Δομές Δεδομένων Εργασία 4

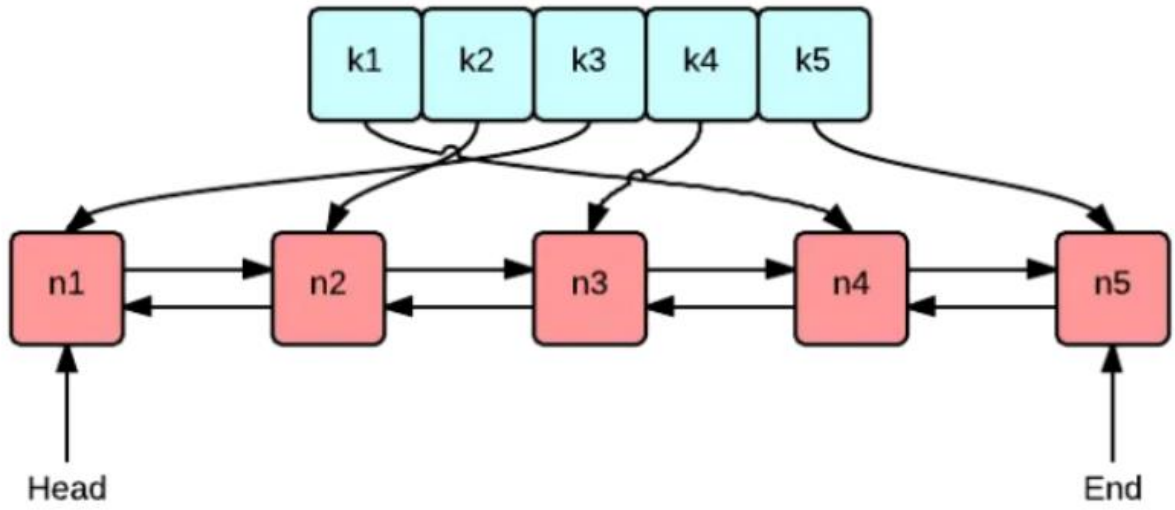
Υλοποίηση LRU Cache – Αναφορά Παράδοσης

Μητσάκης Νίκος : p3210122

Παντελίδης Ιπποκράτης : p3210150

1)Χρησιμοποιώντας όλες τις πληροφορίες της εκφώνησης αναφορικά με την μνήμη cache αποφασίσαμε να χρησιμοποιήσουμε ένα συνδυασμό δομών δεδομένων για την υλοποίηση της. Ειδικότερα επιλέξαμε μία διπλά συνδεδεμένη λίστα(doubly linked list) και ένα πίνακα κατακερματισμού(hash table) ή αλλιώς (hash map) και θα εξηγήσουμε παρακάτω τον τρόπο με τον οποίο αξιοποιήσαμε τις συγκεκριμένες επιλογές. Σύμφωνα με την εκφώνηση η Cache που κληθήκαμε να υλοποιήσουμε ακολουθεί την πολιτική Least-Recently-Used(LRU) και αυτή την ιδέα εξυπηρετεί πολύ αποδοτικά η διπλά συνδεδεμένη λίστα. Ειδικότερα, μας επιτρέπει να τοποθετούμε και το κυριότερο να αφαιρούμε πολύ γρήγορα και χωρίς να χρειαστεί ολόκληρη η διάσχιση των κόμβων της(απλά χρησιμοποιώντας τον δείκτη prev) γεγονός που πολλάκις χρησιμοποιήσαμε στην υλοποίηση μας. Επομένως το καθήκον της δίπλα συνδεδεμένης λίστας είναι να αποθηκεύει τα αντικείμενα πληροφορίας(HashMapEntry) που βρίσκονται την δεδομένη χρονική στιγμή στην cache με τέτοιο τρόπο ώστε αυτό που βρίσκεται στην αρχή(head) να είναι το παλαιότερο(δηλαδή αυτό που αναζητήθηκε πρώτο == χαμηλή προτεραιότητα) και καθώς προχωράμε προς το tail, τα στοιχεία βρίσκονται αποθηκευμένα στην δομή με την σειρά που αναζητήθηκαν. Είναι επίσης προφανές ότι στο tail βρίσκεται το πιο πρόσφατο στοιχείο που αναζητήθηκε. Αυτή είναι ιδέα πίσω από την χρήση της διπλά συνδεδεμένης λίστας και αργότερα θα δούμε πως μεταβάλλεται με την κλήση των μεθόδων της Cache. Επίσης χρησιμοποιούμε και μια συνάρτηση κατακερματισμού με χωριστή αλυσίδωση και έτσι κάθε κελί του πίνακα του hash map έχει και από μία μονά συνδεδεμένη λίστα(Bucket or singly linked list) με τα στοιχεία που έχουν το κλειδί του index. Με αυτόν τον τρόπο μπορούμε πολύ γρήγορα όπως θα δούμε στην ανάλυση πολυπλοκοτήτων ότι μπορούμε πολύ εύκολα και γρήγορα να προσθέσουμε και αναζητήσουμε έναν αντικείμενο πληροφορίας από την δομή μας. Δυο λειτουργίες που υποστηρίζει η Cache μας είναι η lookUp(K key) και η store(K key, V value), μέθοδοι που επιδρούν στις δομές μας. Ειδικότερα η lookUp ψάχνει εάν ένα δοσμένο κλειδί υπάρχει στο hash map και αν υπάρχει τότε έχουμε cache hit. Σε αυτή την περίπτωση δεν

