] <- c("i", "v", "d", "z", "h", "l", "t", "p")
lgrid[3,] <- c("u", "r", "o", "y", "w", "c", "a", "c")
lgrid[4,] <- c("x", "r", "f", "n", "d", "p", "g", "v")
lgrid[5,] <- c("h", "j", "f", "f", "k", "h", "g", "m")
lgrid[6,] <- c("k", "y", "e", "x", "x", "g", "k", "i")
lgrid[7,] <- c("l", "q", "e", "q", "f", "u", "e", "b")
lgrid[8,] <- c("l", "s", "d", "h", "i", "k", "y", "n")
```

Next, we make an R function which takes the current square(row,column) as an argument and this function then returns the square(row,column) to which the token will move following the rules given. We employ different outputs for different kind of squares. If the square is on edge i.e. either the row number or column number is equal to 1 or 8, then we randomly choose a square for next move. If the square is not on the edge, we simply move the token to the adjacent square. A working example is, if the current square is (4,6), the possible values for row to take can be 3,4 or 5. Although column can also take value 5,6 or 7, we have to prevent the occurrence of square (4,6) again. So we make a condition that if randomly same row is chosen, then the column cannot be the same one as old one.

```
move_token<- function(curr_sq){
  row_num=curr_sq[1]
  col_num=curr_sq[2]
  if(any(row_num==c(1,8))|any(col_num==c(1,8))) {  # If on the edge, select random square
    row_num_new<- sample(1:8, size = 1)
    col_num_new<- sample(1:8, size = 1)
    }
  else{  # If not edge
    row_num_new<-sample(c(row_num-1,row_num,row_num+1),size=1)
    if (row_num_new==row_num) # To prevent landing on same square
      col_num_new<-sample(c(col_num-1,col_num+1),size=1)
    else{
      col_num_new<-sample(c(col_num-1,col_num,col_num+1),size=1)}
  }
  next_sq<-c(row_num_new,col_num_new)
  return(next_sq)
}
```

Part 2

(a)

The following rules were created to decide whether to add the token to collection or not. The number of occurrences of a letter in the grid was found using table(lgrid)[letter]. The rules are as follows- i) Collect the letter from the square where token lands on, i.e. we automatically add the first letter from first move itself unless it is a green square. ii) For the second letter in the collection, we employ 3 conditions for addition. If the current token letter is already in the collection(same as first letter), we add the current token to selection. Otherwise, depending on number of moves, we select the second token letter. If total moves are less than 5, we look for the letters which have high enough chance of occurrence(2 times or more). Finally, if we don't find any token letter matching the above 2 conditions, we select a letter which has above average occurrence(3 or more). We choose more frequent letter so that we have a higher chance of completion of palindrome at the expense of few more moves. iii) For the third letter, if the current letter is already in the collection, we simply add it to the collection irrespective of its occurrence in the letter grid. Otherwise, we employ the conditinality that we add the letter to collection if the current square's letter has more than 2 times occurrence(2 or more times) in the grid. This will help us find other matching letters very easily. iv) For the fourth letter, if there is repetition of letter already in the collection, we simply add the letter from the present square. Otherwise, if all 3 letter are different in the collection, we only add the letter that is same as any of the 3 in the collection. v) For the final letter in the collection, we employ multiple conditions to complete the palindrome. If the collection contains only 2 different letters with twice occurrence each, we simply add the current letter. If a letter occurs 3 times in the collection and 1 letter is different, we try to find any of these 2 letters to complete off the palindrome. If the 4th letter collected was not the same as first 3 letters collected(case of f,f,h,k or cases where letters were collected twice in first 3 item collection), we search of letter which is same as the letter whose occurrence is only once in the collection(like h or k in the case of f,f,h,k). And final rule is, if first 3 letters were all different, then the 4th one is similar to any of 1st,2nd or 3rd. Thus we want to make sure we don't collect the 5th letter same as the 4th one.

(b)

The criteria for letters to form a palindrome is that they should read the same from the front same as from the end. But if we ignore the order of letters and look at occurrences, it can be observed that a 5 letter palindrome has occurrences either in format of (2,2,1) or (3,2) or (4,1). The rules written in 1st part of this question follow all these mentioned possibilities for the completion of a palindrome and with the least number of moves possible. Also, as per the rules of the question, the collection is updated when the token lands on a green square. Although we can improve the chances of collection of letters by giving a condition for first element too, and make sure the palindrome is formed, which it in turn increases the complexity.

Part 3

We now write an R code which implements the above defined rules. First we make a function which checks if current square is green or white. If the square is green, we update the collection depending on the value of p. If it is a green square, we just update the collection and move to the next square instead of checking the letter present in the green square. Then we write the main function which takes in a string for row and column as an argument along with the p value and returns the number of moves. We include a 3rd argument which decides whether to print the collection and number of moves. We provide 'f' to avoid print by default and 't' if we want to print.

