Αναφορά Υλοποίησης LRUCache

Επισκόπηση κλάσεων:

Node: Αντιπροσωπεύει τους κόμβος που χρησιμοποιούνται από την κλάση CustomLinkedList

Fields:

-key: το μοναδικό αναγνωριστικό

-value: τα δεδομένα

-next: δείκτης στον επόμενο κόμβο

-prev: δείκτης στον προηγούμενο κόμβο

Methods:

Getters:

-getValue

-getKey

-getNext

-getPrev

Setters:

-setValue

-setKey

-setNext

-setPrev

toString: για την εμφάνιση ενός κόμβου

CustomLinkedList: Διπλά Συνδεδεμένη Λίστα με μεθόδους που κάνουν ευκολότερη την υλοποίηση της LRUCache

Fields:

-**head**: δείκτης στην αρχή της Λίστας

-**tail**: δείκτης στο τέλος της Λίστας

Methods:

-**CustomLinkedList**: (constructor) που αρχικοποιεί τις τιμές των δυο δεκτών σε null

-**getHead**: επιστρέφει τον κόμβο κεφαλής της λίστας

- **getTail**: επιστρέφει τον κόμβο ουράς της λίστας

-**clear**: θέτει τους κόμβους ουράς και κεφαλής σε null αποσυνδέοντας τις αναφορές στους κόμβους της λίστας και αφήνοντας τον σκουπιδοσυλλέκτη να επανακτήσει τη μνήμη που χρησιμοποιείται από τους κόμβους.

-**isEmpty**: επιστρέφει true αν η λίστα είναι κενή

-**insertAtTail**: δέχεται ένας κόμβο και τον τοποθετεί στην ουρά της λίστας εφόσον δεν είναι άδεια αλλιώς θέτει την κεφαλή και την ουρά ως τον νέο κόμβο

-**insertAtHead**: παρόμοια λειτουργιά με την insertAtTail αλλά τοποθετεί τον κόμβο στην κεφαλή στης λίστας

-**deleteHead**: αν η λίστα είναι κενή τότε δεν κάνει τίποτα. Αν έχει μόνο ένα αντικείμενο απλά το αφαιρει αλλιώς μετακινεί την κεφαλή της λίστας μια θέση μπροστά, αφαιρει την σύνδεση με τον προηγούμενο κόμβο και τον επιστρέφει

-**deleteTail**: παρόμοια λειτουργιά με την deleteHead αλλά για την ουρά της λίστας

-**detachFromList**: Δέχεται ένα κόμβο ώστε να τον αποσυνδέσει από την λίστα. Ελέγχει πρώτα αν ο κόμβος είναι null. Εάν είναι, δεν κάνει τίποτα. Διαφορετικά, η μέθοδος αφαιρεί τον κόμβο από τη λίστα αποσυνδέοντάς τον από τους γείτονές του. Εάν ο κόμβος είναι η κεφαλή ή η ουρά της λίστας, ενημερώνει τον δείκτη κεφαλής ή ουράς αντίστοιχα. Εάν στη λίστα έχει απομείνει μόνο ένα στοιχείο, ορίζει τους δείκτες κεφαλής και ουράς σε null. Τέλος, σπάει τους συνδέσμους του κόμβου για να τον αποσπάσει πλήρως από τη λίστα.

-**moveToTail**: Δέχεται ενα κόμβο και με την βοήθεια της detachFromList και της insertAtTail τον μετακινεί στην ουρά της λίστας

-**moveToHead**: Παρόμοια λειτουργία με την moveToTail αλλά μετακινεί τον κόμβο στην κεφαλή της λίστας

LRUCache: Κλάση που υλοποιεί την Διεπαφή Cache

Fields:

-capacity: η μέγιστη χωρητικότητα της Cache.

-size: ο τρέχων αριθμών των στοιχείων της Cache.

-map: δομή δεδομένων που θα χρειαστεί για την υλοποίηση.

-list: επίσης δομή δεδομένων που θα χρειαστεί για την υλοποίηση.

