Technical University of Crete

Εργαλεία Ανάπτυξης Λογισμικού

Αναφορά 1ης Εργαστηριακής άσκησης

Φοιτητές

Απόστολος-Νικόλαος Βαϊλάκης | 2014030174

Στέφανος Καλογεράκης | 2015030064

Διδάσκων

Δεληγιαννάκης Α.

Επισκόπηση

Στην πρώτη εργαστηριακή άσκηση του μαθήματος μας ζητήθηκε να υλοποιήσουμε δύο shell scripts, ένα για τον υπολογισμό των παραμέτρων γραμμικής παλινδρόμησης δύο διανυσμάτων, και ένα για τον υπολογισμό του συνολικού σκορ ποδοσφαιρικών ομάδων, και την ταξινομημένη εκτυπωση τους. Μέσω αυτών εγινε μια καλή εισαγωγή στον κόσμο του ισχυρού και ευέλικτου bash scripting.

Μερος Α

Κώδικας υλοποίησης

Για ευκολία στην υλοποίηση, το πρόβλημα χωρίστηκε σε επιμέρους συναρτήσεις:

sum_vector():

Δέχεται ως είσοδο αριθμητικό πίνακα X, k στοιχείων και επιστρέφει:

$$\sum_{i=0}^{k-1} X[i]$$

Για τον παραπάνω υπολογισμό γίνεται χρήση του εργαλείου *bc,* το οποιο και επιλύει βασικές μαθηματικές εκφράσεις.

```
# 15| for i in "${_vector[@]}"; do
# 16|     sum=`echo "scale=9; $sum + $i" | bc `
# 17| done
```

sum_vector_pow2():

Δέχεται ως είσοδο αριθμητικό πίνακα X, k στοιχείων και επιστρέφει:

$$\sum_{i=0}^{k-1} (X[i])^2$$

Κάνοντας για άλλη μια φορά χρήση του εργαλείου bc

```
# 29| for i in "${_vector[@]}"; do
# 30|     sum=`echo "scale=9; $sum + ($i^2)" | bc `
# 31| done
```

sum_vector_vector():

Δέχεται ως είσοδο αριθμητικούς πίνακες X,Y, k στοιχείων και επιστρέφει:

$$\sum_{i=0}^{k-1}(X[i] * Y[i])$$

Κάνοντας για ακόμη μια φορά χρήση του εργαλείου bc

```
# 49| for i in $(seq 0 $len); do
# 50|     sum=`echo "scale=9; $sum +(${_vector_X[$i]}*${_vector_Y[$i]})" | bc`
# 51| done
```

regr():

Δέχεται ως είσοδο αριθμητικούς πίνακες X,Y, k στοιχείων και επιστρέφει τις παραμέτρους a, b, c, err, γραμμικής παλινδρόμησης, όπως αυτές ορίστηκαν στην εκφώνηση της άσκησης, για τον υπολογισμό των οποίων χρησιμοποιεί τις συναρτήσεις που αναφέρθηκαν παραπάνω.
Για την παράμετρο a, υπολογίζονται ξεχωριστά αριθμητής και παρονομαστής του κλάσματος, ωστε να αποφευχθεί διαίρεση με το 0, η οποία και προκύπτει όταν ένα από τα διανύσματα έχει όλα τα στοιχεία του ίσα. Σε αυτήν την περίπτωση η συνάρτηση επιστρέφει ένα μήνυμα σφάλματος.

```
# 76| # calculate a
# 77| a_num=`echo "scale=9; ($length * $sum_xy) - ($sum_x * $sum_y)" | bc`
# 78| a_denom=`echo "scale=9; ( $length * $sum_x2 ) - ($sum_x * $sum_x)" | bc`
# 79|
# 80| # check for division by zero
# 81| if [[ $a_denom == '0' ]]; then
# 82|     echo $RED "\bError: Incorrect vector" $NC
# 83|     return 1
# 84| fi
```

Αφου γίνει αυτός ο έλεγχος, υπολογίζονται όλες οι παράμετροι

```
# 86| a=`echo "scale=2; $a_num / $a_denom" | bc | sed -r $sed_regex`
# ..| ...
# 89| b=`echo "scale=2; ($sum_y - ($a * $sum_x)) / ($length)" | bc | sed -r   #   |
$sed_regex`
# ..| ...
# 92| c=1
# ..| ...
# 98| err=0
# 99| for i in $(seq 0 $len); do
#100|     err=`echo "scale=2; $err + (${Vector_Y[$i]} - ( $a *                #   |
(${_Vector_X[$i]}) | + $b))^2" | bc | sed -r $sed_regex`
#101| done
```