```
check_green_sq<- function(sq_num){ # Check for Green square
  row_num=sq_num[1]
  col_num=sq_num[2]
```

```
  if (any(row_num==c(2,3,6,7)) & (abs(col_num-row_num)==4)) #(2,6),(3,7),(6,2),(7,3)
    {return('yes')}
  else {return('no')}
}

output_of_green_sq<-function(square_num,p,collection){
  row_num=square_num[1]
  col_num=square_num[2]
  choice=sample(c(1,2),size=1,prob=c(p,1-p))
  if (choice==1){ # If choice 1 with probability of p
    collection=c('f','f','h','k')} # new collection
  else{           # If choice 2 with probability of 1-p
    collection<-collection[collection!=lgrid[row_num,col_num]]}
  return(collection)
}

num_moves_for_palindrome<- function(start_sq,probability,ToPrint='f'){
  row_start=unlist(strsplit(start_sq,""))[1]
  col_num_start=strtoi(unlist(strsplit(start_sq,""))[2])
  num_moves=1      # We start with 1 because we already selected the square to start on
  p=probability
  collection<-list()     # Initialise collection with empty list
  row_num=which(letters == tolower(row_start))     # Convert from alphabet to integer
  col_num=col_num_start
  curr_sq<-c(row_num,col_num)
  if (check_green_sq(curr_sq)=='yes') { # Green square
    collection<- output_of_green_sq(curr_sq,p,collection)}
  else{collection<-c(collection,lgrid[row_num,col_num])} # First letter is directly added

  # Main loop which forms a palindrome
  while(length(collection)<5){
    curr_sq<-c(row_num,col_num)
    next_sq=move_token(curr_sq)  # returns a list of (row,column)
    num_moves=num_moves+1
    row_num=next_sq[1]
    col_num=next_sq[2]
    alphabet<-lgrid[row_num,col_num] # letter from the square the token is on
    collection<-unlist(collection) #unlist stops multiple list creation within a list
    if (check_green_sq(curr_sq)=='yes') { # Green square
      collection<- output_of_green_sq(curr_sq,p,collection)}

    else { # If not green square

    if (length(collection)==1){ # For 2nd letter
      if(is.element(alphabet,collection)) {
        collection<-c(collection,alphabet)} # if element already present
      else if(num_moves<=5 & table(lgrid)[alphabet] >= 2) {
        collection<-c(collection,alphabet)}
      else if(num_moves>5 & table(lgrid)[alphabet] >= 3) {
        collection<-c(collection,alphabet)}
      }
    else if(length(collection)==2){ # For 3rd letter
      if(is.element(alphabet,collection)) {
```

```
        collection<-c(collection,alphabet)} # if element already present
      else if(table(lgrid)[alphabet] >= 2) {
        collection<-c(collection,alphabet)}
      }
    else if(length(collection)==3) { # For 4th letter
      if(any(table(collection)>=2)) { # cases like (b,c,b)
        collection<-c(collection,alphabet)}
      else if(is.element(alphabet,collection)){#4th letter is subset of first 3 letters
        collection<-c(collection,alphabet)}
      }
    else if (length(collection)==4) { # For 5th letter
      if (!is.element(collection[4],collection[1:3])){ # cases like (f,f,h,k)
        if (is.element(alphabet,collection) & (table(collection)[alphabet]!=c(2))) {
          collection<-c(collection,alphabet)
        }}
      else if (all(table(collection)>=c(2))) { # cases like (g,u,u,g),(g,g,g,g)
        collection<-c(collection,alphabet)
      }
      else if (is.element(alphabet,collection) & alphabet!=collection[4]) {
        collection<-c(collection,alphabet)}} # cases like (a,b,c,a)
    }
  }
  if (ToPrint=='t') {cat("The letters in collection are-",collection,
                         ",and number of moves is",num_moves,"\n")}
  return(num_moves)
}
```

We can check the working of our function by providing randomly chosen non-edge square and probability

