Αναφορά Εργαστηριακής Άσκησης Δομών Δεδομένων 2014-15

Ονοματεπώνυμο : Κυριακού Ανδρόνικος

ΑΜ : 5806

Email: kyriakou@ceid.upatras.gr

Γλώσσα Προγραμματισμού που χρησιμοποιήθηκε : C++

Μέρος Α:

Για την υλοποίηση του Μέρους Α έγιναν τα εξής:

Αρχικά , για την προσπέλαση του αρχείου .csv που δίνεται σαν είσοδος στο πρόγραμμα , κατασκευάστηκε ένα header file με όνομα `FileActions`. Σε αυτό περιέχεται η μέθοδος η οποία διαβάζει το αρχείο εισόδου και το μετατρέπει σε κατάλληλη μορφή προκειμένου να είναι διαθέσιμο στο πρόγραμμα (συνάρτηση `loadCompanies`) και εκείνη η οποία αποθηκεύει τις όποιες αλλαγές έχουν γίνει όποτε το επιθυμεί ο χρήστης, ή στο τέλος του προγράμματος (`writeFile`) .

Στη συνέχεια , προκειμένου τα δεδομένα τα οποία υπάρχουν στο .csv να είναι διαθέσιμα στο πρόγραμμα δημιουργήθηκαν :

- Η κλάση `Employee`, η οποία περιέχει πληροφορίες για τον κάθε υπάλληλο
- Η κλάση `Company` , η οποία περιέχει πληροφορίες για την κάθε εταιρία, καθώς και δύο μεθόδους. Η πρώτη (`printInfo`) , εμφανίζει όλες την πληροφορία η οποία υπάρχει για κάθε εταιρία, ενώ η δεύτερη (`compByID`) έχει αποκλειστικό σκοπό την χρήση ως κριτήριο για την ενσωματωμένη συνάρτηση `sort` της C++ (ουσιαστικά κάνει μια σύγκριση ως προς τα id και επιστρέφει μια λογική τιμή ανάλογα με το αποτέλεσμα)
- Η κλάση `Companies` αποτελεί τον πυρήνα του προγράμματος καθώς ουσιαστικά αποτελεί μια συλλογή από αντικείμενα τύπου `Company` (αποθηκευμένα σε ένα vector) και έχει ως στόχο την αντιμετώπιση όλων των εταιριών ως μια οντότητα.

Η συνάρτηση `main` του προγράμματος υλοποιεί όλη την διεπαφή με τον χρήστη αλλά και διαχειρίζεται τα αντικείμενα που δημιουργήθηκαν προκειμένου να επιτευχθεί η λειτουργικότητα του προγράμματος.

Με την επιλογή 1 , ζητείται από τον χρήστη να εισάγει ένα όνομα αρχείου και καλείται η συνάρτηση `loadCompanies` με όρισμα το παραπάνω όνομα. Αν το αρχείο που δόθηκε ως είσοδος δεν είναι άδειο, φορτώνεται σε ένα αντικείμενου τύπου Companies, το companies.

Με την επιλογή 2 , ζητείται από τον χρήστη ένα όνομα για το αρχείο στο οποίο θα εγγραφούν οι εταιρίες και αφού ταξινομηθεί το vector που περιέχεται στο `companies` (έγινε σύμβαση ότι οι εταιρίες πρέπει να είναι σε αύξουσα σειρά στο αρχείο) περνιούνται στο αρχείο. Αν δεν υπάρχει αρχείο με το όνομα που δόθηκε , τότε αυτό δημιουργείται αυτόματα.

Στην επιλογή 3 , η οποία υλοποιεί την προσθήκη μια εταιρίας, δημιουργείται χώρος στο vector του companies και ενημερώνεται η νέα εταιρία που δημιουργήθηκε με στοιχεία τα οποία εισάγει ο χρήστης.