Για την κατάλληλη μορφοποίηση των αποτελεσμάτων χρησιμοποιείται το εργαλείο sed (γραμμες #86 #89 #100) το οποίο και καλείται με την παράμετρο -r αφου γίνεται χρήση extended regex. Η regular expression που χρησιμοποιείται είναι η:

```
# 74| sed_regex='s|^(-*)\.|\10.|; s|([0-9]+\.[0-9]{2})[0-9]*|\1|;
#   | s|(\.[0-9])$|\10|;s|\.0{2}||'
```

Η οποία καθοδηγεί το sed να:

1. Εισάγει leading zero στους αριθμούς ανάμεσα σε -1 και 1. Αυτό χρειάζεται επειδή το bc αναπαριστά αριθμούς όπως το -0.23 ως -.23.
2. Αν υπάρχουν πάνω από 2 δεκαδικά ψηφία τα κόβει. Η παράμετρος scale=2 του bc δεν αρκεί, αφού στην περίπτωση που κάνουμε την πράξη 1 + 1.1234 το bc θα επιστρέψει 2.1234.
3. Αν υπάρχει μόνο ένα δεκαδικό ψηφίο, εισάγει ένα trailing zero στο τέλος.
4. Αν ο αριθμός είναι ακέραιος, διαγράφει όλα τα δεκαδικά ψηφία, και την υποδιαστολή.

regr_file():

Δέχεται ως είσοδο το ονομα ενος αρχείου, διαβάζει τα περιεχόμενά του (αφου ελέγξει ότι υπάρχει και είναι προσπελάσιμο), κατασκευάζει τα δύο διανύσματα X και Y και τα περνάει στην συνάρτηση regr().

Για την προσπέλαση του αρχείου γίνεται χρήση των εντολών **while** και **read**. Η μεταβλητή **IFS** χρησιμοποιείται ως delimemeter απο την **read** για να “σπαει” την γραμμη σε πολλες μεταβλητές, στην δική μας περίπτωση αυτή ορίζεται ως ‘’. Δηλαδή

ως τίποτα, ωστε να μην πραγματοποιείται ο διαχωρισμός της γραμμης σε αυτο το στάδιο.
Τα αρχεία εισόδου δεν τερματίζουν με χαρακτήρα **newline**[1], και η εντολή **read** τερματίζει την **while** στην προτελευταία γραμμή. Γι αυτό το λόγο προστίθεται στην **while** ένας δεύτερος έλεγχος για να συμπεριλάβει την τελευταία γραμμή.

```
#125| while IFS='' read -r line || [[ -n "$line" ]]; do
```

Έπειτα κάθε γραμμή χωρίζεται σε 2 κελιά πίνακα, αυτό γίνεται μετατρέποντας τον delimiter που ορίζει η άσκηση σε κενό, και ερμηνεύοντας το αποτέλεσμα ως πίνακα.

```
#127| arrLine=(${line//:/ })
```

Ετσι για παράδειγμα το **23:43** ερμηνεύεται ως πίνακα **(23 43).**

Τελος τα δύο στοιχεία ελέγχονται για λάθη με βάση το παρακάτω regex

```
#130| regex='^[0-9]+(\.[0-9]+)?$'
```

ωστε το προγραμμα να δέχεται μόνο αριθμούς στην είσοδο και εισάγονται στους πίνακες Vector_X και Vector_Y.

Παρατήρηση: *Για την κλήση των συναρτήσεων με πολλαπλούς πίνακες ως ορίσματα, αυτα περνιούνται ως pass-by-reference δηλώνοντας εσωτερικες μεταβλητές με την εντολή local -n π.χ.*

```
# 64| local -n _Vector_X=$1
# 65| local -n _Vector_Y=$2
```

Συνεπώς το πρόγραμμα απλώς καλεί την regr_file() για κάθε όνομα αρχείου που βρίσκεται στα ορίσματα. Η εντολή **echo** εδω καλείται με -e για να γίνει σωστή χρήση των '\' τα οποία χρησιμοποιήθηκαν κυρίως για την εισαγωγή χρωμάτων στα μηνύματα λάθος.

```
# 64| for file in "$@"
# 64| do
# 64|     echo -e "FILE:" $file '\b,' `regr_file $file`
# 64| done
```