```
rand_row<-sample(LETTERS[2:7],size=1)
rand_col<-sample(2:7,size=1)
rand_prob<- sample(seq(0,1,by=0.01),size = 1)
rand_sq<- paste(rand_row, rand_col, sep = "")
cat("starting random square is-",rand_sq,"and the random probability is-",rand_prob,"\n")
```

```
## starting random square is- E2 and the random probability is- 0.49
```

```
moves<- num_moves_for_palindrome(rand_sq,rand_prob,ToPrint = 't')
```

```
## The letters in collection are- j h g j h ,and number of moves is 14
```

Part 4

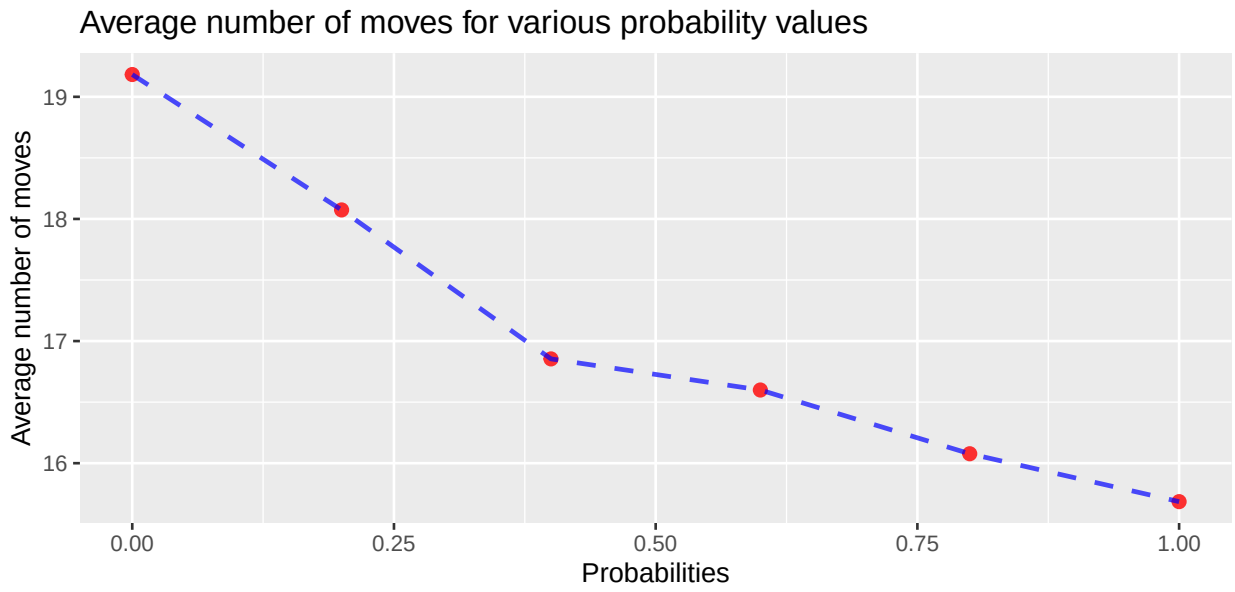
To check the dependency of number of moves on values of p, we find the average of 10000 simulations of each probability case in the list created below. We then plot a line graph to check for relation between p and number of moves.

