AM : 2017030021 ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΙΩΑΝΝΗΣ-ΙΑΣΩΝ ΓΕΩΡΓΑΚΑΣ

ΑΝΑΦΟΡΑ 3ης ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

ΥΛΟΠΟΙΗΣΗ-ΣΚΟΠΟΣ ΑΣΚΗΣΗΣ

Για την υλοποιήση της 3η εργαστηριακής δημιούργησα 6 διαφορετικά πακέτα ένα για την main δηλαδή το εκτελέσιμο αρχείο και άλλα 5 ένα για κάθε δομή δεδομένων. Οι δομές δεδομένων είναι ένα BST, separate Chaining, δύο linear hashing με διαφορετικές υλοποιήσεις η μία έχει έναν παρπάνω περιορισμό ως προς το πότε θα γίνεται split και ένα multi-way search tree . Στη κλάση main επιδεικνύω τη λειτουργία της κάθε δομής εισάγωντας στην καθέ δομή 1.000,10.000,30.000,50.000, 70.000,100.000 κλειδία με τη σειρά και στη συνέχεια αναζητώ 30 κλειδία που υπάρχουν(επιτυχής αναζήτηση).Στο project η έκδοση του eclipse που χρησιμοποιήσα ειναι jdk 1.8.0_161. Ο σκοπός της άσκησης ήταν εξοικείωση με δομές κατακερματισμού και να αποδείξει ότι οι δυναμικές δομές κατακερματισμού έχουν καλύτερη απόδοση σε σύγκριση με στατικές δομές και με δένδρα έρευνας, για αναζητήσεις τυχαίας τιμής στην κεντρική μνήμη.

ΚΑΤΑΣΚΕΥΗ ΔΟΜΩΝ ΔΕΔΟΜΕΝΩΝ

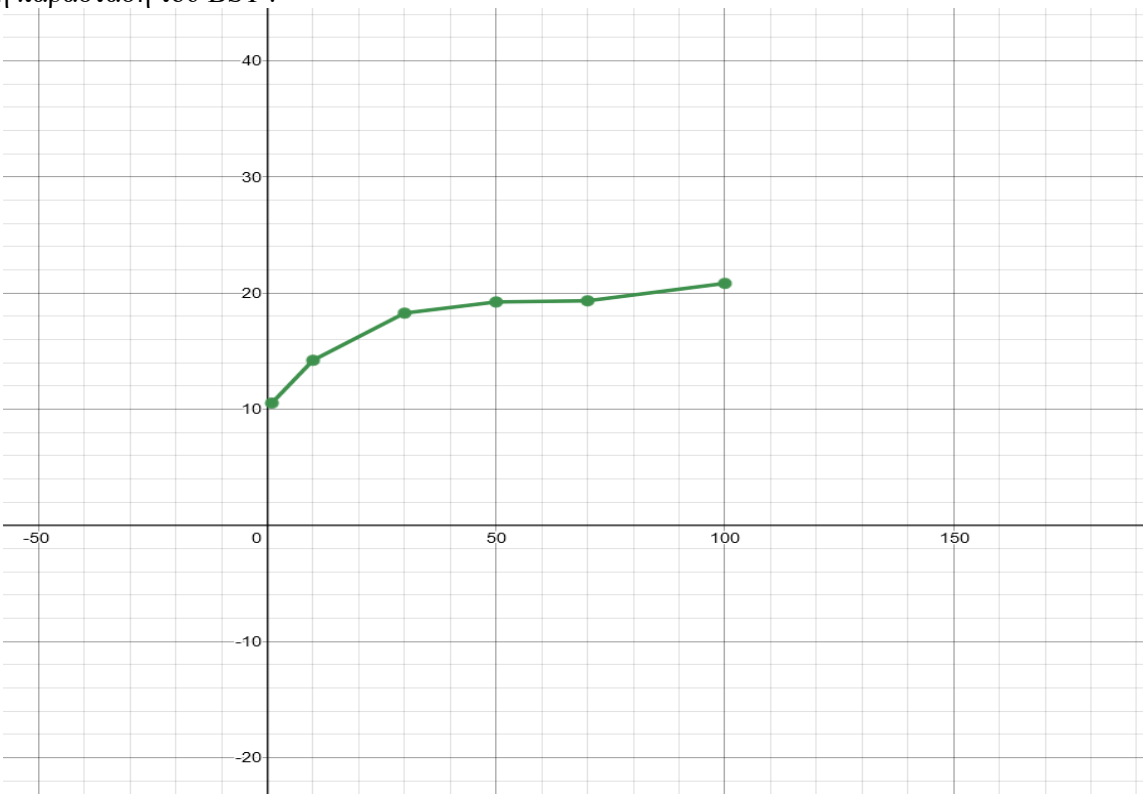
Για την κατασκευή των δομών χρησιμοποιήσα έτοιμους κώδκες τόσο απο το internet όσο και απο τις σημειώσεις του μαθήματος,οι πηγές άντλησης των κωδίκων αναφέρονται στο τέλος της παρούσας αναφοράς , προσθέτοντας στις κατάλληλες συναρτήσεις την μέτρηση των συγκρίσεων. Ως σύγκριση θεωρώ τη σύγκριση μεταξύ του κλειδιού που ψάχνω και κλειδιών που βρίσκονται στην εκάστοτε δομή . Ακόμα στην δεύτερη υλοποιήση της linear hashing με τον επιπλέον περιορισμό σχετικά με το πότε θα κάνει split χρησιμοποίησα μία ακόμα συνάρτηση για να ελέγχω αν μία (οποιαδήποτε) αλυσίδα σελίδων υπερχείλισης γίνει μεγαλύτερη από 2 (`IsLengthChainOfOverflowBuckets`).

