SYNOPSYS
INTENSIVE TRAINING PROGRAM 2023

EXERCISE REPORT

AWK - SCRIPTING LANGUAGE

Trainee: Nguyễn Thành Quí
Group: GROOT

Ho Chi Minh City, 6/2023

Mục lục

1 Exercise 1:

Mr.Quang has sent us a solution, so I used this to solve the remaining exercises.

2 Exercise 2

Complexity: O(n)

Short explanation: The script starts by setting up the necessary variables and removing any existing output files.

It then uses the first awk command to read input1 and input2 files simultaneously. Depending on the specified mode (line or column), it extracts the desired lines or columns from both files and stores them in separate arrays (line1 and line2).

Next, the extracted lines or columns from both files are combined and stored in the 02_full.txt file using the second awk command. Simultaneously, the common elements between the two files are identified and stored in the 02_same.txt file. The count of common elements is also tracked.

Finally, the third awk command compares the elements in 02_full.txt with the common elements stored in 02_same.txt. It identifies the different elements and stores them in the 02_diff.txt file. The count of different elements is tracked as well.

The algorithm essentially involves extracting the desired lines or columns from the input files, combining the extracted elements, identifying the common elements, and finding the different elements. The counts of common and different elements are maintained throughout the process.

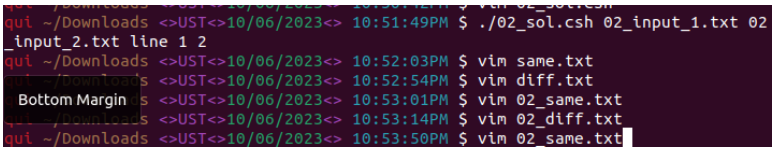
Source code:

```
#!/bin/csh -f

set input1 = $1
set input2 = $2
set mode = $3
set number1 = $4
set number2 = $5

rm -rf 02_com1.txt 02_com2.txt 02_same.txt 02_diff.txt 02_full.txt

# sort column and lines for 02_input1.txt and 02_input2.txt

awk 'BEGIN{\
   column = "column";\
   line = "line";\
}\
{\
   if ('$mode' == line){\
      if (FNR == NR) {\
         if (NR == '$number1') {\
            for (i = 1; i <= NF; i++) {\
```

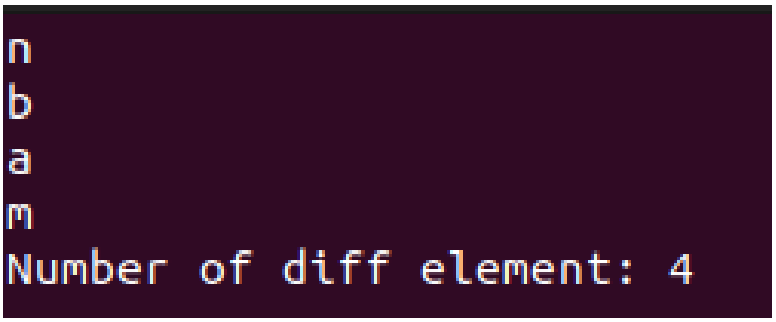
```
                line1[i] = $i;\
                count1++;\
            }\
         }\
      } else {\
         if (FNR == '$number2') {\
            for (i = 1; i <= NF; i++) {\
               line2[i] = $i;\
               count2++;\
            }\
         }\
      }\
   } else if ($mode == column){\
      if (FNR == NR) {\
         line1[FNR] = $'$number1';\
         count1++;\
      } else {\
         line2[FNR] = $'$number2';\
         count2++;\
      }\
   }\
}\
END {\
   for (i = 1; i <= count1; i++) {\
      print line1[i] >> "02_com1.txt";\
   }\
   for (i = 1; i <= count2; i++) {\
      print line2[i] >> "02_com2.txt";\
   }\
}' $input1 $input2

# combine two files

awk 'BEGIN{ \
   i = 1; \
   n = 1; \
   while ((getline < "02_com2.txt") > 0 ) {\
      a[i] = $1;\
      i++;\
      print $1 >> "02_full.txt";\
   }\
   close ("02_com2.txt");\
   while ((getline < "02_com1.txt") > 0 ) {\
      print $1 >> "02_full.txt";\
   }\
   close ("02_com1.txt");\
}\
{\
```

```
   ipt = $1;\
   for (k = 1; k <= i; k ++) {\
      if (ipt == a[k]) {\
         count_same++;\
         print ipt >> "02_same.txt";\
      }\
   }\
}\
END {\
   print "Number of same element:", count_same >> "02_same.txt";\
}' 02_com1.txt

# compare

awk 'BEGIN {\
   while ((getline < "02_same.txt") > 0 ) {\
      n++;\
      a[n] = $1;\
   }\
   close ("02_same.txt");\
}\
{\
   b = $1;\
   for (k = 1; k <= n; k++) {\
      if (a[k] == b) {\
         k = n + 1;\
      } else {\
         if (k == n) {\
            count_diff++;\
            print b >> "02_diff.txt";\
         }\
      }\
   }\
}\
END {\
   print "Number of diff element:", count_diff >> "02_diff.txt";\
}' 02_full.txt
```