```
set.seed(210116270)
prob_list<-seq(0,1,by=0.2)
mean_moves<-rep(NA,length(prob_list))  # empty list to append number of moves
```

```
pos=0
for (prob in prob_list) {
  pos=pos+1
  mean_moves[pos]<-mean(replicate(10000,num_moves_for_palindrome('D4',prob)))
  cat("For probability=",prob,",and D4 start, average number of moves=",mean_moves[pos],
      "\n")
}
```

```
## For probability= 0 ,and D4 start, average number of moves= 19.1825
## For probability= 0.2 ,and D4 start, average number of moves= 18.0738
## For probability= 0.4 ,and D4 start, average number of moves= 16.854
## For probability= 0.6 ,and D4 start, average number of moves= 16.5993
## For probability= 0.8 ,and D4 start, average number of moves= 16.0775
## For probability= 1 ,and D4 start, average number of moves= 15.6853
```

```
data<- data.frame(prob_list,mean_moves)
ggplot(data, aes(x=prob_list, y=mean_moves)) +
  geom_point(color="red",size=2.2, alpha=0.8)+
  geom_line(color="blue", size=0.9, alpha=0.7, linetype=2) + xlab("Probabilities")+
  ylab("Average number of moves") +
  ggtitle("Average number of moves for various probability values")
```



We can see from the line plot that number of moves needed to form a palindrome reduces as we increase the p value. It might be because when the probability is high, the higher is the probability of the event that the letters in the collection set are replaced by the letters F, F, H and K. Since we have only one letter left(either a H or K), the number of movements required will be relatively less.

Part 5

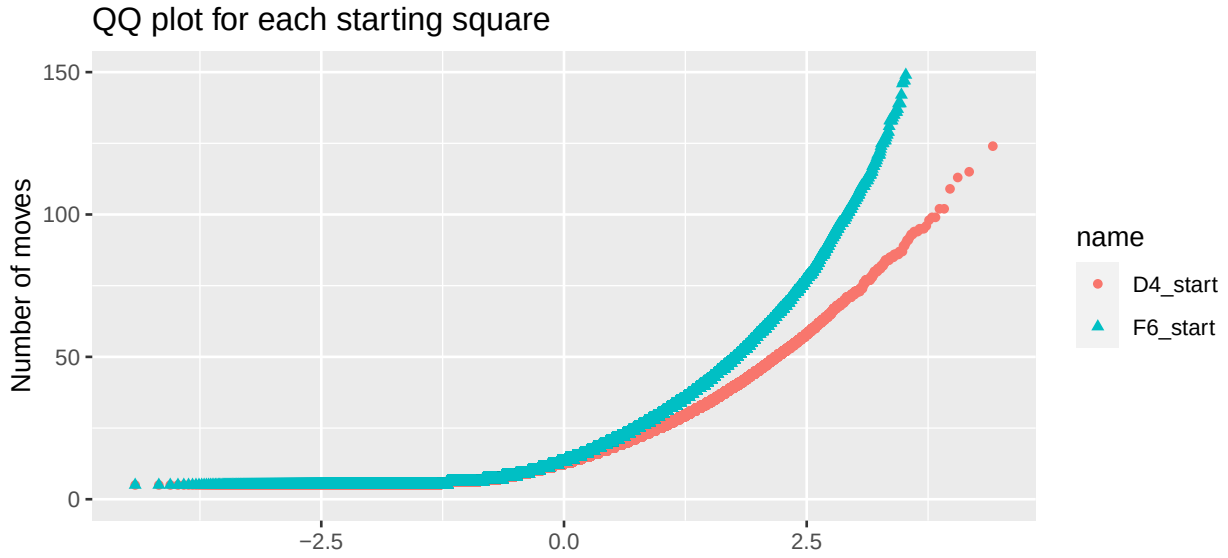
Now we are comparing if the distributions of the number of moves required to complete the game are identical for the starting points D4 with p=0.95 and F6 with p=0.05. After simulating the function with different starting points and p values 100000 times. Next, we use descr() from sjmisc package to find various statistical

metrics for the two distributions.The mterics we are interested in are- mean, standard deviation, median, trimmed mean, interquartile range and skewness. We also plot a Quantile-Quantile Plot to see where the number of moves lies for each quantile.