Συγκρίσεις-Εκτιμίσεις απόδοσης

Data(k)	1	10	30	50	70	100
Binary search Tree	10.53	14.21	18.27	19.23	19.33	20.83
Hashing with separate chaining	1.43	6.4	13.87	27.13	36.2	61.5
Linear Hashing	5.4	3.43	3.8	3.07	4.3	3.93
Linear Hashing with chain of Overflows	3.83	3.13	2.73	2.57	3.53	3.03
BTree	34.03	69.4	66.93	97.53	80.77	90.2

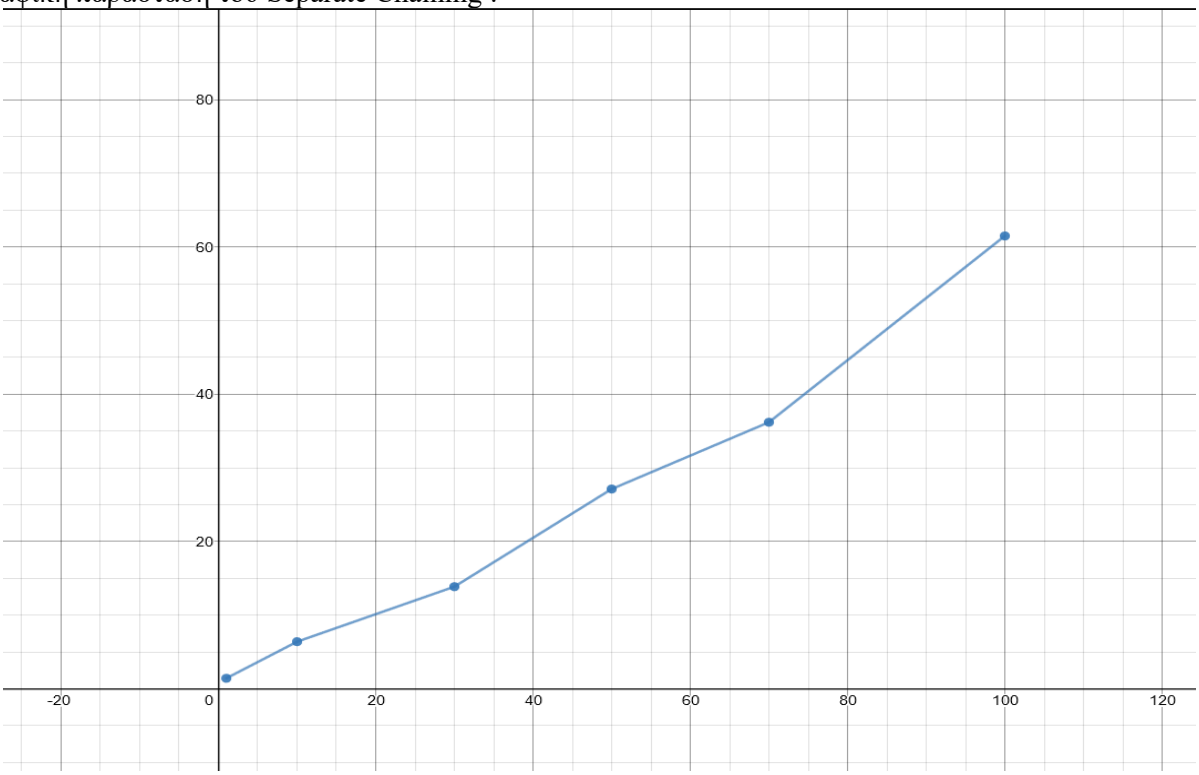
Για να συγκρίνω και να εκτιμήσω την απόδοση των δομών δεδομένων δημιούργησα στο πακέτο main ένα εκτελέσιμο αρχείο(κλάση main). Αρχικά δημιουργώ ένα instance της κάθε δομής δεδομένων,παράγω με την συνάρτηση generatekeysForInsertion M τυχαίους αριθμούς. Ταυτόχρονα τα αποθηκεύω σε ένα πίνακα μεγέθους M έτσι ώστε να τα χρησιμοποιήσω στην επιτυχή αναζήτηση. .Στη συνέχεια εισάγω στις δομές δεδομένων τους αριθμούς ,που βρίσκονται στον αντίστοιχο πίνακα που τα έχω αποθηκεύσει, με την αντίστοιχη συνάρτηση `insertKeysInDataStructures.`.Μετά κάνω 30 επιτυχημένες αναζητήσεις χρησιμοποιώντας την αντίστοιχη search κάθε δομής.Τέλος όλο ο κώδικας που περιγράφτηκε περιβάλεται από ένα for loop που εκτελείται 6 φορές κάθε φορά για διαφορετικό αριθμό δεδομένων M = 1.000,10.000,30.000,50.000,70.000,100.000 .