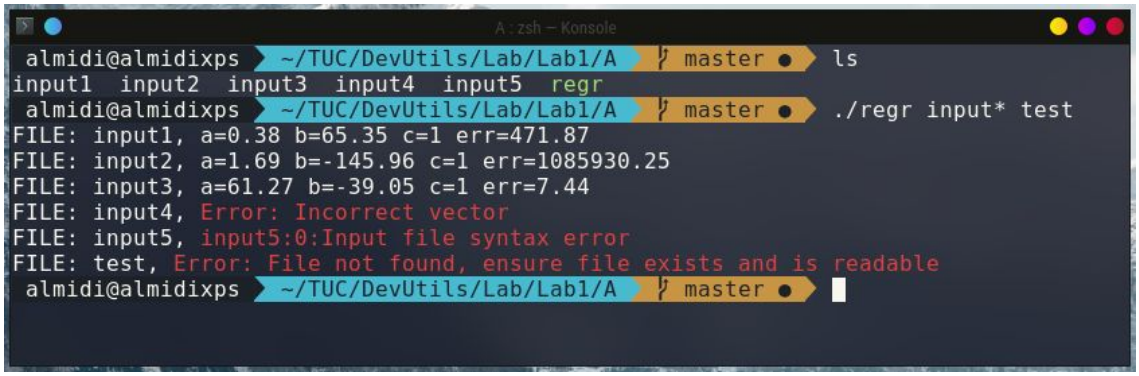
[1] Το POSIX standard ορίζει ότι κάθε γραμμή αρχείου πρέπει να τελειώνει με newline character

Παράδειγμα εκτέλεσης

Για την επαλήθευση ορθής λειτουργίας του προγράμματος δημιουργήθηκαν διάφορα δοκιμαστικά αρχεία ικανά να δοκιμάσουν πολλές περιπτώσεις. Πιο συγκεκριμένα τα αρχεία ελέγχουν αν το προγραμμα μορφοποιεί σωστά τους αριθμούς εξόδου, και αν τα αρχεία εισόδου έχουν σωστή μορφή, με ορθά διανύσματα.

input1	input2	input3	input4	input5
43:99 21:65 25:79 42:75 57:87 59:81	232.1223:13.3342 124.12523:14.2232 12.1023:22.1233 481.2022:1441.234 491.103:44.963	1.47:52.21 1.50:53.12 1.52:54.48 1.55:55.84 1.57:57.20 1.60:58.57 1.63:59.93 1.65:61.29 1.68:63.11 1.70:64.47 1.73:66.28 1.75:68.10 1.78:69.92 1.80:72.19 1.83:74.46	10:1 10:2 10:3 10:4 10:5 10:6 10:7	1.21:23..2 5.41:4.43 34.33:6.21

Τέλος στην κλήση του προγράμματος προστέθηκε και ως όρισμα αρχείο που δεν υπάρχει.



Μερος Β

Κώδικας υλοποίησης

Στο πρώτο στάδιο της υλοποίησης πραγματοποιείται ανάγνωση των δεδομένων με τα ονόματα αλλά και τα αποτελέσματα των ομάδων από το αρχείο το οποίο περνάει σαν παράμετρο ο εκάστοτε χρήστης. Από αυτό προκύπτει όλη η πληροφορία προς επεξεργασία.

Τα δεδομένα έχουν ανά γραμμή ένα συγκεκριμένο format το οποίο δίνεται απο την εκφώνηση και είναι το εξής:

Ομάδα1-Ομάδα2:Σκορ1-Σκορ2

Η εξαγωγή όλων των απαιτούμενων δεδομένων έγινε με την χρήση κανονικών εκφράσεων(Regular Expression). Πιο συγκεκριμένα, η regular expression που αξιοποιήθηκε είναι:

```
#17| regex="^([a-zA-Z0-9][a-zA-Z0-9.& ]*[a-zA-Z0-9])\-([a-zA-Z0-9][a-zA-Z0-9.&
]*[a-zA-Z0-9])\:((0|[1-9]+)-(0|[1-9]+))$"
```

Αξίζει να σημειωθεί ότι μια κανονική έκφραση αποτελεί ένα αυστηρό μοτίβο αναζήτησης μια ακολουθίας χαρακτήρων και οποιοσδήποτε χαρακτήρας δεν είναι αυστηρά ορισμένος από τον χρήστη μπορεί να οδηγήσει την κανονική έκφραση σε λάθος αποτέλεσμα. Η εκφώνηση της άσκησης δεν όριζε με μεγάλη σαφήνεια το dataset ή λεπτομέρειες αναφορικά με το τελικό αρχείο εκτέλεσης αλλά πραγματοποιήθηκαν συμβάσεις για την υποστήριξη αρκετών ρεαλιστικών περιπτώσεων. Οι περιπτώσεις/συμβάσεις που ικανοποιούνται από τον τρέχον κώδικα είναι:

