

SYNOPSYS
INTENSIVE TRAINING PROGRAM 2023



EXERCISE REPORT

AWK - SCRIPTING LANGUAGE

Trainee: Nguyễn Thành Quý
Group: GROOT

Ho Chi Minh City, 6/2023

Mục lục

| | | |
|---|-----------------------|----|
| 1 | Exercise 1: | 2 |
| 2 | Exercise 2 | 2 |
| 3 | Exercise 3 | 5 |
| 4 | Exercise 4 | 6 |
| 5 | Exercise 5 | 7 |
| 6 | Exercise 6 | 8 |
| 7 | Exercise 7 | 10 |
| 8 | Exercise 8 | 10 |
| 9 | Link to the exercises | 14 |

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:

```
1  #!/bin/csh -f
2
3  set input1 = $1
4  set input2 = $2
5  set mode = $3
6  set number1 = $4
7  set number2 = $5
8
9  rm -rf 02_com1.txt 02_com2.txt 02_same.txt 02_diff.txt 02_full.txt
10
11 # sort column and lines for 02_input1.txt and 02_input2.txt
12
13 awk 'BEGIN{\
14     column = "column";\
15     line = "line";\
16 }\
17 {\
18     if ('$mode' == line){\
19         if (FNR == NR) {\
20             if (NR == '$number1') {\
21                 for (i = 1; i <= NF; i++) {\
```

```

22         line1[i] = $i;\
23         count1++;\
24     }\
25 }\
26 } else {\
27     if (FNR == '$number2') {\
28         for (i = 1; i <= NF; i++) {\
29             line2[i] = $i;\
30             count2++;\
31         }\
32     }\
33 }\
34 } else if ($mode == column){\
35     if (FNR == NR) {\
36         line1[FNR] = '$number1';\
37         count1++;\
38     } else {\
39         line2[FNR] = '$number2';\
40         count2++;\
41     }\
42 }\
43 }\
44 END {\
45     for (i = 1; i <= count1; i++) {\
46         print line1[i] >> "02_com1.txt";\
47     }\
48     for (i = 1; i <= count2; i++) {\
49         print line2[i] >> "02_com2.txt";\
50     }\
51 }' $input1 $input2
52
53 # combine two files
54
55 awk 'BEGIN{ \
56     i = 1; \
57     n = 1; \
58     while ((getline < "02_com2.txt") > 0 ) {\
59         a[i] = $1;\
60         i++;\
61         print $1 >> "02_full.txt";\
62     }\
63     close ("02_com2.txt");\
64     while ((getline < "02_com1.txt") > 0 ) {\
65         print $1 >> "02_full.txt";\
66     }\
67     close ("02_com1.txt");\
68 }\
69 {\

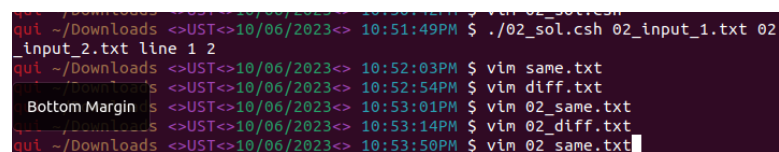
```

```

70     ipt = $1;\
71     for (k = 1; k <= i; k ++ ) {\
72         if (ipt == a[k]) {\
73             count_same++;\
74             print ipt >> "02_same.txt";\
75         }\
76     }\
77 }\
78 END {\
79     print "Number of same element:", count_same >> "02_same.txt";\
80 }' 02_com1.txt
81
82 # compare
83
84 awk 'BEGIN {\
85     while ((getline < "02_same.txt") > 0 ) {\
86         n++;\
87         a[n] = $1;\
88     }\
89     close ("02_same.txt");\
90 }\
91 {\
92     b = $1;\
93     for (k = 1; k <= n; k++) {\
94         if (a[k] == b) {\
95             k = n + 1;\
96         } else {\
97             if (k == n) {\
98                 count_diff++;\
99                 print b >> "02_diff.txt";\
100             }\
101         }\
102     }\
103 }\
104 END {\
105     print "Number of diff element:", count_diff >> "02_diff.txt";\
106 }' 02_full.txt