Γραφικη παράσταση του BST :



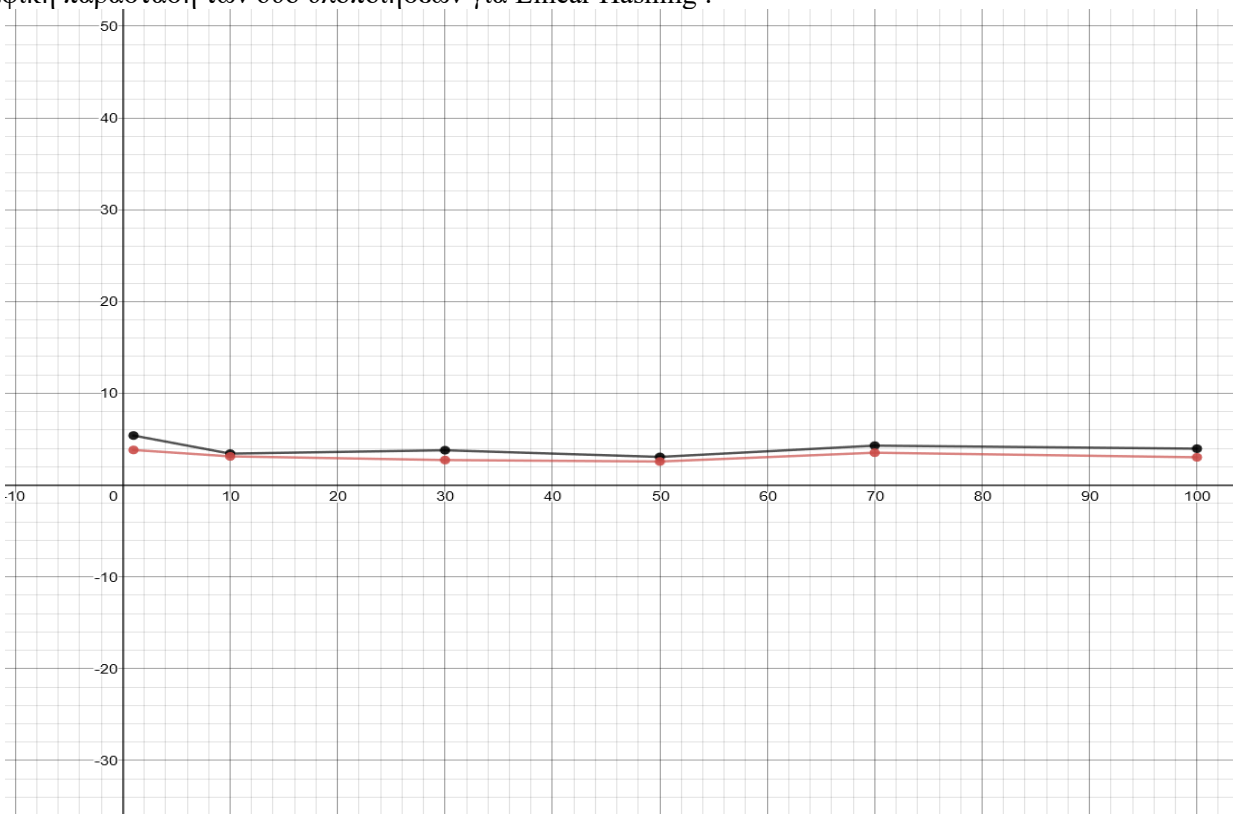
Ο οριζόντιος άξονας χ είναι για τον εκάστοτε αριθμό των δεδομένων και ο κατακόρυφος άξονας ψ είναι ο μέσος όρος αριθός συγκρίσεων.

