CSE 4304: DATA STRUCTURES AND ALGORITHMS

Instructor: Prof. Ashwin Ganesan

Lab report

Name: Dhara Awasthi

Registration ID: 4973

Title of Lab: Debugging_Programming

Date: 18 June'2022

Question-1:

Problem description

The problem aims to check if the binary string entered is an even palindrome or not. This means that the string should be of even length and needs to be a palindrome.

Algorithm description

The logic is to first check if the string is of even length, by dividing it by 2 and checking the remainder. Next, if the string is of even length we will find out the reverse of the string and check whether the reverse of the string is equal to the string itself.

Program:

```
def even_palindrome(x):
 if len(x)%2!=0:
   return False
 else:
    rev = (x)[::-1]
    if rev!= x:
      return False
    return True
```

Sample Outputs

1) Input: 1010

Output: False

```
False
```

2) Input: 10101

Output: False

```
False
```

3) Input: 11

Output: True

```
True
```

Analysis and conclusion:

The main objective of doing this experiment was to check if the given binary string is an even palindrome or not. Checking a palindrome can be done in a couple of ways like checking the reverse, iterating over the first half of the string and comparing it with the second half. I took a few strings which were palindrome but not even, even but not palindrome etc.and checked if they are giving any errors.The program can handle all kinds of strings.

Question 2: Valid sequence (i's and e's)

Problem statement:

The problem aims to find out whether the given sequence is valid or not if a prefix of a string has number of i's equal or greater than number of e's.

Algorithm description:

The algorithm basically aims to count the number of i's and e's and then construct the sequence using the count of i's and e's i.e multiplying "i" with count of i's and concatenating this with the product of "e" and count of e's. In pseudocode : i*count_i+e*count_e
Final step is to compare this constructed sequence with the original sequence of string.

Program:

```
count_i=0
count_e=0
def checkvalid(a2):
  global count_e
  global count_i
  if a2==" " :
   print("Empty string")
  else:
   for j in range(0,len(a2)):
     if a2[j] =="i":
      count_i=count_i+1
     elif a2[j] =="e":
      count_e=count_e+1
   if count_i>=count_e:
    print("Valid")
   else:
    print("Invalid")
```

Sample outputs:

1) Input: iie

Output: Valid

```
Valid
```

2) Input: iieee

Output: Invalid

```
Invalid
```

3) Input: iiieeeei

Output: Invalid

```
Invalid
```

Question 3: Valid strings (0^n 1^n 2^n)

Problem statement:

The aim of the problem is to determine if the string is valid or not if the string contains equal number of 0s followed by equal number of 1s and equal number of 2s in the format (0^n 1^n 2^n).

Algorithm description:

The algorithmic approach is to count the number of 0s,1s and 2s in the given string. Next step is to construct the sequence using concatenation of number of zeroes, ones and twos in the respective order (Pseudocode form: str(0)*count_zero + str(1)*count_one+ str(2)*count_two). The final step is to compare this constructed sequence using counts with the given string to check if they are equal.

Program:

```
count_zero=0
count_one=0
count_two=0
def valid_string(y):
 global count_zero
 global count_two
 global count_one
 if y==" ":
   print("Empty string")
 elif y[0]=="0":
    for i in range(0,len(y)):
     if y[i]=="0":
      count_zero = count_zero+1
     elif y[i]=="1":
      count_one= count_one+1
     elif y[i]=="2":
      count_two= count_two+1
    x = "0"*count_zero + "1"*count_one + "2"*count_two
    print("count of 0s1s2s in given string: ",x)
    if y == x:
      print("Valid string")
    else:
```

```
print("Invalid string")
```

Sample outputs:

1) Input: 001122

Output: Valid String

```
Valid string
```

2) Input: 012012

Output: Invalid string

```
Invalid string
```

3) Input: 0

Output: Invalid

```
Invalid string
```

Analysis and conclusion:

I learnt about dealing with lists containing strings and integers and how to concatenate strings with integers. I spent a lot of time getting my code to work for all kinds of inputs.

Question 4: Longest run

Problem statement:

The problem aims to find the subsequence of continuously increasing numbers and find out the length of the subsequence which is having the longest run.

Algorithm:

The algorithmic approach is to first check adjacent elements starting from the first element. If the (k+1)th element is greater or equal to the kth element, the length of the run or running sequence should be summed or incremented by 1 until the condition is satisfied.

As and when (k+1)th element becomes less than kth element, the longest run should be equal to zero and length of the running should start from 1. The length of the longest running sequence will be stored and will be compared with the length of the next running sequence. As and when the length of the consecutive running sequence will be greater than the value of the longest run, it will be replaced with the length of the running sequence.

Program:

```
longest_seq=0
run_seq= 1
def longestRun(x):
  global longest_seq
  global run_seq
  for j in range(1,len(x)):
     if (x[j]>=x[j-1]):
      run_seq= run_seq+1
      if run_seq>longest_seq:
       longest_seq= run_seq
     else:
       run_seq=1
print("longest run value is",longest_seq)
```

Sample outputs:

1) Input: 468345772

Output: Longest run value is 5

```
longest run value is 5
```

2) Input: 236345772

Output: Longest run value is 5

```
longest run value is 5
```

3) Input: 13423456789

Output: Longest run value is 8

```
longest run value is 8
```

Analysis and conclusion:

This was a little tricky problem. I got to learn how to write and modify the code based on the input. At first my code was only working for very simple input and hence I was trying to handle exceptional cases using try blocks, but later on I used Python tutor to visualize the output of each step of code while iterating over the loop and analyzing where my code was going wrong. It helped me to improvise my code without any limitations. I used different kinds of sequences which had multiple subsequences of different runs. This helped me to check if my code is working for all kinds of sequences.

Acknowledgements:

https://pythontutor.com/visualize.html (For visualizing code step by step)

```
#Question 1: Even palindromes
def even_palindrome(x):
 if len(x)%2!=0:
   return False
 else:
    rev = (x)[::-1]
    if rev!= x:
      return False
    return True
even_palindrome("11")
```

```
 True
```

```
#Question 2: Valid sequence
count_i=0
count_e=0
def checkvalid(a2):
  global count_e
  global count_i
  if a2==" " :
   print("Empty string")
  else:
   for j in range(0,len(a2)):
     if a2[j] =="i":
      count_i=count_i+1
     elif a2[j] =="e":
      count_e=count_e+1
   if count_i>=count_e:
    print("Valid")
   else:
    print("Invalid")
checkvalid("iie")
```

```
Valid
```

```
#Question 3: Valid string
count_zero=0
count_one=0
count_two=0


def valid_string(y):
 global count_zero
 global count_two
 global count_one
 if y==" ":
   print("Empty string")
 elif y[0]=="0":
    for i in range(0,len(y)):
```

```
        if y[i]=="0":
         count_zero = count_zero+1
        elif y[i]=="1":
         count_one= count_one+1
        elif y[i]=="2":
         count_two= count_two+1
       x = "0"*count_zero + "1"*count_one + "2"*count_two
       if y == x:
         print("Valid string")
       else:
         print("Invalid string")
valid_string("0123")
```

```
     Invalid string
```

```
#Question 4: Longest run
longest_seq=0
run_seq= 1
def longestRun(x):
  global longest_seq
  global run_seq
  for j in range(1,len(x)):
     if (x[j]>=x[j-1]):
      run_seq= run_seq+1
      if run_seq>longest_seq:
       longest_seq= run_seq
     else:
       run_seq=1
longestRun("468345772")
print("longest run value is",longest_seq)
```

```
     longest run value is 5
```

✓ 0s completed at 9:55 PM