- Υποστήριξη λατινικού αλφάβητου για τα ονόματα ομάδων (πχ Greece)
- Υποστήριξη αριθμών στα ονόματα ομάδων (πχ Portugal1992 ή 1992Portugal)
- Υποστήριξη έγκυρου σκορ (πχ το 01-04 δεν είναι έγκυρο, αλλά το 1-4 είναι)
- Υποστήριξη κενού χαρακτήρα (πχ Crystal Palace)
- Υποστήριξη τελείας (πχ Man. United)
- Υποστήριξη ampersand χαρακτήρα (πχ Brighton & Home Albion)

Οι περιπτώσεις που καλύπτονται αναγράφονται και στον πηγαίο κώδικα σαν σχόλια.

***Προσοχή**: Στην περίπτωση των 3 τελευταίων χαρακτήρων υποστηρίζεται μόνο αν αυτοί οι χαρακτήρες βρίσκονται ενδιάμεσα στο όνομα και όχι στην αρχή του ή στο τέλος(πχ .Spain ή Portugal& δεν υποστηρίζονται). Επίσης δεν υποστηρίζεται η περίπτωση που το όνομα της ομάδας είναι ο κενός χαρακτήρας.*

Αφού η υλοποίηση πραγματοποιήθηκε με χρήση regular expression αξιοποιήθηκε ο πίνακας BASH_REMATCH(Regular Expression Matching) όπως χαρακτηριστικά φαίνεται παρακάτω

```
# 63| team1=${BASH_REMATCH[1]}
# 64| team2=${BASH_REMATCH[2]}
```

Ο πίνακας αυτός δημιουργείται και ενημερώνεται κατά την σύγκριση του regular expression με την εκάστοτε γραμμή του αρχείου. Το κάθε κομμάτι του string που γίνει match με την regular expression αποθηκεύεται στον πίνακα. Σε ένα παράδειγμα από την εκτέλεση του κώδικα μας φαίνεται το output παρακάτω

```
#61 echo ${BASH_REMATCH[@]}
#output Portugal-Greece:1-2 Portugal Greece 1-2 1 2
#more info about rematch
#http://molk.ch/tips/gnu/bash/rematch.html
```

Η δομή που αξιοποιήθηκε για την αποθήκευση των δεδομένων είναι τέσσερα ανεξάρτητα arrays:

- Πίνακας με τα ονόματα των ομάδων
- Πίνακας με βαθμολογίες των ομάδων
- Πίνακας με τα γκολ υπέρ των ομάδων
- Πίνακας με τα γκολ κατά των ομάδων

Όλοι οι πίνακες ενημερώνονται παράλληλα προκειμένου να εξασφαλίζεται συνοχή στο κώδικα. Για την κάθε ομάδα που διαβάζεται πραγματοποιείται αναζήτηση στο πίνακα των ομάδων για την περίπτωση που υπάρχει ήδη. Σε αυτή την περίπτωση,

ενημερώνονται όλες οι εγγραφές κατάλληλα ενώ στην αντίθετη περίπτωση γίνεται προσθήκη καινούργιων εγγραφών στο τέλος κάθε πίνακα.

***Παρατήρηση**: Η σύγκριση όλων των ομάδων γίνεται αφού έχουν μετατραπεί πρώτα σε lowercase(πχ Arsenal και arsenal είναι μια εγγραφή). Το όνομα που εμφανίζεται τελικά είναι με την μορφή της πρώτης καινούργιας εγγραφής που συνάντησε το αρχείο.*

Στην συνέχεια, προκειμένου να προκύψει το απαιτούμενο αποτέλεσμα γίνεται χρήση δύο temporary files τα οποία στο τέλος της εκτέλεσης διαγράφονται.

Στο πρώτο, με όνομα **tempfile**, συγκεντρώνονται όλα τα arrays χωρίς ταξινόμηση ενώ τα περιεχόμενα τους χωρίζονται με κομμα(,) για την διευκόλυνση της ταξινόμησης τους δημιουργώντας έναν νοητό πίνακα.

