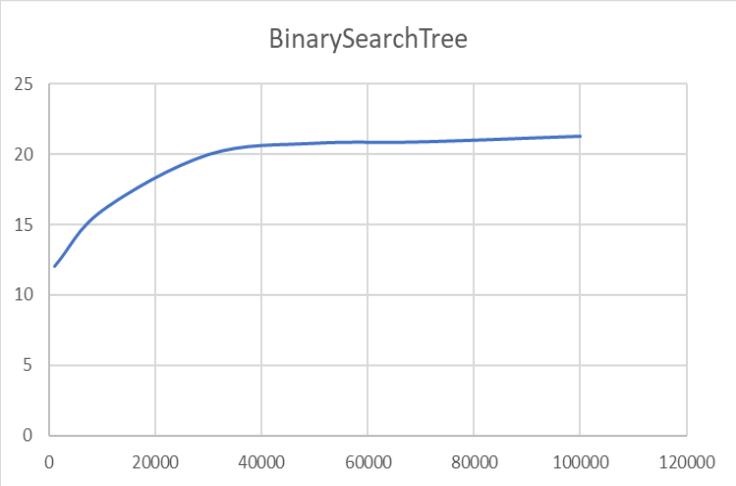
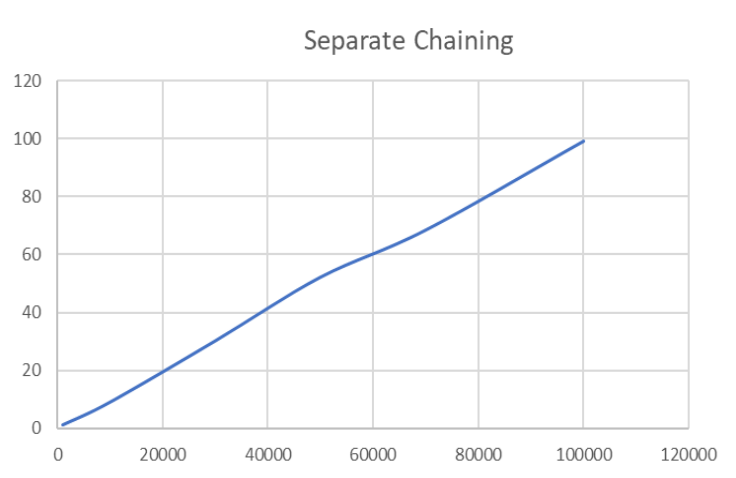
3Η ΑΣΚΗΣΗ ΑΝΑΦΟΡΑ

