Δομές Δεδομένων – Εργασία 4

ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

(**Όνομα:** Αναστάσιος Παπαπαναγιώτου, **ΑΜ:** 3200143, **Email:** p3200143@aueb.gr)

(**Όνομα:** Φοίβος Παπαθανασίου, **ΑΜ:** 3200138, **Email:** p3200138@aueb.gr)

a.

ΤΡΟΠΟΣ ΛΕΙΤΟΥΡΓΙΑΣ ΔΟΜΗΣ

Για την αποθήκευση και ανάκτηση δεδομένων χρησιμοποιήθηκε **δομή κατακερματισμού** (hashmap) ενώ για την υλοποίηση του πρωτοκόλλου LRU χρησιμοποιήθηκε **διπλά-συνδεδεμένη λίστα** (doubly-linked list).

Υλοποίηση cache: Στο hashmap αποθηκεύονται κόμβοι οι οποίοι περιέχουν τα δεδομένα ενώ ταυτόχρονα οι κόμβοι αυτοί τοποθετούνται σε μία διπλά συνδεδεμένη λίστα.

Κάθε φορά που καλείται η μέθοδος lookUp(), τοποθετείται στην **κεφαλή** της διπλά συνδεδεμένης λίστας το στοιχείο με το κλειδί που αναζητήθηκε (εφόσον υπάρχει).

Κατά την κλήση της store() εάν η cache είναι γεμάτη τότε με βάση το **πρωτόκολλο LRU** θα πρέπει να αφαιρεθεί το παλαιότερο στοιχείο, αυτό επιτυγχάνεται αφαιρώντας το στοιχείο που βρίσκεται στην **ουρά** της διπλά-συνδεδεμένης λίστας από αυτή και απ'το hashmap.

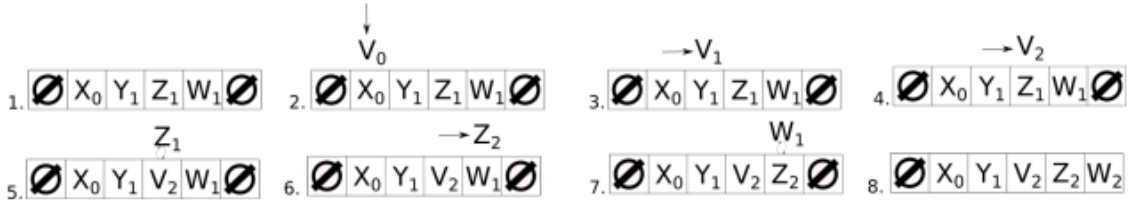
Υλοποίηση hashmap:

Ορίζουμε hash-position = hash(key)

Επιλέξαμε να υλοποιήσουμε το hashmap με αλγόριθμο κατακερματισμού **robin hood**. Ο αλγόριθμος αυτός χρησιμοποιεί το μοντέλο **open-addressing** και εξασφαλίζει ότι κάθε στοιχείο βρίσκεται κατά το δυνατόν πιο κοντά στο hash-position του. Για να επιτευχθεί αυτό, κάθε κόμβος κρατάει ένα επιπλέον πεδίο το οποίο καλείται **PSL** (Probe Sequence Length) και είναι η απόσταση του από το hash-position του.

put() – Κατά την εισαγωγή ενός στοιχείου, εάν είναι κενό το hash-position τότε ο προς εισαγωγή κόμβος εισάγεται κατευθείαν εκεί(PSL=0), διαφορετικά συνεχίζει να αναζητά θέση μέχρις ότου το PSL του ξεπεράσει το PSL του κόμβου κάποιας θέσης, στην περίπτωση αυτή, εναλάσσουμε τους δύο κόμβους (έτσι επιτυγχάνεται και η ελαχιστοποίηση των PSL) και επανεισάγουμε τον κόμβο του οποίου έλαβε τη θέση(σημειώνεται ότι τότε αυξάνεται το PSL του κόμβου).