Στην επιλογή 4 , αρχικά, υλοποιήθηκε γραμμική αναζήτηση για να διαγραφεί μια εταιρία με βάση το id της. Προκειμένου να επιτευχθεί αυτό , ταξινομείται το vector του companies και στην συνέχεια καλείται η συνάρτηση `searchByID` με όρισμα 1. Αν βρεθεί η εταιρία με το συγκεκριμένο id, η συνάρτηση επιστρέφει την θέση του στο vector, και αφού διαγραφεί το αντικείμενο τύπου Company , αφαιρείται από το vector και ο συνολικός αριθμός των εταιρειών μειώνεται .

Στην επιλογή 5 , όπως και παραπάνω, καλείται η `searchByID` με όρισμα 1 και αν επιστραφεί επιτρεπτή τιμή καλείται μέσα από το vector του companies η μέθοδος `printInfo()` της συγκεκριμένης εταιρίας.

Στην επιλογή 6 , μέσω μιας for , καλείται η μέθοδος `printInfo()` κάθε εταιρίας και εμφανίζονται οι πληροφορίες της.

Στην επιλογή 7 , μια διπλή for προσπελάσει σε κάθε εταιρία, κάθε επώνυμο υπαλλήλου και αν βρεθεί το αναζητούμενο επώνυμο τότε εμφανίζεται το όνομα της εταιρίας. Δεδομένου ότι μπορεί ένα επώνυμο να εμφανίζεται σε παραπάνω από μια εταιρίες , μόλις βρεθεί το πρώτου όνομα η for δεν σπάει αλλά συνεχίζεται η αναζήτηση μέχρι το τέλος των εταιριών.

Τέλος , στην επιλογή 8 , ουσιαστικά είναι η επιλογή της εξόδου από το πρόγραμμα. Η λειτουργία του είναι πέρα από το να τερματίζει το πρόγραμμα να διαγράφει τα αντικείμενα τύπου Digital Tree και AVL που έχουν δημιουργηθεί (περισσότερα για αυτά παρακάτω) και να αποθηκεύει για κάθε ενδεχόμενο τις αλλαγές που τυχόν έγιναν σε ένα αρχείο με όνομα backup.csv. μετά από αυτό το πρόγραμμα τερματίζεται.

Μέρος Β:

Για την υλοποίηση του Β μέρους , εμπλουτίστηκε η μέθοδος `searchByID` . Προστέθηκαν ουσιαστικά , η δυαδική αναζήτηση και binary interpolation αναζήτηση. Όσον αφορά την πρώτη, δημιουργήθηκε μια νέα συνάρτηση η Binary Search ,η οποία καλείται από την `searchByID` και η οποία είναι αναδρομική. Η νέα αυτή συνάρτηση υποδιαιρεί τον χώρο της αναζήτησης κάθε φορά μέχρι να βρει το αναζητούμενο στοιχείο ή ο δείκτης του αριστερού άκρου να γίνει μεγαλύτερος από αυτόν του δεξιού και άρα να μην υπάρχει το αναζητούμενο στοιχείο.

Για την δυαδική αναζήτηση με παρεμβολή υλοποιήθηκε ο κώδικας του βιβλίου Δομές Δεδομένων του κ.Τσακαλίδη στον οποίο έγιναν οι απαραίτητες προσθήκες προκειμένου να χειρίζεται ακραίες περιπτώσεις όπως η υπέρβαση του δεξιού άκρου του πίνακα.

Η χρήση των παραπάνω δύο αναζητήσεων γίνεται στην main όπου στις επιλογές 4 και 5 ο χρήστης έχει στην διάθεση του και τα δύο παραπάνω ενώ δεν υπάρχει κάποια άλλη αλλαγή στην λειτουργικότητα του προγράμματος.

Μέρος Γ:

Σε αυτό το μέρος, υλοποιήθηκαν AVL trees. Για την επίτευξη αυτού, δημιουργήθηκε μια κλάση με όνομα AVLNode ,η οποία αναπαριστά κάθε κόμβο του AVL (επιλέχθηκε να έχει μια ιδιότητα στην οποία να αντιγράφεται μια εταιρία ολόκληρη και όχι μόνο το id της) , και μια με όνομα AVLTree η οποία αναπαριστά το δέντρο στο σύνολο του.