Γραφικη παράσταση του Separate Chaining :



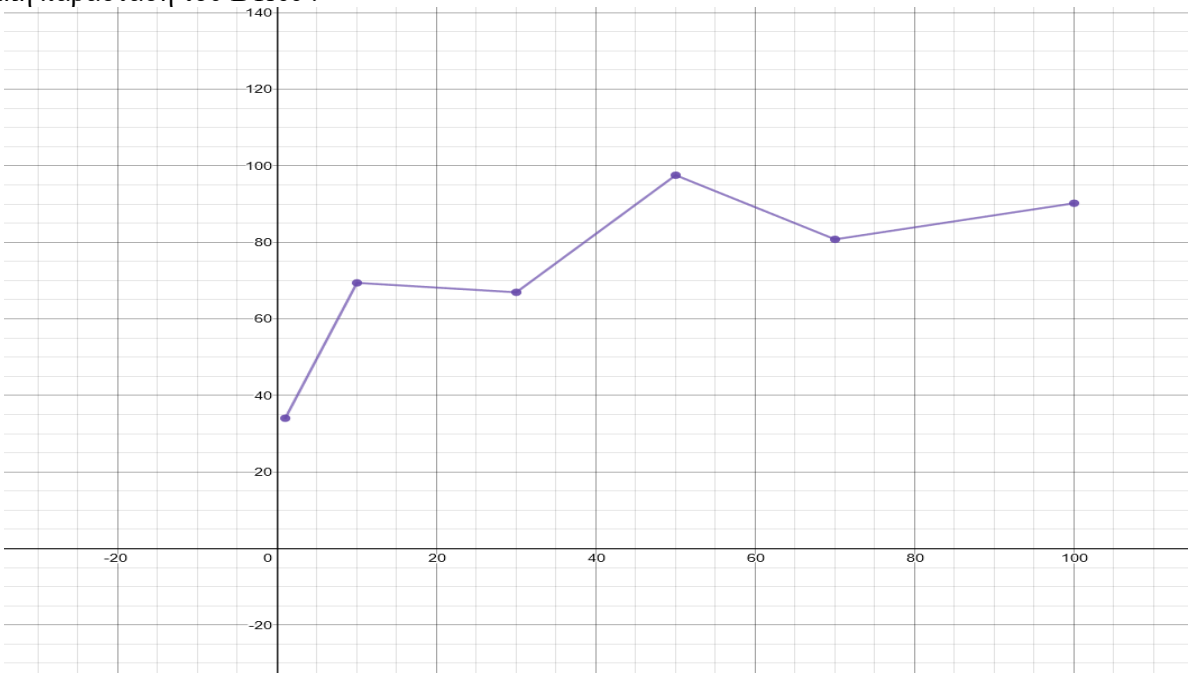
Ο οριζόντιος άξονας χ είναι για τον εκάστοτε αριθμό των δεδομένων και ο κατακόρυφος άξονας ψ είναι ο μέσος όρος αριθός συγκρίσεων.

Γραφικη παράσταση των δύο υλοποιήσεων για Linear Hashing :



Ο οριζόντιος άξονας χ είναι για τον εκάστοτε αριθμό των δεδομένων και ο κατακόρυφος άξονας ψ είναι ο μέσος όρος αριθός συγκρίσεων.

Γραφικη παράσταση του BTree :



Ο οριζόντιος άξονας χ είναι για τον εκάστοτε αριθμό των δεδομένων και ο κατακόρυφος άξονας ψ είναι ο μέσος όρος αριθός συγκρίσεων.

ΤΕΚΜΗΡΙΩΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Για το BST : Στο δυαδικό δένδρο έρευνας παρατηρώ ότι όσο αυξάνονται τα δεδομένα, αύξανεται και ο μέσος αριθμός συγκρίσεων για να βρει το στοιχείο .Αυτό συμβαίνει δίοτι κάθε κόμβος έχει δύο παιδία εφόσον είναι δυαδικό, άρα όταν περιέχει το δένδρο πιο πολλά στοιχεία θα έχει και μεγαλύτερο βάθος.Άρα για την έυρεση του εκάστοτε στοιχείου θα χρειάζονται πιο πολλές συγκρίσεις. Η έκφραση πολυπλοκότητας που αντιστοιχεί στην καμπύλη που εχει παρατεθεί παραπάνω είναι $O(\log_2 N)$ η οποία αντιστοιχεί στην average case ,με το $\log_2 N$ να έχει ως βάση του το 2 καθώς κάθε κόμβος έχει δύο παιδία και το N είναι ο αριθμός των στοιχείων που έχει το δένδρο.Στη best-case είναι O(1) ενώ στη worst-case O(N).

Για το SEPARATE CHAINING : Στον κατακερματισμό χωριστών αλυσίδων με μέγεθος 1000 παρατηρώ ότι ο μέσος όρος αριθμός συγκρίσεων για να βρει το κλειδί με τα λιγότερα δεδομένα(M = 1000) είναι αρκετά χαμηλός περίπου 1.5 ενώ με τα περισσότερα(M =100000) έχει αυξηθεί αρκετά της τάξης των 50 . Αυτό συμβαίνει διότι αφού η δομή του separate chaining έχει μέγεθος 1000 δηλαδή 1000 θέσεις στο hashTable τότε σημαινεί οτι όσο τα στοιχεία παραμένουν λιγότερα απο 1000 τότε η πιθανότητα δύο ή περισσότερα στοιχεία να είναι στην ίδια θέση του hashTable(δηλαδή να έχουμε collisions), δημιουργώντας μια λίστα στη συγκεκριμένη θέση και κατα συνέπεια θα έπαιρνε πιο πολλές συγκρίσεις για να βρουμε το στοιχείο, είναι αρκετα μικρή. Ενώ όσο αυξάνονται ο αριμθός των στοιχείων τόσο θα αυξάνεται και η πιθανότητα σε κάθε θέση του HashTable η λίστα με τα στοιχεία να είναι μεγαλύτερη και κατά συνέπεια θα αυξάνεται και ο αριθμός των συγκρίσεων που απαιτείται για την έυρεση του εκάστοτε στοιχείου. Η έκφραση πολυπλοκότητας που αντιστοιχεί στην γραφική παράσταση που εχει παρατεθεί παραπάνω είναι O(N) η οποία αντιστοιχεί στη worst-case και στην average ,όπου N έιναι ο αριθμος των στοιχείων,καθώς κάθε θέση στο hashTable είναι μια λίστα και συγκρίνουμε με κάθε στοιχείο της εκάστοτε λίστας ενώ O(1) είναι στη best-case. Αυτό επιβεβαιώνεται και πειραματικά απο την παραπάνω γραφική αλλά και απο το γεγονός ότι η καλύτερη περίπτωση είναι να το βρεί στη πρώτη θέση της λίστας στην εκάστοτε θέση του hashTable δηλαδη κατευθείαν, σε σταθερό χρόνο. Οι άλλες δύο περιπτώσεις(worst, average case) έχουν να κάνουν με το μήκος της λίστας στην αντίστοιχη θέση του hashTable δηλαδή με τον αριθμό των δεδομένων.