```
set.seed(210116270)
D4_start<- replicate(100000,num_moves_for_palindrome('D4',probability=0.95))
F6_start<- replicate(100000,num_moves_for_palindrome('F6',probability=0.05))
temp_data<- data.frame(D4_start,F6_start)
metrics<-c("n","mean", "sd", "md", "trimmed", "iqr", "skew")
descr(temp_data,show = metrics)
```

```
##
## ## Basic descriptive statistics
##
##       var      n  mean    sd md trimmed iqr skew
##  D4_start 100000 15.72 10.87 13   13.99  14 1.74
##  F6_start 100000 17.90 14.59 13   15.35  17 2.28
```

```
long_data<-pivot_longer(temp_data,cols = everything())
qplot(sample = value, data = long_data, color=name, shape=name)+ylim(0,150)+
  labs(title="QQ plot for each starting square",y = "Number of moves")
```



From the metrics calculated and QQ plot above, it can be seen that both the distribution have identical median and they have same distribution till 80th quartile. The outliers are high in case of second distribution and hence the standard deviation and skew are high compared to first case.

We now perform a z-test to check if both the distributions have same mean. The sigma.x and sigma.y values were calculated by simulating 1000 times. The null hypothesis is H0: The difference(mu) of average number of moves from the squares D4 and F6 is 0 i.e. E(D4)-E(F6)=0 . The alternative hypothesis is HA: The means are not identical i.e. E(D4) $\neq$ E(F6).

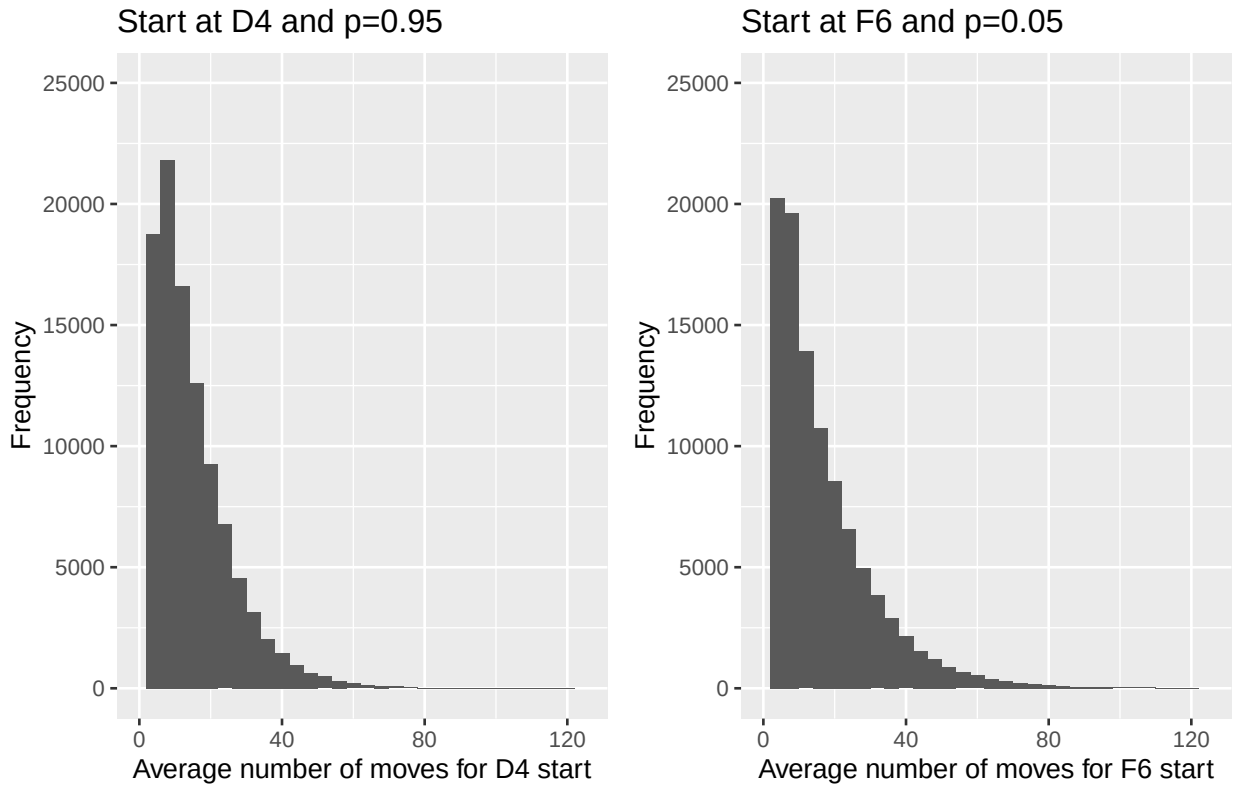
```
# We perform a z-test using above hypothesis
z.test(x=D4_start, sigma.x=11,y=F6_start, sigma.y=27,mu=0,
       alternative = "two.sided",conf.level = 0.95)
```