Η AVLTree περιέχει τις εξής μεθόδους :

- calculateHeight : υπολογίζει το ύψος του εκάστοτε κόμβου που δίνεται σαν όρισμα. Αυτό επιτυγχάνεται καλώντας τον εαυτό της με όρισμα τα παιδιά του κόμβου και επιστρέφοντας μόλις φτάσει σε φύλλο.
- balanceFactor : υπολογίζει την υψοζύγιση ενός κόμβου με χρήση της παραπάνω μεθόδου για τα δύο παιδιά του κόμβου.
- rotateLeft : η μέθοδος που υλοποιεί την αριστερή περιστροφή ενός κόμβου. Η περιστροφή γίνεται με τη δημιουργία ενός κόμβου με όνομα pivot. Αυτός παίρνει την τιμή του δεξιού παιδιού του κόμβου που έχει μη επιτρεπτή υψοζύγιση (έστω root) . Το δεξί παιδί του root παίρνει την τιμή του αριστερού παιδιού του pivot και το root γίνεται αριστερό παιδί του pivot. Τέλος επιστρέφεται ο pivot.
- rotateRight : αντίστοιχα με την rotateLeft αλλά για δεξια περιστροφή
- balancer : η μέθοδος αυτή χρησιμοποιεί όλα τα παραπάνω προκειμένου να επιτύχει την επαναφορά της ζύγισης στο δέντρο. Η επιλογή του είδους της περιστροφής που θα γίνει εξαρτάται από το αποτέλεσμα της balance Factor . Πιο συγκεκριμένα :

  Αν ο balanceFactor του κόμβου είναι >1 τότε αυτό σημαίνει ότι το δεξί υποδέντρο έχει παράνομα μεγαλύτερο ύψος από το αριστερό . Αν ταυτόχρονα ο balanceFactor του δεξιού παιδιού του κόμβου είναι μικρότερος του 0 , δηλαδή το αριστερό υποδέντρου του δεξιού παιδιού προκαλεί πρόβλημα , θα γίνει πρώτα μια δεξιά περιστροφή με ρίζα το δεξί παιδί ενώ , και να μην ισχύει ο παραπάνω περιορισμός, στην συνέχεια θα γίνει μια αριστερή περιστροφή με ρίζα τον αρχικό κόμβο.

  Αντίστοιχα αν ο balanceFactor του κόμβου είναι <1, αυτό σημαίνει ότι αριστερό υποδέντρο έχει παράνομα μεγαλύτερο ύψος από το δεξί , ενώ , όπως και παραπάνω , αν το αριστερό παιδί του κόμβου έχει balanceFactor >0 (το δεξί υποδέντρο του αριστερού παιδιού δημιουργεί πρόβλημα) θα γίνει πρώτα μια αριστερή περιστροφή με ρίζα το αριστερό παιδί και στην συνέχεια μια δεξιά περιστροφή με ρίζα τον κόμβο.

- `Insert :` χρησιμοποιείται για εισαγωγή ενός νέου κόμβου. Η μέθοδος αυτή καλεί τον εαυτό της μέχρι να βρει την θέση που πρέπει να εισαχθεί ο νέος κόμβος και μόλις το επιτύχει καλεί την balancer για να διορθώσει τυχόν λάθος υψοζυγίσεις.
- `SearchAVL` : χρησιμοποιείται προκειμένου να βρει την εταιρία με ένα δεδομένο-αναζητούμενο id και να εμφανίσει τα στοιχεία της.
- `minValueNode :` επιστρέφει τον επόμενο κόμβο στην συμμετρική διάταξη.
- `Delete` : διαγράφει ένα κόμβο από το δέντρο. Για να το επιτύχει αυτό , βρίσκει τον εκάστοτε κόμβο προς διαγραφή , ξεκινώντας από την ρίζα, και κάνει τα εξής :

  Αν ο κόμβος έχει 1 παιδί , το συνδέει με τον πρόγονο του προς διαγραφή κόμβου και διαγράφει τον κόμβο
  Αν ο κόμβος δεν έχει παιδιά τον διαγράφει
  Αν ο κόμβος έχει 2 παιδιά, βρίσκει τον επόμενο κόμβο από συμμετρική διάταξη (τον μικρότερο παρακάτω στο δέντρο) και αφού ανταλλάξει τα δυο περιεχόμενα διαγραφεί τον κόμβο - φύλλο.