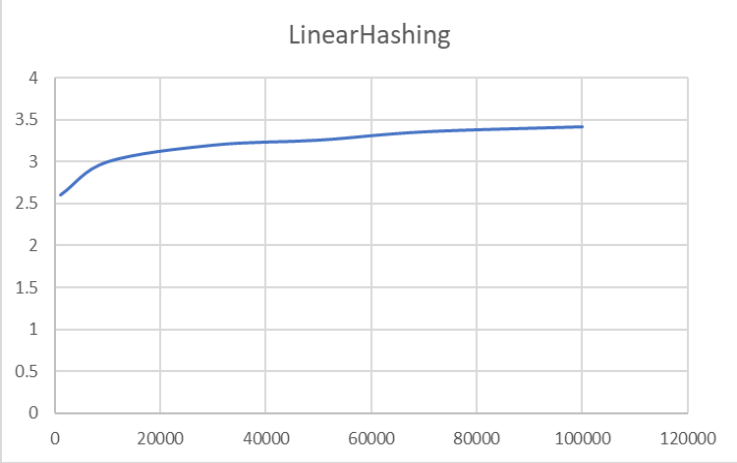
Αναστασίου Μιχάλης 2017030185

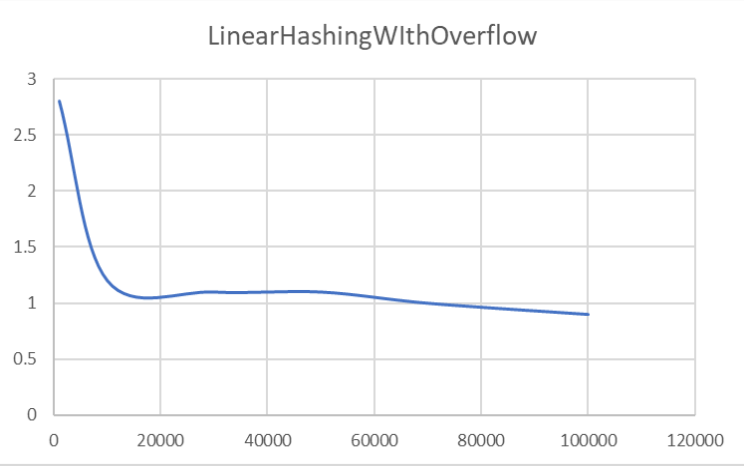
Εκτίμηση – Σύγκριση Απόδοσης

Όπως μας ζητήθηκε από την άσκηση κατασκεύασα 5 γραφικές παραστάσεις για 30 συγκρίσεις κλειδιών με τον αριθμών στοιχείων που μας έδινε η άσκηση για όλες τις δομές κατακερματισμού. Τα αποτελέσματα φαίνονται πιο κάτω.









Τεκμηρίωση Αποτελεσμάτων

Binary Search Tree:

Το binary Search Tree σύμφωνα με της γραφικές μου παραστάσεις έχει την 3η καλύτερη απόδοση από της 5 μεθόδους. Η έκφραση πολυπλοκότητας που το εκφράζει είναι log(n).

Η απόδοσή της είναι πλήρως δικαιολογημένη αφού κάθε φορά μοιράζει το πλήθος στοιχείων δια 2 σύμφωνα με αν το κλειδί είναι μεγαλύτερο η μικρότερο. Έτσι δεν χρειάζεται να κάνει πολλές συγκρίσεις για να βρει ή να μην ένα κλειδί.

Separate Chaining:

Αύτη η μέθοδος όπως φέρνετε είναι η χειρότερη από τις υπόλοιπες. Η έκφραση πολυπλοκότητας τής μεθόδου όπως φαίνεται ξεκάθαρα είναι O(N). Η κακή απόδοση αυτής της μεθόδου για μεγάλο πλήθος στοιχείων είναι δικαιολογημένη αφού ο πίνακας κατακερματισμού έχει μέγεθος 1000 , με αποτέλεσμα όταν το κλειδί 2 καταχωρήσεων είναι ίδιο να δημιουργούνται τεράστιες λίστες λίστες. Οπόταν για μεγάλο πλήθος στοιχείων είναι αναμενόμενο να έχουμε μια μεγάλη λίστα από αριθμούς με το ίδιο Key με αποτέλεσμα να χρειάζονται τόσες πολλές συγκρίσεις.

Linear Hashing:

Σε αύτη την μέθοδο παρατηρούμε πολύ καλά αποτελέσματα αφού και για μικρές και για μεγάλες ποσότητες καταχωρήσεων έχουμε πολύ λίγες συγκρίσεις. Η έκφραση πολυπλοκότητας για αυτήν την μέθοδο είναι log(1). Ομοίως η απόδοση αυτής της μεθόδου είναι δικαιολογημένη αφού δεν χρησιμοποιά μόνο ένα πίνακα κατακερματισμού αλλά “buckets”. Ακόμη έχει την συνάρτηση “h(key)=key%100” που κάθε φορά τροποποιείτε σύμφωνα με τα splits που έκανε και τέλος έχουμε και load factor που με αυτόν τον τρόπο κάνει split περισσότερες φορές έτσι έχουμε αυτό το αποτέλεσμα.

Linear Hashing Overflow:

Η έκφραση πολυπλοκότητας για αύτην την γραφικής είναι O(1). Αύτη η συνάρτηση είναι η ίδια με την παραπάνω με μόνη διαφορά ότι διασπάτε κάθε φόρα που μια αλυσίδα υπερχείλισης είναι μεγαλύτερη από το 2. Για αυτόν τον λόγο η απόδοση της είναι δικαιολογημένη αφού για μικρό αριθμό καταχωρήσεων έχουμε τα ίδια αποτελέσματα με την απλή Linear Hashing αφού δεν δημιουργείτε Overflow, όμως για μεγαλύτερο αριθμό καταχωρήσεων δημιουργούνται περισσότεροι πίνακες , οπόταν τελικά τόσο λίγες μετρήσεις. Αυτή η μέθοδος κάνει τις λιγότερες συγκρίσεις αλλά όταν βάλουμε ένα τεράστιο αριθμό στοιχείων θα χρειαστεί τεράστιο χώρο και χρόνο αφού θα πρέπει συνεχώς να κάνει split.