```

Output:



```

root@kali:~/Downloads <>UST<>10/06/2023<> 10:51:42PM $ ./02_sol.csh 02_input_1.txt 02_input_2.txt line 1 2
root@kali:~/Downloads <>UST<>10/06/2023<> 10:52:03PM $ vim same.txt
root@kali:~/Downloads <>UST<>10/06/2023<> 10:52:54PM $ vim diff.txt
root@kali:~/Downloads <>UST<>10/06/2023<> 10:53:01PM $ vim 02_same.txt
root@kali:~/Downloads <>UST<>10/06/2023<> 10:53:14PM $ vim 02_diff.txt
root@kali:~/Downloads <>UST<>10/06/2023<> 10:53:50PM $ vim 02_full.txt

```

Hình 1: Output for Exercise 2

```
n
b
a
m
Number of diff element: 4
```

Hình 2: Output for Exercise 2

```
c
Number of same element: 1
~
```

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 structure 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:

```
108 #!/bin/csh -f
109
110 set input1 = $1
111
112 awk 'BEGIN {\
113 }\
114 {\
115     count[$0]++\
116 }\
117 END {\
118     n = asorti(count, sorted)\
119     for (i = 1; i <= n; i++) {\
120         line = sorted[i]\
121         print line, count[line]\
122     }\
123 }' $input1
124 }
```

Output:

```

quit ~/Downloads <>UST<>10/06/2023<> 08:14:31PM $ ./03_sol.csh 03_input.txt
a 6
b 1
c 2
d 2
e 1
i 1
m 1

```

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 position 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:

```

125 #!/bin/csh -f
126
127 set input1 = $1
128 set input2 = $2
129
130 awk 'BEGIN { \
131     i = 1 \
132     lineCount = 0 \
133     fre = 0 \
134     '$input2' \
135 } \
136 { \
137     lineCount++ \
138     arr[i] = substr($0, 0, 2 * '$input2' - 1) \
139     i++ \
140 } \

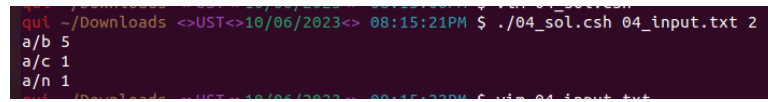
```

```

141 END { \
142     n = asort(arr) \
143     temp = 0 \
144     for (i = 1; i <= lineCount; i++) { \
145         #print arr[i] \
146     } \
147     for (i = 1; i <= lineCount; i++) { \
148         for (j = i; j <= lineCount; j++) { \
149             if (arr[j] == arr[i]) { \
150                 fre++; \
151                 temp = j \
152             } \
153         } \
154         print arr[i] " " fre \
155         fre = 0 \
156         i = temp \
157     } \
158 }' $input1

```

Output:



```

qui ~/Downloads <>UST<>10/06/2023<> 08:15:21PM $ ./04_sol.csh 04_input.txt 2
a/b 5
a/c 1
a/n 1

```

Hình 5: Output for Exercise 4

5 Exercise 5

Complexity: $O(n)$

Source code:

```

159 #!/bin/csh -f
160 set input1 = $1
161 set input2 = $2
162 set wordCount = $0
163 awk 'BEGIN {\
164     lineCount = 0 \
165     charCount = 0 \
166     wordCount = 0 \
167 } \
168 { \
169     lineCount = lineCount + 1 \
170     blankCount = NF \
171     wordCount += blankCount \
172     for (i = 1; i <= length($0); i++) { \
173         if (substr($0, i, 1) == " ") \
174             continue \

```

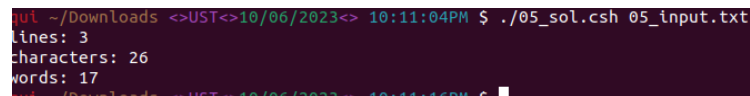


```

175 else \
176 charCount++ \
177 } \
178 } \
179 END { \
180 print lineCount \
181 print charCount \
182 print wordCount \
183 }' $input1