  Τέλος χρησιμοποιεί την `balancer` για να αναδιαμορφώσει το δέντρο.

Όσον αφορά την `main`, για την υλοποίηση του AVL , δημιουργείται ένα αντικείμενο τύπου `AVLTree` το οποίο αρχικοποιείται μόλις φορτωθούν στο πρόγραμμα οι εταιρίες και μπορεί να προσπελαστεί στις επιλογές 4 και 5 προκειμένου να χρησιμοποιηθεί για αναζητήσεις και διαγραφές . Πιο συγκεκριμένα στην επιλογή 5 , αν μια εταιρία διαγραφεί με τρόπο στον οποίο δεν επηρεάζεται το AVL καλείται η μέθοδος `delete` με όρισμα την συγκεκριμένη εταιρία προκειμένου να συγχρονιστεί το δέντρο με το υπόλοιπο πρόγραμμα και αντίστροφα , αν κάποια εταιρία διαγραφεί με χρήση AVL γίνεται `binary search` προκειμένου να διαγραφεί η συγκεκριμένη εταιρία από το vector του companies.

Μέρος Δ:

Για την υλοποίηση των ψηφιακών δέντρων δημιουργήθηκε μια κλάση `TrieNode` η οποία αναπαριστά τον κάθε κόμβο. Η κλάση αυτή περιέχει ένα πίνακα από pointers τύπου `TrieNode` προκειμένου να κατασκευαστούν τα επίπεδα του δέντρου και ένα vector τύπου `Company` προκειμένου να αποθηκευτούν οι εταιρίες που δουλεύει ο υπάλληλος με το ζητούμενο όνομα. Παράλληλα, δημιουργήθηκε η κλάση `Trie` η οποία αναπαριστά το ψηφιακό δέντρο. Σε αυτή την κλάση αναπτύχθηκαν οι εξής μέθοδοι:

- `Insert` : δέχεται σαν όρισμα ένα επώνυμο και την εταιρία για την οποία δουλεύει. Ξεκινάει από την ρίζα του δέντρου και αφού κάνει μετατροπή το κάθε γράμμα του ονόματος σε ακέραιο προκειμένου να δεικτοδοτήσει τον πίνακα προσπελάσει κάθε επίπεδο είτε δημιουργώντας ένα νέο κόμβο όπου δεν υπάρχει είτε απλά προσπερνώντας το μέχρι να φτάσει στο τέλος του ονόματος όπου και προσθέτει στο Vector του τελευταίου κόμβου την εκάστοτε εταιρία.
- `Search` : όπως και παραπάνω ακολουθεί μια πορεία μέσω δεικτών από την ρίζα μέχρι κάποιο κόμβο , ο οποίος αν το όνομα το οποίο αναζητείται υπάρχει,

θα περιέχει τα ονόματα των εταιρειών για τις οποίες ο συγκεκριμένος υπάλληλος δουλεύει και θα εμφανιστούν.

Στη main το Trie αρχικοποιείτα μαζί με τις υπόλοιπες εταιρίες και είναι διαθέσιμο στον χρήστη κατά την αναζήτηση με βάση το επώνυμο , δηλαδή την επιλογή 7.