Για τα LINEAR HASHING : Για του γραμμικο κατακερματισμό τόσο με κριτήριο διάσπασης u>80% όσο και με κριτήριο διάσπασης u>80% ή το μήκος μίας αλυσίδας σελίδων υπερχείλισης να είναι μεγαλύτερο από 2 παρατηρώ ότι ο μέσος όρος αριθμών συγκρίσεων κυμαίνεται από 3.07 μέχρι 5.04 και στη δεύτερη υλοποίηση απο 2.57 μεχρι 3.83 δηλαδή παρατηρούμε ότι η απόδοση του γραμμικού κατακερματισμού είναι περίπου η ίδια και για πολύ μεγάλα δεδομένα και για πολύ μικρά δεδομένα. Αυτό συμβαίνει διότι το linear hashing είναι μια δομή για δυναμικό hashing δηλαδή κατακερματισμό που θα έχει περίπου την ίδια απόδοση τόσο για πολλα στοιχεία όσο και για λίγα. Επίσης παρατηρώ ότι στη δεύτερη υλοποιήση του linear hashing ο μέσος όρος των συγκρίσεων είναι ελαφρώς μικρότερος απο τον αντίστοιχο της πρώτης υλοποιήσης λόγω του επιπλέον περιορισμού σχετικά με το πότε γίνεται split. Η έκφραση πολυπλοκότητας του linear hashing και για τις δύο υλοποιήσεις είναι O(1) αυτό επιβεβαιώνεται και πειραματικά καθώς ο αριθμός των συγκρίσεων είναι καθε φορά πολύ κοντά κάτι το οποίο φαίνεται και στην αντίστοιχη γραφική παράσταση.Επίσης η έκφραση πολυπλοκότητας για τις δύο υλοποιήσεις επιβεβαιώνεται και απο το γεγονός ότι αφού είναι δυναμικός κατακερματισμός σημαίνει ότι τα δεδομένα εισάγωνται και αφαιρουνται δυναμικά μειώνοντας τα overflow bucket , για αυτό αλλωστε και δεν παρατηρούμε μεγάλες αλλαγές μεταξυ των δύο υλοποιήσεων του linear hashingως προς την απόδοση τους. Έτσι βρίσκουμε το κλειδί στην αντίστοιχη θέση του στο hashTable χωρίς να πάμε συνήθως σε overflow bucket άρα σε σταθερό χρόνο.

Για τα multi-way search tree: Για το δένδρο πολλαπλών διαδρομών παρατηρώ οτι ο αριθμός των συγκρίσεων κυμαίνεται από 34.03 μεχρί 97.53 και έχει μια ανοδική τάση μαζι με κάποια μικρές αυξομειώσεις.Ο μεγάλος αριθμός συγκρίσεων που εμφανίζεται οφείλεται στο γεγονός ότι για την ανζήτηση σε κάθε κόμβο έχω χρησιμοποιήσει γραμμική αναζήτηση. Η έκφραση πολυπλοκότητας που αντιστοιχεί στο multi-way search tree είναι $O(\log_m N)$,όπου N ο αριθμός των στοιχείων και m η τάξη του δένδρου, αυτό επιβεβαιώνεται και πειραματικά απο την αντίστοιχη γραφική παράσταση και είναι το average-Case. Ενώ για τη best-case είναι O(1) και για την worst-case είναι O(N).

Γενικά η πιο γρήγορη δομή απο αυτές που αναφέρθηκαν παραπάνω είναι το linear hashing ως dynamic hashing αφού η έκφραση πολυπλοκότητας του είναι O(1) και είναι πάντοτε σταθερό με μικρές αποκλίσεις ανεξαρτήτως του όγκου των δεδομένων.Αυτό επιβεβαιώνεται και πειραματικά από τα αποτελέσματα του παραπάνω πίνακα. Συμπερασματικά μπορούμε να καταλάβουμε ότι οι δυναμικές δομές κατακερματισμού έχουν καλύτερη απόδοση απο τις αντίστοιχες στατικές και απο τα δέντρα(BST,Btree) γιατί ειναι ανεξάρτητα απο τον όγκο των δεδομένων και έχουν σχεδόν σταθερή απόδοση.

ΒΙΒΛΙΟΓΡΑΦΙΑ-ΠΗΓΕΣ:

- Για το BST ο κώδικας αντλήθηκε απο τις διαλέξεις του κ. Λαγουδάκη απο το μαθημα του Δομημενου Προγραμματισμού του εαρινού εξαμήνου 2017-2018.
- Για το Separate Chaining αντλήθηκε απο το http://www.java2s.com/Code/Java/Collections-Data-Structure/Hashtablewithseparatechaining.htm?fbclid=IwAR0h0iCVm7yhsj6Fi75V9mlp6_pD0fPVpDyIXadLIQ5fkUY1PH5fXK73ams
- Για το linear hashing ο κώδικας αντλήθηκε απο το φροντιστήριο του μαθήματος δομές Δεδομένων και αρχείων του εαρινού εξαμήνου 2018-2019.
- Για το multi-way search tree χρησιμοποιήθηκε ενα Btree του οποίος ο κώδικας αντλήθηκε απο https://gist.github.com/adderllyer/3bfa2d04200386b5664c?fbclid=IwAR3RipXz7IXditTUUbDKbdyd0t17i8NIcKXMQ9Zv7UJS25oXRKVETK-BVmk