Στο δεύτερο, με όνομα **tempfile2**, γίνεται η τελική ταξινόμηση με βάση την εκφώνηση όπως φαίνεται στην συνέχεια

```
sort -t ',' -k2,2r -k1,1  tempfile | tr , ' ' > tempfile2
```

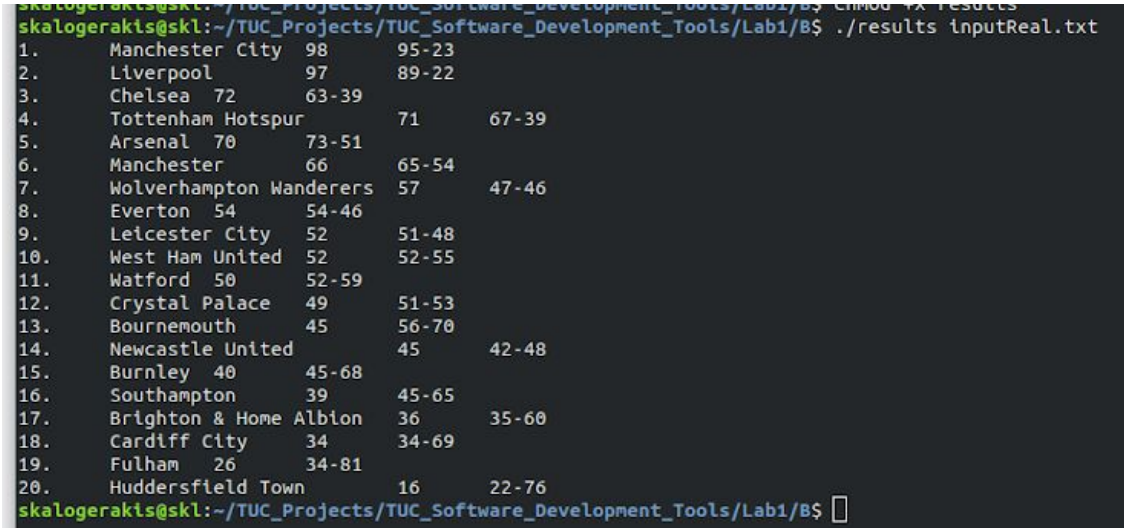
Αρχικά η ταξινόμηση γίνεται με βάση την τελική βαθμολογία της κάθε ομάδας με το όρισμα -r προκειμένου οι ομάδες με την μεγαλύτερη τιμή να φαίνονται πρώτες, ενώ σε περιπτώσεις ισοβαθμίας γίνεται η σύγκριση με βάση το όνομα της ομάδας όπου τα string με μικρότερη τιμή προηγούνται, από την εκφώνηση. Τέλος, αφαιρούνται τα κόμμα(,) που προστέθηκαν κατά την δημιουργία του προηγούμενου προσωρινού αρχείου και εμφανίζονται τα τελικά αποτελέσματα με την μορφή που ζητήθηκε χωρισμένα με ένα tab.

Κατά την εκτέλεση του κώδικα, υπάρχουν κάποιοι βασικοί έλεγχοι εγκυρότητας. Πιο συγκεκριμένα, ελέγχεται ότι το αρχείο που ζητήθηκε από τον χρήστη υπάρχει και ότι δίνεται μόνο ένα όρισμα ως παράμετρος στο αρχείο. Αν αποτύχει καποιος απο αυτούς τους ελέγχους ο κώδικας δεν συνεχίζει να εκτελείται και εμφανίζεται μήνυμα σφάλματος. Τέλος, αν σε κάποια γραμμή το regular expression δεν κάνει match εμφανίζεται μήνυμα σφάλματος χωρίς να τερματίζεται η εκτέλεση του κώδικα

Παράδειγμα εκτέλεσης

Για την επαλήθευση ορθής λειτουργίας του προγράμματος δημιουργήθηκαν διάφορα δοκιμαστικά αρχείο ικανά να δοκιμάσουν πολλές περιπτώσεις. Παρακάτω χρησιμοποιούμε αρχείο εισόδου ένα αρχείο που έγινε κοινόχρηστο από συμφοιτητή στην συζήτηση στο courses και περιέχει τα αποτελέσματα του αγγλικού πρωταθλήματος την σεζόν 2018-2019.

Το συγκεκριμένο αρχείο χρησιμοποιήθηκε ώς βάση αλλά πραγματοποιήθηκαν αλλαγές για την αποκατάσταση των πραγματικών ονομάτων όλων των ομάδων με επαναφορά ειδικών συμβόλων και κενών. Έτσι καθίσταται δυνατή η πλήρης αναπαράσταση του τελικού βαθμολογικού πίνακα με εξέταση αρκετών περιπτώσεων αναφορικά με την ονοματοδοσία των ομάδων.