Μέρος Ε: (Η υλοποίηση βρίσκεται στο `DataStructures v2`*)*

Για την υλοποίηση του τελευταίου μέρους, δημιουργήθηκε μια γεννήτρια τυχαίων αρχείων , η συνάρτηση `randomFile`. Η συνάρτηση δέχεται σαν είσοδο ένα αντικείμενο τύπου `Companies` και αφού δημιουργήσει ένα vector με τυχαίους-μοναδικούς αριθμούς, το χρησιμοποιεί σαν διεύθυνση για εταιρίες στο vector του `Companies` και τις προσθέτει σε ένα νέο αντικείμενο τύπου `Companies` (έστω `searchInput`) ,το οποίο και επιστρέφει . Στην main ,στην επιλογή 5, μέσω μιας for , όλα τα ids που υπάρχουν στο νέο αντικείμενο `searchInput` χρησιμοποιούνται σαν όροι αναζήτησης και καταγράφεται συνολικός χρόνος και συνολικές συγκρίσεις (μεταβλητή comparisons) για τις αναζητήσεις με βάση το id που είναι διαθέσιμες. Παράλληλα, για την καταμέτρηση συγκρίσεων και χρόνου στην επιλογή 7(αναζήτηση με βάση το επίθετο) , προκειμένου να υλοποιηθεί τυχαιότητα στο αναζητούμενο επίθετο, το vector του searchInput δεικτοδοτείται με το εξής : `searchTerm = searchInput->myVector[k]->Employees[k % (searchInput->myVector[k]->Employees.size())]->lastName;` Όπου `searchTerm` το αναζητούμενο επίθετο σε κάθε επανάληψη.

Οι μετρήσεις έγιναν 10 φορές με σύνολα μεγέθους 1500 και οι συνολικές συγκρίσεις και χρόνοι μαζί με τους μέσους όρους για κάθε μοναδική αναζήτηση φαίνονται στους παρακάτω πίνακες :

(* Τα τυχαία αρχεία που περιέχουν τις αναζητούμενες εταιρίες βρίσκονται στον φάκελο Είσοδοι στο Πρόγραμμα)

Χρόνοι (sec) για αναζητήσεις με βάση το id :

	Γραμμική Αναζήτηση	Δυαδική Αναζήτηση	Binary Interpolation Search	AVL
Είσοδος 1	0.761	2.868	0.040	0.001
Είσοδος 2	0.740	2.832	0.040	0.001
Είσοδος 3	0.732	2.842	0.040	0.001
Είσοδος 4	0.730	2.843	0.043	0.001
Είσοδος 5	0.721	2.839	0.043	0.001
Είσοδος 6	0.751	2.866	0.040	0.001
Είσοδος 7	0.730	2.829	0.040	0.001
Είσοδος 8	0.730	2.840	0.030	0.001
Είσοδος 9	0.759	2.864	0.047	0.001
Είσοδος 10	0.721	2.845	0.040	0.001
ΜΟ	0.000492	0.001898	0.000027	0.000001

Συνολικές συγκρίσεις για αναζητήσεις με βάση το id :

	Γραμμική Αναζήτηση	Δυαδική Αναζήτηση	Binary Interpolation Search	AVL
Είσοδος 1	7207527	47346	22006	48683
Είσοδος 2	7068921	47177	22051	48604
Είσοδος 3	7009716	47373	22083	48508
Είσοδος 4	7040780	47458	21856	48326
Είσοδος 5	6852107	46941	21813	48369
Είσοδος 6	7085493	47363	21869	48636
Είσοδος 7	6944313	47354	21901	48563
Είσοδος 8	7015631	47531	21887	48601
Είσοδος 9	7212308	47255	22001	48744
Είσοδος 10	6854525	47289	21730	48532
ΜΟ	4686.1	31.5	14.6	32.4

Χρόνοι (sec) για αναζητήσεις με βάση το επώνυμο :

	Γραμμική Αναζήτηση	Ψηφιακό Δεντρο
Είσοδος 1	38.530	0.002
Είσοδος 2	37.622	0.001
Είσοδος 3	37.652	0.001
Είσοδος 4	37.503	0.002
Είσοδος 5	36.675	0.001
Είσοδος 6	37.799	0.001
Είσοδος 7	36.646	0.001
Είσοδος 8	37.184	0.001
Είσοδος 9	38.260	0.001
Είσοδος 10	37.742	0.001
ΜΟ	0.025041	0.000001

Συνολικές Συγκρίσεις για αναζητήσεις με βάση το επώνυμο :