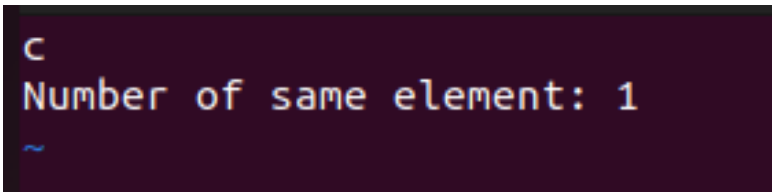
Output:



Hình 1: *Output for Exercise 2*



Hình 2: *Output for Exercise 2*



Hình 3: *Output for Exercise 2*

3 Exercise 3

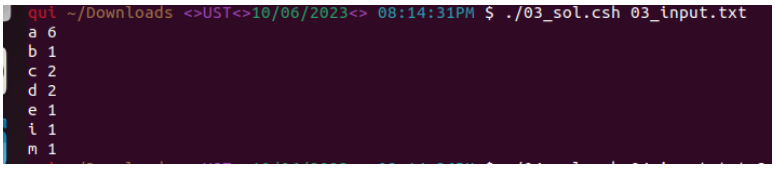
Complexity: O(n)

Short explanation: Using the special array structure of AWK, which is quite different from the array structre of C language to make the code simpler. For example the syntax arr["a"] = 1 is not allowed in C. Using the built-in function asorti() to sort the array before printing to the terminal.

Source code:

```
#!/bin/csh -f

set input1 = $1

awk 'BEGIN {\
}\
{\
        count[$0]++\
}\
END {\
  n = asorti(count, sorted)\
        for (i = 1; i <= n; i++) {\
            line = sorted[i]\
            print line, count[line]\
        }\
    }' $input1
}
```

Output:



Hình 4: *Output for Exercise 3*

4 Exercise 4

Complexity: $O(n^2)$

Short explanation: Using the concept of C array for easier traversing. The line $arr[i] = substr(0,\ 0,\ 2 * 'input2' - 1)$ is to retrieve the desired string, for example if the input is 2, we will have the substring starting from position 0 and ending at postion 3.

n = asort(arr): This line sorts the array arr in ascending order and assigns the number of elements in the sorted array to the variable n.

temp = 0: We initialize a temporary variable temp to zero.

The following block of code prints the elements of the array arr without any duplicates. It loops through each element in the array and prints it.

The next block of code calculates the frequency of each unique element in the array. It uses nested loops: an outer loop iterates over each element in the array, and an inner loop compares the current element with the remaining elements. If a duplicate is found, the frequency counter fre is incremented and the index of the last occurrence of the duplicate is stored in the variable temp.

After finding the frequency of a particular element, it is printed along with the element itself using print arr[i]" " fre. The frequency counter fre is then reset to zero.

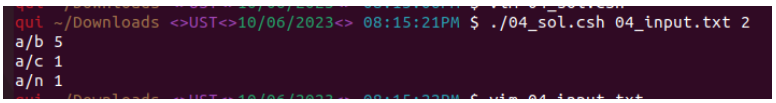
The variable i is updated to the last occurrence of the current element, which effectively skips the iterations for the duplicates in the next iteration of the outer loop.