```

Output:



```

jul ~/Downloads <>UST<>10/06/2023<> 10:11:04PM $ ./05_sol.csh 05_input.txt
lines: 3
characters: 26
words: 17
jul ~/Downloads <>UST<>10/06/2023<> 10:11:16PM $

```

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.

```

1 Path 1:
2 StartP: a/b/c/d/
3 e/f/g/h
4 EndP: m/n/q/q/
5 r/s/t
6 Slack: -0.6

```

Source code:

```

184 #!/bin/csh -f
185
186 set input1 = $1
187
188 awk 'BEGIN {\
189     # Decoration lines\
190     printf "%-12s | %-20s | %-20s | %-10s \n", "Path num", "StartPoint", "EndPoint", "Slack" \
191     printf "%s%s%s\n", "-----", "-----", "-----", '
192 \
193     i = 1 \
194     command = "cat 06_input.txt | grep Path" \

```

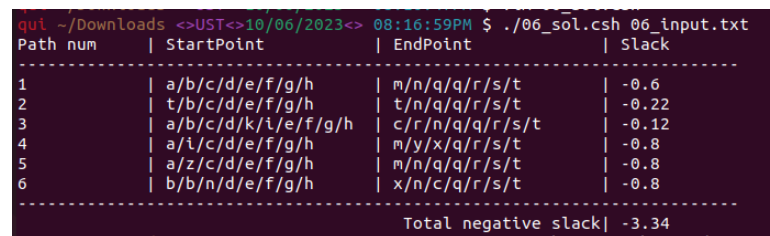
```

195     while (command | getline > 0) { \
196 num = substr($2, 0, length($2) - 1) \
197     path[i] = num \
198     i++ \
199   } \
200   i = 1 \
201   command = "cat 06_input.txt | grep StartP" \
202   command1 = "cat 06_input.txt | grep -A 1 StartP | grep -v StartP | grep /" \
203   while (command | getline > 0) { \
204     startP[i] = $2 \
205     i++ \
206   } \
207   i = 1 \
208   while (command1 | getline > 0) { \
209     temp = startP[i] $0 \
210     startP[i] = temp\
211     i++ \
212   } \
213   i = 1 \
214   command = "cat 06_input.txt | grep EndP" \
215   command1 = "cat 06_input.txt | grep -A 1 EndP | grep -v EndP | grep /" \
216   while (command | getline > 0) { \
217     endP[i] = $2 \
218     i++ \
219   } \
220   i = 1 \
221   while (command1 | getline > 0) { \
222     temp = endP[i] $0 \
223     endP[i] = temp \
224     i++ \
225   } \
226   i = 1 \
227   totalSlack = 0 \
228   command = "cat 06_input.txt | grep Slack" \
229   while (command | getline > 0) { \
230     slack[i] = $2 \
231     totalSlack += slack[i] \
232     i++ \
233   } \
234   n = i \
235   for (j = 1; j < n; j++) { \
236     printf "%-12s | %-20s | %-20s | %-10s \n", path[j], startP[j], endP[j], slack[j] \
237   } \
238   printf "%s%s%s\n", "-----", "-----", "-----", "-----", " "
239   printf "%59s| %-10s \n", "Total negative slack", totalSlack \
240 } \
241 { \
242 } \

```

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

Output:



| Path num | StartPoint | EndPoint | Slack |
|----------------------|---------------------|---------------|-------|
| 1 | a/b/c/d/e/f/g/h | m/n/q/r/s/t | -0.6 |
| 2 | t/b/c/d/e/f/g/h | t/n/q/r/s/t | -0.22 |
| 3 | a/b/c/d/k/l/e/f/g/h | c/r/n/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/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:

```
a/b/c/d/e/f/g/h m/n/q/q/r/s/t -0.6
a/b/c/z/t/r/q m/n/q/x/f/z -0.22
a/b/c/d/e/i/e/f/g/h m/n/v/5/q/r/s/t -0.12
c/a/z/d/i/f/g/h q/w/e/q/r/s/t -0.26
c/a/z/d/e/f/g/h q/w/e/k/4/s/t -0.02
c/a/n/d/e/f/g/h q/w/v/q/r/s/t -0.68
c/a/n/f/k/r/m r/t/c/d/g/2/c/ -0.06
c/a/n/f/k/x/m r/t/c/d/e/e/e -0.08
```