```
##
##  Two-sample z-Test
##
## data:  D4_start and F6_start
## z = -23.613, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.35768 -1.99628
## sample estimates:
## mean of x mean of y
##  15.72011  17.89709
```

As we can see from above, the hypothesis is rejected because z-statistic does not lie in the 95% confidence interval.

We now try to visualise the histogram of the two distribution side by side using grid.arrange function.

```
plot1 <- ggplot() +geom_histogram(data =temp_data, aes(x= D4_start),binwidth = 4)+
  ylim(0,25000)+xlab("Average number of moves for D4 start")+ylab("Frequency")+
  ggtitle("Start at D4 and p=0.95")+xlim(0,125)
plot2 <- ggplot() +geom_histogram(data =temp_data, aes(x= F6_start),binwidth = 4)+
  ylim(0,25000)+xlab("Average number of moves for F6 start")+ylab("Frequency")+
  ggtitle("Start at F6 and p=0.05")+xlim(0,125)
grid.arrange(plot1, plot2, ncol=2)
```



Although both the distributions look similar, there are many outliers in second case(F6 start), and hence the distribution is more right skewed. If we assess the evidence of this case using simple statistics, the median and 25-75 quartile range of both the distribution indicate that the distribution might be same. But looking at the plots and z-test, we can conclude with more than 95% confidence that the average number of moves might not be identical to each other.

Part 6

We are now interested in whether the mean number of moves required to complete the game are the same for the two starting squares A and B. We perform a Welch Two Sample t-test to check the same. We perform a t-test and not a z-test because the number of samples is less than 30. The null hypothesis is H0: Expectation of (XA) = Expectation of (XB). The alternative hypothesis is HA: The expectations are different i.e., E(XA) $\neq$ E(XB).

```
moves_sq_A <- c(25,13,16,24,11,12,24,26,15,19,34)
moves_sq_B <- c(35,41,23,26,18,15,33,42,18,47,21,26)
t.test(x=moves_sq_A, y=moves_sq_B, alternative="two.sided", paired=F, conf.level= 0.95)
```

```
##
##  Welch Two Sample t-test
##
## data:  moves_sq_A and moves_sq_B
## t = -2.3463, df = 19.468, p-value = 0.02968
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -16.7146664  -0.9671518
## sample estimates:
## mean of x mean of y
##  19.90909  28.75000
```

We can see from the t-test above, the t value(-2.34) lies in the 95% confidence interval(-16.71 to -0.96), and hence we can't reject the null hypothesis. We now try with 89.9% confidence interval to check if we can reject the null hypothesis.

```
t.test(x=moves_sq_A, y=moves_sq_B, alternative="two.sided", paired=F, conf.level= 0.899)
```

```
##
##  Welch Two Sample t-test
##
## data:  moves_sq_A and moves_sq_B
## t = -2.3463, df = 19.468, p-value = 0.02968
## alternative hypothesis: true difference in means is not equal to 0
## 89.9 percent confidence interval:
##  -15.327835  -2.353983
## sample estimates:
## mean of x mean of y
##  19.90909  28.75000
```

This time, the t value does not lie in the confidence interval(-15.32 to -2.35). Hence we can conclude with 89.9% level of confidence that the expectation of number of moves starting at square A is not equal to the expectation of number of moves starting at square B.