Source code:

```
#!/bin/csh -f

set input1 = $1
set input2 = $2

awk 'BEGIN  {\
        i = 1 \
        lineCount = 0 \
        fre = 0 \
  '$input2' \
} \
{ \
        lineCount++  \
        arr[i] = substr($0, 0, 2 * '$input2' - 1) \
        i++ \
} \
```

```
END { \
        n = asort(arr) \
        temp = 0 \
        for (i = 1; i <= lineCount; i++) { \
                #print arr[i] \
        } \
        for (i = 1; i <= lineCount; i++) { \
                for (j = i; j <= lineCount; j++) { \
                        if (arr[j] == arr[i]) { \
                                fre++; \
                                temp = j \
                        } \
                } \
                print arr[i]" " fre \
                fre = 0 \
                i = temp \
        } \
}' $input1
```

Output:



Hình 5: *Output for Exercise 4*

5 Exercise 5

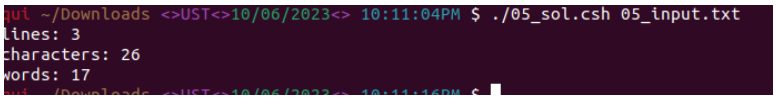
Complexity: O(n)

Source code:

```
#!/bin/csh -f
set input1 = $1
set input2 = $2
set wordCount = $0
awk 'BEGIN {\
lineCount = 0 \
charCount = 0 \
wordCount = 0 \
} \
{ \
lineCount = lineCount + 1 \
blankCount = NF \
wordCount += blankCount \
for (i = 1; i <= length($0); i++) { \
if (substr($0, i, 1) == " ") \
continue \
```

```
else \
charCount++ \
} \
} \
END { \
print lineCount \
print charCount \
print wordCount \
}' $input1
```

Output:



Hình 6: *Output for Exercise 5*

6 Exercise 6

Complexity: O(n)

Short explanation: The whole idea of my solution is retrieving the desired input from input file to an array and then printing it following the format. The variable "command" is used to store a command that retrieves specific lines from the file "06_input.txt" based on a pattern.

command1 = "cat 06_input.txt | grep -A 1 StartP | grep -v StartP | grep /": retrieves the line after the StartP in 06_input.txt, the same idea for EndP.

```
Path 1:
StartP: a/b/c/d/
e/f/g/h
EndP: m/n/q/q/
r/s/t
Slack: -0.6
```

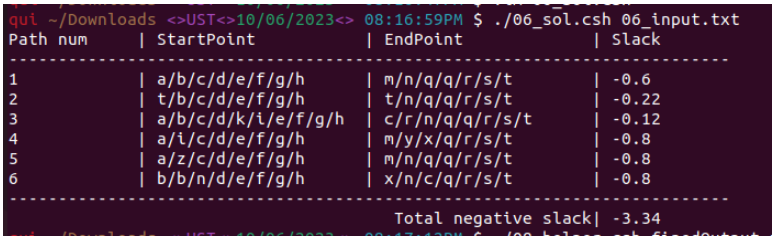
Source code:

```
#!/bin/csh -f

set input1 = $1

awk 'BEGIN {\
    # Decoration lines\
    printf "%-12s | %-20s | %-20s | %-10s \n", "Path num", "StartPoint", "EndPoint", "Slack" \
    printf "%s%s%s%s\n", "-----------------------", "--------------------", "--------------------", "
\
    i = 1 \
    command = "cat 06_input.txt | grep Path" \
```

```
    while (command | getline > 0) { \
  num = substr($2, 0, length($2) - 1) \
        path[i] = num \
        i++ \
    } \
    i = 1 \
    command = "cat 06_input.txt | grep StartP" \
    command1 = "cat 06_input.txt | grep -A 1 StartP | grep -v StartP | grep /" \
    while (command | getline > 0) { \
        startP[i] = $2 \
        i++ \
    } \
    i = 1 \
    while (command1 | getline > 0) { \
        temp = startP[i] $0 \
        startP[i] = temp\
        i++ \
    } \
    i = 1 \
    command = "cat 06_input.txt | grep EndP" \
    command1 = "cat 06_input.txt | grep -A 1 EndP | grep -v EndP | grep /" \
    while (command | getline > 0) { \
        endP[i] = $2 \
        i++ \
    } \
    i = 1 \
    while (command1 | getline > 0) { \
        temp = endP[i] $0 \
        endP[i] = temp \
        i++ \
    } \
    i = 1 \
    totalSlack = 0 \
    command = "cat 06_input.txt | grep Slack" \
    while (command | getline > 0) { \
        slack[i] = $2 \
        totalSlack += slack[i] \
        i++ \
    } \
    n = i \
    for (j = 1; j < n; j++) { \
        printf "%-12s | %-20s | %-20s | %-10s \n", path[j], startP[j], endP[j], slack[j] \
    } \
    printf "%s%s%s%s\n", "-----------------------", "--------------------", "--------------------", "
    printf "%59s| %-10s \n", "Total negative slack", totalSlack \
}\
{\
}\
```

```
END {\
}' $input1
```