Παράδειγμα εισαγωγής:



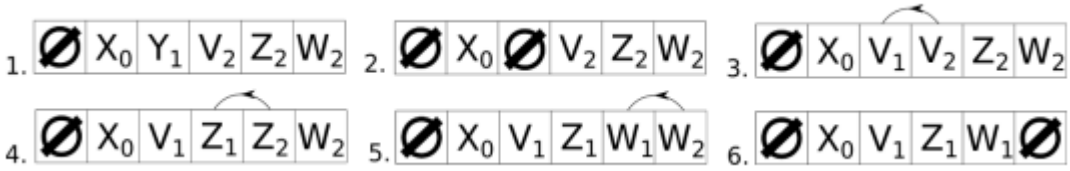
remove() – Στην περίπτωση που θέλουμε να γίνει αφαίρεση στοιχείου, τότε δεν αφαιρούνται πάντα απαραίτητα τα clusters. Για την αφαίρεση κάνουμε **back-shifting** με δύο διαφορετικά conditions τερματισμού:

(1) Η αμέσως επόμενη θέση του στοιχείου που αφαιρούμε είναι κενή (null). Διότι τότε σημαίνει ότι δεν υπάρχει στοιχείο που να έχει hash-position κάποια απ'τις προηγούμενες θέσεις. (αφού ελαχιστοποιούμαι την απόσταση των στοιχείων απ'το hash position τους)

(2) Η αμέσως επόμενη θέση περιέχει στοιχείο με PSL=0. Αφού τότε το στοιχείο αυτό θα μπορεί να βρεθεί σίγουρα καθώς βρίσκεται στο hash-position του.

Σημείωση: κατά το back-shifting μειώνουμε το PSL των εμπλεκόμενων στοιχείων.

Παράδειγμα αφαίρεσης:



findPos() – Για την υλοποίηση της findPos() χρησιμοποιούμε μία τεχνική η οποία καλείται **Smart Search**. Επειδή καθώς γεμίζει το hashmap αυξάνεται η πιθανότητα για collisions και άρα μειώνεται η πιθανότητα να βρούμε ένα στοιχείο κατευθείαν στο hash-position του, υπολογίζουμε το μέσο PSL των στοιχείων και ξεκινάμε από εκεί την αναζήτηση του συγκεκριμένου στοιχείου αναζητώντας στην εξής ακολουθία:

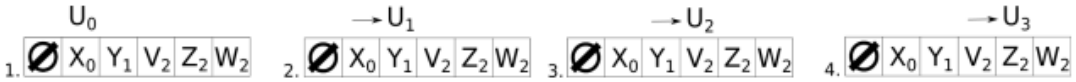
hash_position + mean, hash_position + mean -1, hash_position + mean+1

Η αναζήτηση σταματάει, όταν βρεθεί το στοιχείο είτε όταν έχουμε ψάξει απ'το hash-position (PSL=0) μέχρι και το μέγιστο PSL των στοιχείων + το hash position (διότι τότε σίγουρα δεν υπάρχει ακόμα πιο μακριά απ'το hash-position)

Επομένως για την υλοποίηση της findPos() διατηρούμε (και ενημερώνουμε όπου είναι αναγκαίο στις υπόλοιπες μεθόδους) ένα πεδίο που περιέχει το

άθροισμα των PSL's καθώς επίσης διατηρούμε ένα πεδίο για το μέγιστο PSL (το οποίο ενημερώνεται μόνο στην put – στη remove δεν ενημερώνεται)

Παράδειγμα αναζήτησης:



get() – Ανατρέξτε στη findPos().

hash() – Καλούμε τη μέθοδο hashCode() της java και εφαρμόζουμε τον τελεστή modulo στην τιμή που επιστρέφεται ώστε να λάβουμε index εντός του πίνακα μας.

Υλοποίηση διπλά-συνδεδεμένης λίστας:

Η υλοποίηση αυτή δεν παρουσιάζει κάποιο ενδιαφέρον.

Αξίζει όμως να σημειωθεί ότι χρησιμοποιήθηκε ψευδοκόμβος κεφαλής του οποίου ο δείκτης prev δείχνει στον εαυτό του και ψευδοκόμβος ουράς του οποίου ο δείκτης next δείχνει στον εαυτό του. Με τον τρόπο αυτό δεν χρειάζεται να πραγματοποιηθεί κανένας έλεγχος κατά την εισαγωγή ή εξαγωγή στοιχείων.

b.

ΥΠΟΛΟΓΙΣΤΙΚΟ ΚΟΣΤΟΣ

HashMap

put() – Στη **χειρότερη περίπτωση** το κόστος εισαγωγής ενός στοιχείου είναι O(ln(n)) (με βάση το paper[1] σελίδα 12). Ενώ στη **μέση περίπτωση** το κόστος είναι O(1) αφού κατά μέσο όρο η απόσταση των στοιχείων από το hash-position θα είναι πολύ μικρή.

remove() – Δεν έχει βρεθεί ακριβώς. Είναι σίγουρα όμως κάτω από O(N) και έχει για μέση περίπτωση O(1).

findPos() – Πάντα O(1) με βάση τη σελίδα 36 του paper[1].