πειράζουμε καθόλου το hash map μας καθώς υπάρχει μέσα το κλειδί, όμως πρέπει να αποκαταστήσουμε την προτεραιότητα στην λίστα και έτσι αφαιρούμε τον κόμβο από το σημείο που βρισκόταν και τον μεταφέρουμε στο τέλος(tail) ώστε να γίνει αυτός που αναζητήθηκε πιο πρόσφατα. Αν όμως δεν συμβεί αυτό τότε έχουμε cache miss, δηλαδή το στοιχείο δεν βρίσκεται στο hash map μας, και επομένως πρέπει να το προσθέσουμε, δουλειά που κάνει η store. Πρώτα από όλα ελέγχει αν το hash map έχει γεμίσει και αν ναι αφαίρει από την λίστα τον κόμβο με την χαμηλότερη προτεραιότητα(head) και παίρνοντας τα στοιχεία του, τον αφαιρούμε και από το hash map,απελευθερώνοντας έτσι χώρο για την νέα εγγραφή. Στην συνέχεια προσθέτουμε την νέα εγγραφή στο τέλος της λίστας(πιο πρόσφατη) καθώς και στο hash map, ενέργεια που κάνουμε και όταν το hash map μας δεν έχει γεμίσει. Ακολουθεί μια εικόνα για καλύτερη κατανόηση του τρόπου σκέψης μας.



2)Υπολογιστικό Κόστος:

-Doubly Linked List

Methods	Time Complexity
• removeNode()	O(1)
• insertAtBack()	O(1)
• getNode()	O(N)

Είναι πολύ σημαντικό το ότι η διαγραφή του οποιοδήποτε κόμβου από την λίστα(removeNode) γίνεται σε O(1) χάρις την χρήση του δείκτη prev, ενώ αν χρησιμοποιούσαμε λίστα μονής σύνδεσης θα κάναμε την διαγραφή σε O(N),καθώς θα έπρεπε να ξέρουμε τον προηγούμενο από αυτόν που θέλουμε να διαγράψουμε και στην χειρότερη θα την διασχίζαμε ολόκληρη. Η εισαγωγή πολύ απλά γίνεται σε O(1), ενώ για την εύρεση ενός κόμβου στην λίστα χρειαζόμαστε διάσχιση της λίστας και για αυτό στην χειρότερη έχουμε O(N).

-Hash Map

Methods	Time Complexity
• get()	O(N)
• remove()	O(N)
• add()	O(1)

Ουσιαστικά εδώ οι πολυπλοκότητες προκύπτουν από την μονά συνδεδεμένη λίστα και επειδή η get και η remove χρειάζονται διάσχιση της λίστας είναι O(N).Η add απλά εισάγει σε O(1). Είναι σημαντικό να αναφερθεί ότι τα Buckets βρίσκονται σε O(1) αφού είναι arrays και ξέρουμε το index κάθε φορά. Επομένως η lookUp και η store είναι O(N).

3)Για να κατανοήσουμε καλύτερα την λογική πίσω από την LRU Cache καθώς και για να βρούμε τις βέλτιστες δομές δεδομένων για την υλοποίηση της ερευνήσαμε στις ακόλουθες ιστοσελίδες:

- https://ogroetz.medium.com/lru-cache-a-cache-data-structure-1fab0d948e94
- https://stackoverflow.com/questions/60259908/why-lru-caches-use-doubly-link-list-and-not-singly-link-list
- https://www.geeksforgeeks.org/lru-cache-implementation/
- https://krishankantsinghal.medium.com/my-first-blog-on-medium-583159139237