Output:



```
qui ~/Downloads <>UST<>10/06/2023<> 08:16:59PM $ ./06_sol.csh 06_input.txt
Path num     | StartPoint            | EndPoint              | Slack
-----------------------------------------------------------------------
1            | a/b/c/d/e/f/g/h       | m/n/q/q/r/s/t         | -0.6
2            | t/b/c/d/e/f/g/h       | t/n/q/q/r/s/t         | -0.22
3            | a/b/c/d/k/i/e/f/g/h   | c/r/n/q/q/r/s/t       | -0.12
4            | a/i/c/d/e/f/g/h       | m/y/x/q/r/s/t         | -0.8
5            | a/z/c/d/e/f/g/h       | m/n/q/q/r/s/t         | -0.8
6            | b/b/n/d/e/f/g/h       | x/n/c/q/r/s/t         | -0.8
-----------------------------------------------------------------------
                                       Total negative slack| -3.34
```

Hình 7: *Output for Exercise 6*

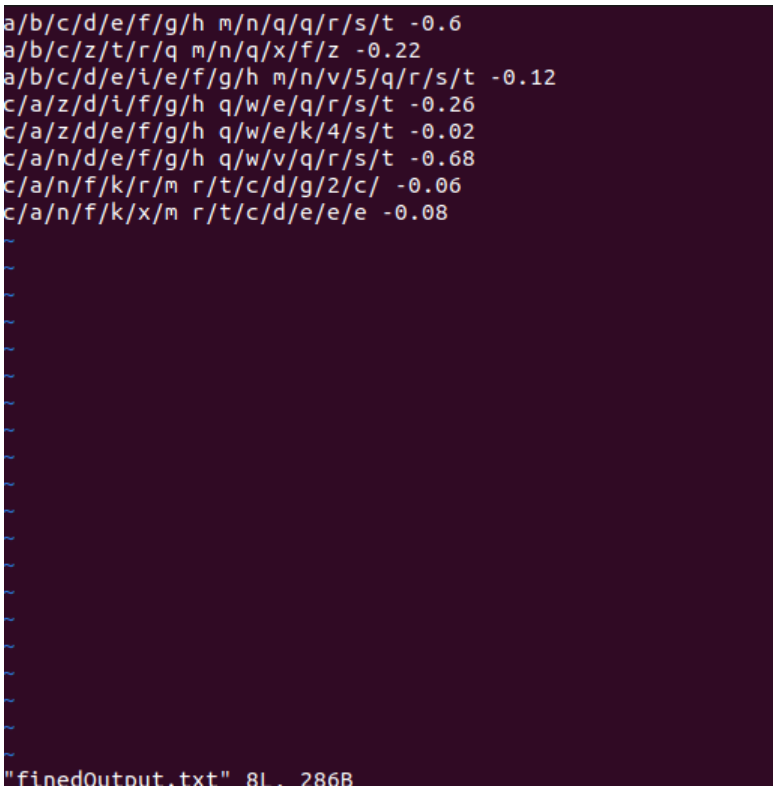
7 Exercise 7

My group took the main responsibility to solve this exercise. I am still working on it on my own but there are some bugs need fixing, so I cannot put it here.