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

```

245 #!/bin/csh -f
246
247
248 set input1 = $1
249 set input2 = $2
250 rm -rf finedOutput.txt
251 awk 'BEGIN {\
252   command = "cat 08_input.txt | grep /" \
253   i = 1 \
254   while (command | getline > 0) { \
255     startP[i] = $3 \
256     endP[i] = $5 \
257     slack[i] = $ 7 \
258     print startP[i]" "endP[i]" " " slack[i] >> "finedOutput.txt" \
259   } \
260 } \
261 { \
262 } \
263 END { \
264 }' finedOutput.txt

```

08_helper.csh

```

265 #!/bin/csh -f
266
267 set input1 = $1
268 set input2 = $2
269
270 awk 'BEGIN { \
271   i = 1 \
272   lineCount = 0 \
273   fre = 0 \
274   '$input2' \
275 } \
276 { \
277   lineCount++ \
278   ele[i] = $0 \
279   i++ \
280 } \
281 END { \
282   printf "%-8s | %-6s | %-6s | %-20s \n", "WNS", "TNS", "NVP", "module2module" \
283   printf "%s%s%s\n", "-----", "-----", "-----", "-----" \
284   for (i = 1; i <= lineCount; i++) { \

```

```

285     tempString = ele[i] \
286     subStart[i] = substr(ele[i], 0, 2 * '$input2' - 1) \
287     subEnd[i] = substr(tempString, index(tempString, " ") + 1, 2 * '$input2' - 1) \
288     slack[i] = substr(tempString, index(tempString, "-")) \
289     extract[i] = subStart[i] " -> " subEnd[i] \
290 } \
291 temp = 0 \
292 total = 0 \
293 min = 999 \
294 flag = 0 \
295 sumT = 0 \
296 sumMin = 999 \
297 count = 0 \
298 for (i = 1; i <= lineCount; i++) { \
299     for (j = i; j <= lineCount; j++) { \
300         if (extract[j] == extract[i]) { \
301             fre++ \
302             total += slack[j] \
303             if (min + 0 > slack[j] + 0){ \
304                 min = slack[j] \
305             } \
306             temp = j \
307         } \
308     } \
309     printf "%-8s | %-6s | %-6s | %-20s \n", min, total, fre, extract[temp] \
310     sumT += total \
311     count += fre \
312     if (slack[temp] + 0 < sumMin + 0) sumMin = slack[temp] \
313     min = 999 \
314     fre = 0 \
315     total = 0 \
316     i = temp \
317 } \
318 printf "%s%s%s%s\n", "-----", "-----", "-----", "-----" \
319 printf "%-8s | %-6s | %-6s | %-20s \n", sumMin, sumT, count, "" \
320 }' $input1

```

Output:

```

qui ~/Downloads <>UST<>10/06/2023<> 08:17:12PM $ ./08_helper.csh finedOutput.tx
t 4
WNS      | TNS      | NVP      | module2module
-----
-0.6      | -0.6      | 1         | a/b/c/d -> m/n/q/q
-0.22     | -0.22     | 1         | a/b/c/z -> m/n/q/x
-0.12     | -0.12     | 1         | a/b/c/d -> m/n/v/5
-0.26     | -0.26     | 1         | c/a/z/d -> q/w/e/q
-0.02     | -0.02     | 1         | c/a/z/d -> q/w/e/k
-0.68     | -0.68     | 1         | c/a/n/d -> q/w/v/q
-0.08     | -0.14     | 2         | c/a/n/f -> r/t/c/d
-----
-0.68     | -2.04     | 8         |
qui ~/Downloads <>UST<>10/06/2023<> 08:17:30PM $ vim 03_sol.csh

```

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