Methods:

-LRUCache: constructor της κλάσης που δέχεται την μεγίστη χωρητικότητα ως παράμετρο, θέτει το size (field) ίσο με το μηδέν (γιατί δεν υπάρχουν ακόμα τιμές μέσα στην Cache), δημιουργεί έναν κενό πίνακα κατακερματισμού και μια διπλά συνδεδεμένη λίστα.

-**evictLRU**: αφαιρεί τον Least Recently Used κόμβο από την λίστα και από τον πίνακα κατακερματισμού και μειώνει το size κατά ένα.

-updateExistingNode: Δέχεται έναν κόμβο και μια τιμή (value), αλλάζει την τιμή του κόμβου και τον μετακινεί στην ουρά της λίστας.

-addNewNode: Δέχεται ένα κλειδί(key) και μια τιμή(value), δημιουργεί έναν κόμβο με αυτές τις παραμέτρους, τον τοποθετεί στον πίνακα κατακερματισμού μαζί με το κλειδί και τέλος αυξάνει το size κατά ένα.

-getSize: getter για το field size.

-get: Δέχεται την τιμή που σχετίζεται με ένα δεδομένο κλειδί σε ένα λεξικό. Εάν το κλειδί βρεθεί, μετακινεί τον αντίστοιχο κόμβο στην ουρά της λίστας και επιστρέφει την τιμή(value) που σχετίζεται με τον κόμβο. Εάν το κλειδί δεν βρεθεί, επιστρέφει null.

-put: Δέχεται ένα ζεύγος κλειδιού-τιμής. Εάν το κλειδί υπάρχει ήδη στο λεξικό, ενημερώνει την τιμή που σχετίζεται με το κλειδί και μετακινεί τον κόμβο στην ουρά της λίστας (με την βοήθεια της updateExistingNode). Εάν το κλειδί δεν υπάρχει στο λεξικό, προσθέτει το ζεύγος κλειδιού-τιμής στο λεξικό και στη λίστα. Εάν τα δεδομένα φτάσουν την μέγιστη χωρητικότητα της cache, η μέθοδος αφαιρεί τον κόμβο που χρησιμοποιήθηκε λιγότερο πρόσφατα (LRU) από τη λίστα και τον χάρτη.

Δοκιμές

LRUCacheTest: Η κλάση με της δοκιμές

Methods:

-testPutAndGetMethods:

Σκοπός:

Επαληθεύει τη βασική λειτουργικότητα των μεθόδων put() και get().

Βήματα:

-Δημιουργεί μια LRUCache με MIN_CACHE_CAPACITY(3)

- Εισάγει τρείς κόμβους.

- Ανακτά τις τιμές για κάθε κλειδί και βεβαιώνει ότι είναι σωστές.

-testLRURemoval:

Σκοπός:

Δοκιμάζει τη συμπεριφορά εξώθησης LRU (Last Recently Used).

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY

-Εισάγει τέσσερις κόμβους.

-Ελέγχει αν το πρώτο στοιχειό είναι null (διότι λόγο της χωρητικότητας της cache θα έχει αφαιρεθεί το LRU)

- Ελέγχει αν τα υπόλοιπα στοιχειά είναι μέσα στην cache

-testKeyUpdate:

Σκοπός:

Δοκιμές ενημέρωσης της τιμής για ένα υπάρχον κλειδί.

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY

-Εισάγει ένα ζεύγος κλειδιού-τιμής.

-Εισάγει το ίδιο κλειδί με νέα τιμή.

- Ανακτά την τιμή για το κλειδί και βεβαιώνει ότι έχει ενημερωθεί.

- Επαληθεύει ότι το μέγεθος της cache δεν έχει αυξηθεί.