8 Exercise 8

Complexity: $O(n^2)$

Short explanation: The 08_sol.csh file will retrieving the desired format from the input file to fined-Output.txt like below:



Hình 8: *Content in finedOutput.txt*

The 08_helper.csh will keep on using the finedOutput.txt as input file

Let's focus more on the complicated for loop in 08_helper.csh file:

The outer for loop iterates over the variable i from 1 to lineCount, which represents the total number of extracted substrings.

Within the outer for loop, there is an inner for loop that iterates over the variable j from i to lineCount. This nested loop is used to find all occurrences of the same extracted substring.

The condition if (extract[j] == extract[i]) checks if the current substring at index j is the same as the substring at index i. If they match, it means we have found another occurrence of the same substring.

Inside the if condition, fre (frequency) is incremented to keep track of the number of occurrences of the substring. The total is updated by adding the value of slack[j] for each occurrence.

The nested if condition if (min + 0 > slack[j] + 0) compares the value of min (initialized as 999) with the current slack[j] value. If the slack[j] value is smaller, it updates the min value to the smaller value.

The variable temp is set to the index j where the last occurrence of the substring was found. This is used to remember the index for later printing.

After the inner loop completes, the values of min, total, fre, and extract[temp] are printed in a formatted manner using the printf statement.

The variables sumT and count are updated by adding the respective values of total and fre for each substring.

The condition if (slack[temp] + 0 < sumMin + 0) compares the value of slack[temp] with the current

value of sumMin. If slack[temp] is smaller, it updates sumMin with that value.

Finally, the variables min, fre, total are reset to their initial values, and the i variable is updated with the value of temp to continue processing the next unique substring.

Source code:

08_sol.csh

```
#!/bin/csh -f


set input1 = $1
set input2 = $2
rm -rf finedOutput.txt
awk 'BEGIN {\
command = "cat 08_input.txt | grep /" \
i = 1 \
while (command | getline > 0) { \
  startP[i] = $3 \
  endP[i] = $5 \
  slack[i] = $ 7 \
  print startP[i]" "endP[i] " " slack[i] >> "finedOutput.txt" \
} \
} \
{ \
} \
END { \
}' finedOutput.txt
```

08_helper.csh

```
#!/bin/csh -f

set input1 = $1
set input2 = $2

awk 'BEGIN { \
    i = 1 \
    lineCount = 0 \
    fre = 0 \
    '$input2' \
} \
{ \
    lineCount++  \
    ele[i] = $0 \
    i++ \
} \
END { \
    printf "%-8s | %-6s | %-6s | %-20s \n", "WNS", "TNS", "NVP", "module2module" \
    printf "%s%s%s%s\n", "----------------", "------", "------", "--------------------" \
    for (i = 1; i <= lineCount; i++) { \
```

```
        tempString = ele[i] \
        subStart[i] = substr(ele[i], 0, 2 * '$input2' - 1) \
        subEnd[i] = substr(tempString, index(tempString, " ") + 1, 2 * '$input2' - 1) \
        slack[i] = substr(tempString, index(tempString, "-")) \
        extract[i] = subStart[i] " -> " subEnd[i] \
    } \
    temp = 0 \
    total = 0 \
    min = 999 \
    flag = 0 \
    sumT = 0 \
    sumMin = 999 \
    count = 0 \
    for (i = 1; i <= lineCount; i++) { \
        for (j = i; j <= lineCount; j++) { \
            if (extract[j] == extract[i]) { \
                fre++ \
                total += slack[j] \
                if (min + 0 > slack[j] + 0){ \
                    min = slack[j] \
                } \
                temp = j \
            } \
        } \
        printf "%-8s | %-6s | %-6s | %-20s \n", min, total, fre, extract[temp] \
        sumT += total \
        count += fre \
        if (slack[temp] + 0 < sumMin + 0) sumMin = slack[temp] \
        min = 999 \
        fre = 0 \
        total = 0 \
        i = temp \
    } \
    printf "%s%s%s%s\n", "----------------", "------", "------", "--------------------" \
    printf "%-8s | %-6s | %-6s | %-20s \n", sumMin, sumT, count, "" \
}' $input1
```

Output:

Hình 9: *Output for Exercise 8*

9 Link to the exercises

Github link to my exercises: https://github.com/CSE-NguyenThanhQui/AWK

I will push my source code along with input file to this repository.

My email: nguyenqui010402@gmail.com