Στην παραπάνω εικόνα φαίνεται η εκτέλεση του κώδικα μας με αρχείο εισόδου το inputReal.txt το οποίο είναι το κατάλληλα μορφοποιημένο αρχείο δοκιμών που προαναφέρθηκε. Στην συνέχεια, φαίνεται ο τελικός πίνακας βαθμολογιών μετά από αναζήτηση στο διαδίκτυο.

Pos	Team [v·t·e]	Pld	W	D	L	GF	GA	GD	Pts
1	Manchester City **(C)**	38	32	2	4	95	23	+72	**98**
2	Liverpool	38	30	7	1	89	22	+67	**97**
3	Chelsea	38	21	9	8	63	39	+24	**72**
4	Tottenham Hotspur	38	23	2	13	67	39	+28	**71**
5	Arsenal	38	21	7	10	73	51	+22	**70**
6	Manchester United	38	19	9	10	65	54	+11	**66**
7	Wolverhampton Wanderers	38	16	9	13	47	46	+1	**57**
8	Everton	38	15	9	14	54	46	+8	**54**
9	Leicester City	38	15	7	16	51	48	+3	**52**
10	West Ham United	38	15	7	16	52	55	−3	**52**
11	Watford	38	14	8	16	52	59	−7	**50**
12	Crystal Palace	38	14	7	17	51	53	−2	**49**
13	Newcastle United	38	12	9	17	42	48	−6	**45**
14	Bournemouth	38	13	6	19	56	70	−14	**45**
15	Burnley	38	11	7	20	45	68	−23	**40**
16	Southampton	38	9	12	17	45	65	−20	**39**
17	Brighton & Hove Albion	38	9	9	20	35	60	−25	**36**
18	Cardiff City **(R)**	38	10	4	24	34	69	−35	**34**
19	Fulham **(R)**	38	7	5	26	34	81	−47	**26**
20	Huddersfield Town **(R)**	38	3	7	28	22	76	−54	**16**

Συγκρίνοντας τις δύο εικόνες με τα αποτελέσματα παρατηρούμε μια πλήρη ταύτιση αποτελεσμάτων τόσο στην τελική βαθμολογία όλων των ομάδων όσο και στα γκολ υπέρ και κατά. Στα δικά μας αποτελέσματα παρατηρούμε μια μικρή διαφοροποίηση στις θέσεις 13 και 14 με τις ομάδες Bournemouth και Newcastle United.

Αυτο συμβαίνει καθώς έχουμε περίπτωση ισοβαθμίας και στην πραγματικότητα λαμβάνονται άλλοι παράγοντες υπόψιν(διαφορά γκολ, γκολ υπέρ,γκολ κατά) για την τελική κατάταξη των ομάδων. Στο πρόβλημα μας τα πράγματα ήταν πιο απλά και σε αυτή την περίπτωση η ομάδα με το μικρότερο αλφαριθμητικά string προηγείται. Όπως δηλαδή βλέπουμε και απο τα αποτελέσματα μας με την Bournemouth να βρίσκεται 13η και η Newcastle United 14η επιβεβαιώνοντας την ορθότητα της υλοποίησης μας. Άλλη μια περίπτωση ισοβαθμίας συναντάμε στις θέσεις 9 και 10 που όμως τυχαίνει να συμπίπτει με τα δικά μας αποτελέσματα.

Καταμερισμός Εργασιών

Η πρώτη άσκηση της εργασίας υλοποιήθηκε κατα βάση από τον φοιτητή Βαϊλάκη Αποστόλη ενώ η δεύτερη απο τον Καλογεράκη Στέφανο. Και τα δύο μέλη της ομάδας όμως συντέλεσαν στο τελικό αποτέλεσμα και των δύο ασκήσεων με συνεχείς προτάσεις για αλλαγές και βελτιώσεις προκειμένου να επιτευχθεί η βέλτιστη δυνατή λύση των προβλημάτων

Βιβλιογραφία

https://devhints.io/bash

https://www.rexegg.com/regex-quickstart.html

https://www.geeksforgeeks.org/bc-command-linux-examples/

http://mywiki.wooledge.org/BashFAQ/001

https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap03.html#tag_03_206

http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html

https://www.gnu.org/software/bash/manual/html_node/Shell-Parameters.html