Btree:

Η άσκηση μας ζήτησε να υλοποιήσουμε multiway tree αλλά κατά την διάρκεια του μαθήματός μας είπατε ότι είναι εντάξει να υλοποιήσουμε btree καθώς είναι το ίδιο πράγμα απλά το btree βάζει τα στοιχεία ταξινομημένα. Αυτή η μέθοδος κάνει αρκετές συγκρίσεις και η έκφραση πολυπλοκότητας είναι log(N). Τέλος η απόδοση αυτής της μεθόδου είναι επίσης δικαιολογημένη αφού το δένδρο είναι τάξης 100 άρα έχει 99 στοιχεία μέσα. Αν κάποιο στοιχείο είναι προς το τέλος δηλαδή του εύρους τιμών θα πρέπει να κάνει πολλές συγκρίσεις κλειδιών.

ΥΛΟΠΟΙΗΣΗ ΚΩΔΙΚΑ

BinaryTree:

Αρχικά για την υλοποίηση του BinaryTree δεν υπήρχε ιδιαίτερη δυσκολία αφού την ξανά υλοποιήσαμε με παρόμοιο τρόπο. Απλά συγκρίνει το στοιχείο με τα παιδία αν είναι μικρότερο η μεγαλύτερο μέχρι να βρει την κατάλληλη θέση που πρέπει να μπει. Για την search ακολουθείτε η ίδια λογική. Τέλος για την μέτρηση συγκρίσεων τοποθέτησα ένα counter στη συνάρτηση search όπως έκανα για τις υπόλοιπες μεθόδους που μετράει κάθε φόρα πόσες συγκρίσεις έκανε για να βρει η όχι ένα αριθμό.

Separate Chaining:

Για αύτη την μέθοδο αρχικά υλοποίησα ένα array list όπου είναι ο πίνακας κατακερματισμού μου. Έπειτα κάθε φορά για την εισαγωγή του στοιχείου έλεγχε το στοιχείο με την συνάρτηση που μας δόθηκε h(key)=key%1000. Αναλόγως τοποθετά το στοιχείο στην θέση του αν δεν είναι καταλαλημένη ή δημιουργία μια άλλη λίστα με parent την θέση που έπρεπε να τοποθετηθεί το στοιχείο. Για την εύρεση του στοιχειού κάνει την συνάρτηση πηγαίνει στην θέση και κάνει συγκρίσεις μέχρι να βρει η όχι το στοιχείο.

Linear Hashing , Linear Hashing Overflow:

Για αυτή την μέθοδο βασίστηκα στην υλοποίηση που μας ανεβάσατε. Έκανα τις απαραίτητες τροποποιήσεις όπως το μέγεθος του πίνακα και τον αριθμό των στοιχείων που μπορεί να εμπεριέχει. Αυτός ο κώδικας καταχωρεί τα στοιχεία και κάθε φορά που το 80% των θέσεων έχει γεμίσει κάνει έλεγχο και έπειτα διασπάτε ο χώρος. Για το επόμενο μέρος της άσκηση έβαλα ένα counter και κάθε φόρα που δημιουργείτε μια σελίδα υπερχείλισης τον αυξάνω και ελέγχω αν έχει ξεπεράσει το 2 αν ναι τότε κάνω split το bucket.

Btree:

Αύτη η μέθοδος σαν βασική ιδέα είναι ότι αρχικά εισάγει 99 στοιχεία σε ένα Array όταν γεμίσει το Array εισάγει στοιχεία αναλόγως το εύρος τιμών τον 99 στοιχείων και όταν πάει να κάνει αναζήτηση εντοπίζει γρήγορα αν υπάρχει ή όχι το κλειδί ανάλογα με το εύρος τιμών των Entries.

ΠΗΓΕΣ ΠΛΗΡΟΦΟΡΗΣΗΣ & ΕΤΟΙΜΑ ΠΡΟΓΡΑΜΜΑΤΑ:

Btree:

http://www.jbixbe.com/doc/tutorial/BTree.html

https://www.geeksforgeeks.org/data-structure-gq/b-and-b-trees-gq/

LinearHashing:

Σημειώσεις μαθήματος.

SeparateChaining:

https://www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/

BinaryTree:

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/