	Γραμμική Αναζήτηση	Ψηφιακό Δεντρο
Είσοδος 1	67210500	9147
Είσοδος 2	67210500	9132
Είσοδος 3	67210500	9033
Είσοδος 4	67210500	9025
Είσοδος 5	67210500	9213
Είσοδος 6	67210500	9024
Είσοδος 7	67210500	9126
Είσοδος 8	67210500	8993
Είσοδος 9	67210500	9072
Είσοδος 10	67210500	9126
ΜΟ	44807.0	6.1

<u>Ανάλυση Αποτελεσμάτων :</u>

Γενικά ισχύει ότι μια συνάρτηση T(n) είναι O(f(n)) αν υπάρχουν σταθερές c>0 και $n_o \geq 0$ τέτοιες ώστε για κάθε n>= n_o να ισχύει T(n) ≤ c*f(n). (1)

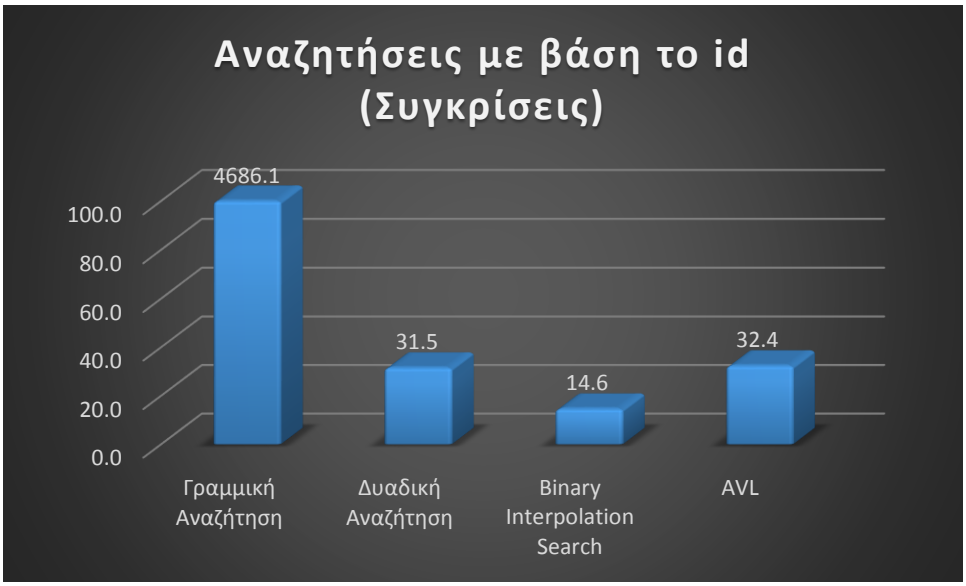
- Αναζητήσεις με βάση το id :
  - Γραμμική Αναζήτηση : Η θεωρία δίνει O(n) το οποίο και επιβεβαιώνεται καθώς : T(n) ≤ c*f(n) → 4686.1≤ c*10.000 στο οποίο αν θέσουμε c=1 προκύπτει 4686.1≤ 10.000 που ισχύει.
  - Δυαδική Αναζήτηση : Η θεωρία δίνει O(logn) το οποίο επιβεβαιώνεται καθώς :

$T(n) \leq c*f(n) \rightarrow 31.5 \leq c*\log(10.000) \rightarrow 31.5 \leq c*13.28$ στο οποίο αν θέσουμε $c=2.4$ προκύπτει $31.5 \leq 31.87$ που ισχύει.

