Γιάννης Περίδης ,ΗΜΜΥ

2018030069

Δομές Δεδομένων και Αρχείων ,3ο project
Αναφορά αξιολόγησης αποτελεσμάτων:

Σχεδιάζουμε τον παρακάτω πίνακα με σκοπό την αναλυτική αξιολόγηση της απόδοσης των αποτελεσμάτων του προγράμματός μας:

Πίνακας αποτελεσμάτων πειράματος για N=10^4

Input size(N)	**Insertion u>50%**	**Search u>50%**	**Deletion u>50%**	**Insertion u>80%**	**Search u>80%**	**Deletion u>80%**	**BST insertion**	**BST search**	**BST deletion**
number of comparisons	829.56	24.96	18.36	353.42	23.66	27.72	56.56	47.18	58.92

Η δειγματοληψία των κλειδιών, έγινε ανά εκατοντάδες. Παρακάτω, θα σημειωθούν οι σχετικές παρατηρήσεις που εντοπίστηκαν από το αποτέλεσμα του πειράματος που αφορούν τον τρόπο που μεταβάλλεται ο αριθμός των συγκρίσεων σε κάθε μια από τις τρις διαδικασίες(insertion,search,deletion) σε σχέση με τον αριθμό N ο οποίος συμβολίζει το πλήθος αριθμών στο αρχείο.

INSERTION:

Για την διαδικασία του insertion με υλοποίηση linear hashing θεωρητικά αναμένουμε και για τις δύο περιπτώσεις όπου max load factor=0,5 και 0,8 πως η πολυπλοκότητα θα ήταν O(1), εξαιρουμένης της χειρότερης περίπτωσης αυτών όπου τα περισσότερα hash buckets θα ήταν σε κατάσταση overflow, που τότε θα βλέπαμε πολυπλοκότητα O(n). Πράγμα το οποίο δεν συμβαίνει στο πρόγραμμά μας. Παρόλα αυτά , η παραπάνω θεωρητική αντιμετώπιση απορρίπτεται καθώς και ισχύει μονάχα όταν το hash map που χρησιμοποιούμε είναι στατικό. Η κύρια αιτία που καθυστερεί η διαδικασία, οφείλεται στην μέθοδο split η οποία αυξάνει δραματικά τον αριθμό συγκρίσεων , όταν καλείται .Ακόμη, σχετικά με την μέθοδο split παρατηρούμε πως επειδή όταν έχουμε max load factor=0,5 σε σχέση με το 0,8 , οι πιθανότητες να υπάρξει split αυξάνονται και συνεπώς η υλοποίηση με max load factor=0,8 κάνει κατά μέσο όρο λιγότερες συγκρίσεις ανά εισαγωγή. Όσον αφορά το binary search tree, θεωρητικά αναμέναμε να έχει πολυπλοκότητα O(logN), πράγμα που επιτυγχάνεται σε έναν ικανοποιητικό βαθμό και πειραματικά. Τέλος, περιμέναμε πως το bst θα ήταν γρηγορότερο εφόσον για τον λόγο που προαναφέρθηκε το linear hashing θα έχει O(n) που είναι χειρότερη από την O(logN).

SEARCH:

Για την διαδικασία του search με υλοποίηση linear hashing σε αντίθεση με το insertion, που όπως προαναφέρθηκε επιβραδύνεται λόγο του split,αυτό δεν επηρεάζεται και επομένως η θεωρητική τιμή πολυπλοκότητας O(1) που περιμέναμε, προσεγγίζεται σε μεγάλο βαθμό και από τις πειραματικές τιμές. Επομένως ξανά σε αντίθεση με το Insertion, εφόσον η πολυπλοκότητα του binary search tree παραμένει παρόμοια με την θεωρητική της τιμή O(logn), είναι μεγαλύτερη από την O(1) τώρα το linear hashing θα είναι γρηγορότερο. Τέλος, φαίνεται πως η υλοποίηση με max load factor=0,5 σε σχέση με την 0,8 , πραγματοποιεί λιγότερες συγκρίσεις κατά μέσο όρο, πράγμα που περιμέναμε εφόσον το μεγαλύτερο load factor, θα κάνει λιγότερα splits που θα έχει ως αποτέλεσμα τα δεδομένα να είναι πιο πυκνά σε λιγότερα hash buckets.

DELETION:

Για την διαδικασία του deletion με υλοποίηση linear hashing επιβεβαιώνεται επίσης η θεωρητική πολυπλοκότητα O(1), καθώς επίσης παρατηρείται πως όσο αυξάνεται το N , ο αριθμός συγκρίσεων μας αυξομειώνεται διαρκώς. Όσον αφορά το binary search tree, ξανά επαληθεύεται η πολυπλοκότητα O(logn).Τέλος, παρατηρούμε πως το bst είναι αργότερο και από τις δύο linear hashing υλοποιήσεις, και πιο συγκεκριμένα η γρηγορότερη από τις τρις είναι αυτή με το μικρότερο max load factor , εφόσον όσο περισσότερα splits γίνονται τόσο πιο αραιωμένα είναι τα δεδομένα στα hash buckets.

ΣΥΜΠΈΡΑΣΜΑ:

Συμπερασματικά, παρατηρήσαμε πως για τις διαδικασίες search και deletion το Linear hashing είναι προτιμότερο καθώς και είναι πολύ πιο γρήγορο , ενώ για το insertion το bst εν τέλει λόγο των καθυστερήσεων των split, είναι πιο αποτελεσματικό .Τέλος, παρατηρήσαμε ακόμα πως ότι όσο πιο καλή εξάπλωση επιτύχουμε στο Hash map τόσο καλύτερη γίνεται και η απόδοση των αναζητήσεων .