-testLRUOrder:

Σκοπός:

Επαληθεύει την σειρά που είναι τα δεδομένα στην cache

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY

-Προσθέτει τρία

-Εισάγει τρία ζεύγοι κλειδιού-τιμής

-Καλεί το πρώτο ζεύγος ώστε να αλλάξει θέση και προσθέτει ένα ακόμα ζεύγος ώστε να διώξει το δεύτερο ζεύγος που προστέθηκε στην cache

-Ελέγχει ότι τα ζεύγη που πρέπει να βρίσκονται μέσα είναι μέσα και αυτό που δεν πρέπει να είναι να μην είναι

-testEdgeCase:

Σκοπός:

Ελέγχει τη συμπεριφορά κατά την προσπάθεια ανάκτησης μιας τιμής για ένα κλειδί που δεν υπάρχει.

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY

-Ελέγχει αν προσπαθήσει ο χρήστης να ανακτήσει μια τιμή που δεν υπάρχει τότε να επιστρεψεί την τιμή null

-testStressTest:

Σκοπός:

Δοκιμάζει την cache υπό μεγάλο φορτίο με τυχαίες εισαγωγές και ανακτήσεις.

Βήματα:

-Δημιουργεί μια cache με MAX_CACHE_CAPACITY

- Εισάγει τον διπλάσιο από τον μέγιστο αριθμό ζευγών κλειδιού-τιμής με τυχαία κλειδιά.

- Ανακτά τις τιμές για όλα τα κλειδιά που έχουν εισαχθεί και βεβαιώνει ότι είναι σωστές ή μηδενικές (εάν έχουν εξαχθεί).

Παράδειγμα Χρήσης LRUCache

Δημιουργία με χωρητικότητα 3:

```
Cache<Integer, String> cache = new LRUCache<>(3);
```

Προσθήκη τριών ζευγών-κλειδιού τιμής:

```
cache.put(1, “one”);
cache.put(2, “two”);
cache.put(3, “three”);
```

Εμφάνιση ενός ζεύγους στην οθόνη:

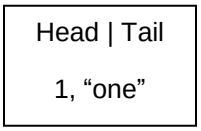
```
System.out.println(cache.get(1)); //θα εμφανίσει “one”
```

Προσθήκη ενός παραπάνω στοιχείου:

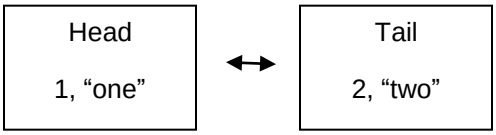
```
cache.put(4, “four”);
//λόγο χωρητικότητας η cache θα χρειαστεί να απελάσει το “Least Recently Used”
```

Οπτικοποίηση παραδείγματος

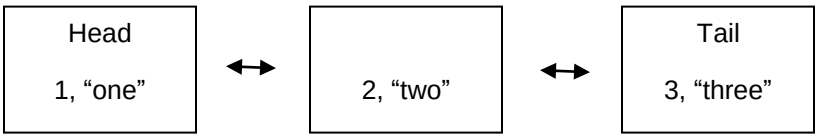
cache.put(1, “one”);



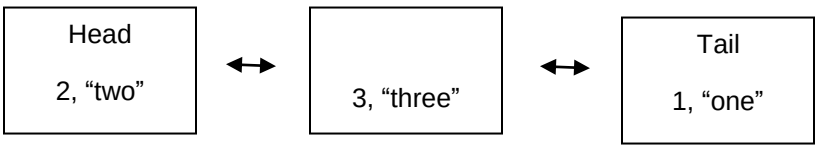
cache.put(2, “two”);



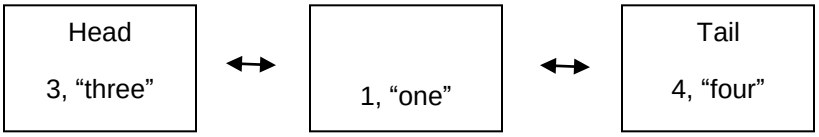
cache.put(3, “three”);



System.out.println(cache.get(1));



cache.put(4, “four”);



Σκοπός LRUCache

Επιτυγχάνει βέλτιστη απόδοση και αξιοπιστία, με (αναμενόμενη) πολυπλοκότητα χρόνου $O(1)$ για τοποθέτηση (put) και λήψη (get).