Διπλά-Συνδεδεμένη Λίστα

Όλες οι μέθοδοι είναι O(1).

Cache

lookup() – O(1) καθώς η findPos() του hashmap είναι O(1).

updateCache() – O(1) καθώς είναι μέθοδος της διπλά-συνδεδεμένης λίστας.

store() – κόστος μέσης περίπτωσης: O(1), κόστος χειρότερης περίπτωσης: O(ln(n)). Καθώς χρησιμοποεί την put() της hashmap η οποία έχει κόστος μέσης περίπτωσης O(1) και worst-case O(ln(n)), και μεθόδους οι οποίες έχουν κόστος O(1).

replaceKey() – O(1)

removeOldest() – worst-case είναι ίδια με της remove. Μέση περίπτωση O(1).

updateCache() – O(1)

Οι getters έχουν όλες προφανώς κόστος O(1) καθώς χρησιμοποιούμε **eager approach** για το μέτρημα του πλήθους των lookups και των hits.

ΕΝΑΛΛΑΚΤΙΚΕΣ

Επιλέξαμε τον συγκεκριμένο αλγόριθμο (Robin Hood) για την υλοποίηση του hashmap καθώς πειραματεικά φαίνεται να αποδείχθηκε βέλτιστος ώς προς την ταχύτητα αναζήτησης [4].

Κάποια πράγματα τα οποία δοκιμάσαμε είναι τα εξής:

- tombstones αντί για backwards-shifting στη remove() (paper[1] σελ 56)

- Αντί για smart search απλή αναζήτηση κατά την οποία ξεκινώντας απ'το hash-position κρατάγαμε μετρητή PSL ώστε στην περίπτωση που ο μετρητής ξεπεράσει το PSL του στοιχείου σε κάποια θέση καθώς αναζητούμε να σταματήσουμε. (διότι τότε είναι σίγουρο πως το στοιχείο δεν υπάρχει αφού θα το είχαμε ήδη βρεί καθώς όλα στοιχεία βρίσκονται όσο πιο κοντά στο hash-position τους.)

- Ορισμός ενός πεδίου resize threshold στο οποίο εάν κατά την εισαγωγή ενός στοιχείου το PSL του στοιχείου έφτανε (ή ξεπέρναγε) το resize threshold τότε γίνεται resize ο πίνακας (το νέο μέγεθος είναι διπλάσιο + το resize threshold) και επανεισάγεται μετά το στοιχείο έτσι ώστε να κρατιούνται μικρά τα PSL's. Επίσης κατά τη resize και την αρχική κλήση του κατασκευαστή του hashmap, προστίθεται στο μέγεθος το resize threshold (ενώ συνεχίζουμε στις άλλες μεθόδους να θεωρούμε ότι το μέγεθος του πίνακα είναι απλώς το διπλάσιο χωρίς να του έχει προστεθεί το resize threshold) έτσι ώστε να μη χρειάζεται να χρησιμοποιηθεί ο τελεστής modulo (που έχει μεγάλο κόστος) στις άλλες μεθόδους καθώς πλέον δε γίνεται να βγούμε εκτός bounds καθώς τότε ο

πίνακας γίνεται resize() (διότι ξεπερνιέται το threshold). Για το resize threshold επιλέγαμε την τιμή log2(n). Για περισσότερες πληροφορίες ανατρέξτε στο [3].

- Υλοποίηση του hash map με χωριστή αλυσίδωση. (διαφάνειες κεφ.15)

- Υλοποίηση του hash map με διπλό κατακερματισμό. (διαφάνειες κεφ.15)

  Τα παραπάνω φαίνεται να αποδίδαν χειρότερα.

c.

ΠΗΓΕΣ

[1] https://cs.uwaterloo.ca/research/tr/1986/CS-86-14.pdf

[2] https://cs.emis.de/LIPIcs/volltexte/2018/10070/pdf/LIPIcs-OPODIS-2018-10_.pdf

[3] https://probablydance.com/2017/02/26/i-wrote-the-fastest-hashtable/

[4] https://martin.ankerl.com/2019/04/01/hashmap-benchmarks-01-overview/