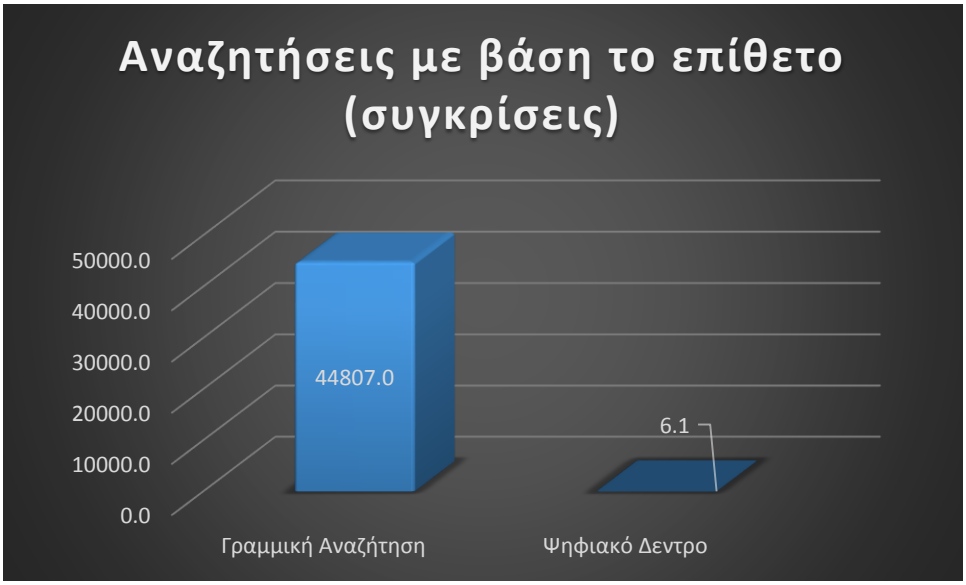
- Δυαδική Αναζήτηση Παρεμβολής : Η θεωρία δίνει O(loglogn) το οποίο και επιβεβαιώνεται καθώς : $T(n) \leq c*f(n) \rightarrow 14.6 \leq c*3.73$ στο οποίο αν θέσουμε $c=4$ προκύπτει $14.6 \leq 14.92$ που ισχύει.
- Αναζήτηση σε AVL δέντρο : Η θεωρία δίνει O(logn) το οποίο και επιβεβαιώνεται καθώς : $T(n) \leq c*f(n) \rightarrow 32.4 \leq c*13.28$ στο οποίο αν θέσουμε $c=2.5$ προκύπτει $32.4 \leq 33.2$ που ισχύει.

- Αναζητήσεις με βάση το επώνυμο :
  - Γραμμική Αναζήτηση : Σε αυτή την αναζήτηση , με δεδομένο ότι ένα επώνυμο μπορεί να εμφανίζεται παραπάνω από μια φορές , το πρόγραμμα που δημιουργήθηκε προσπελαυνει όλες τις εταιρίες μια προς μια και συνεπώς ο αριθμός των συγκρίσεων κάθε φορά είναι σταθερός. Το 44807 που προκύπτει σαν μέσος όρος είναι ο αριθμός του υπαλλήλων που υπάρχουν στο data.csv και άρα είναι προφανές ότι ισχύει O(n).
  - Αναζήτηση στο Ψηφιακό Δέντρο : Η θεωρία δίνει $O(\log_k n)$ ($k = 26$ (όσο το αγγλικό αλφάβητο) και $n=44807$(συνολικοί υπάλληλοι))το οποίο και επιβεβαιώνεται καθώς : $T(n) \leq c*f(n) \rightarrow 6.1 \leq c*3.28$ στο οποίο αν θέσουμε $c=2$ προκύπτει $6.1 \leq 6.56$ που ισχύει.

Η συσχέτιση όλων των παραπάνω φαίνεται στα παρακάτω γραφήματα στα οποία έχει τροποποιηθεί κατάλληλα η κλίμακα στον κατακόρυφο άξονα προκειμένου να είναι προφανής η διαφορά.




Αναζητήσεις με βάση το id (sec)
0.002000
0.001500
0.001000
0.000500
0.000000
0.000492
0.001898
0.000027
0.000001
Γραμμική Αναζήτηση
Δυαδική Αναζήτηση
Binary Interpolation Search
AVL


Αναζητήσεις με βάση το επίθετο (συγκρίσεις)
50000.0
40000.0
30000.0
20000.0
10000.0
0.0
44807.0
6.1
Γραμμική Αναζήτηση
Ψηφιακό Δεντρο


Αναζητήσεις με βάση το επίθετο (sec)
0.025000
0.020000
0.015000
0.010000
0.005000
0.000000
0.025041
0.000001
Γραμμική Αναζήτηση
Ψηφιακό Δεντρο