Αναφορά Υλοποίησης Cache

---

it2023003 & it2023052 & it2023100

1ο Μέρος Εργασίας

Επισκόπηση κλάσεων:

Node: Αντιπροσωπεύει τους κόμβος που χρησιμοποιούνται από την κλάση CustomLinkedList

Πεδία:

-key: το μοναδικό αναγνωριστικό

-value: τα δεδομένα

-next: δείκτης στον επόμενο κόμβο

-prev: δείκτης στον προηγούμενο κόμβο

Μέθοδοι:

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

Μέθοδοι:

-**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 αριθμός των στοιχείων της Cache.

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

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

Μέθοδοι:

-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 το κλειδί που σχετίζεται με μία δεδομένη τιμή 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

Μέθοδοι:

-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

-Εισάγει τρία ζεύγη 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”

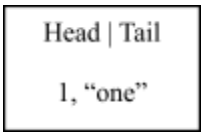
Προσθήκη ενός παραπάνω στοιχείου:

cache.put(4, “four”);

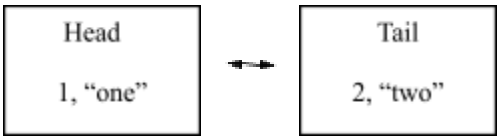
//λόγο χωρητικότητας η cache θα χρειαστεί να απελάσει το “Least Recently Used”

Οπτικοποίηση παραδείγματος

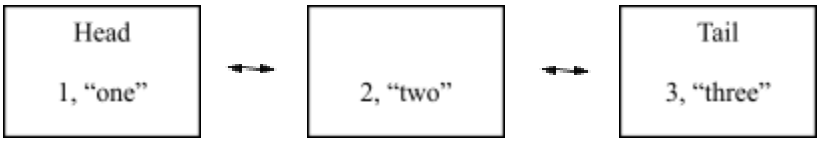
cache.put(1, “one”);



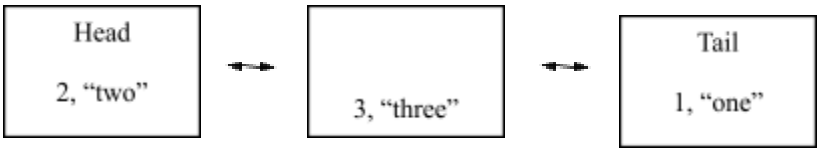
cache.put(2, “two”);



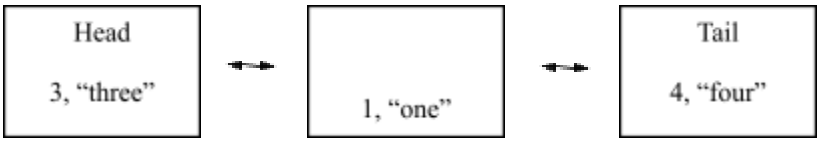
cache.put(3, “three”);



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



cache.put(4, “four”);



2ο Μέρος Εργασίας

Πρόσθετοι τύποι:

CacheReplacementPolicy: Ένα enum που περιγράφει πολιτικές αντικατάστασης δεδομένων στη μνήμη cache. Παρέχει δύο βασικές πολιτικές:

1. **LRU (Least Recently Used)**: Η πολιτική αντικατάστασης που αφαιρεί το στοιχείο που χρησιμοποιήθηκε λιγότερο πρόσφατα.
2. **MRU (Most Recently Used)**: Η πολιτική αντικατάστασης που αφαιρεί το στοιχείο που χρησιμοποιήθηκε πιο πρόσφατα.

Πεδία:

- **description:** Μια συμβολοσειρά που περιγράφει την πολιτική αντικατάστασης.

Μέθοδοι:

- **getDescription():** Επιστρέφει την περιγραφή της πολιτικής.
- **toString():** Υπερκαλύπτει τη μέθοδο toString ώστε να επιστρέφει την περιγραφή.

Αλλαγές σε υπάρχοντες τύπους:

CacheImpl:

Μετονομασία της κλάσης LRUCache σε CacheImpl αφού πλέον υποστηρίζονται περισσότερες από μια πολιτικές αντικατάστασης.

Πρόσθετα Πεδία:

- **hitCount** - Μετρά τον αριθμό των επιτυχημένων αιτήσεων (cache hits).
- **missCount** - Μετρά τον αριθμό των αποτυχημένων αιτήσεων (cache misses).
- **policy** - Καθορίζει την πολιτική αντικατάστασης που θα χρησιμοποιηθεί.

Πρόσθετες Μέθοδοι:

- **getHitCount** - Επιστρέφει τον αριθμό των επιτυχιών στη μνήμη cache.
- **getMissCount** - Επιστρέφει τον αριθμό των αστοχιών στη μνήμη cache.
- **evictMRU** - αφαιρεί τον Most Recently Used κόμβο από την λίστα και από τον πίνακα κατακερματισμού και μειώνει το size κατά ένα.

Αλλαγές στη μέθοδο put: Πλέον η μέθοδος καλεί την μέθοδο evictLRU ή την μέθοδο evictMRU ακολουθώντας την πολιτική αντικατάστασης που έχει καθοριστεί όταν η cache φτάσει τη μέγιστη χωρητικότητα της.

App:

Πεδία:

- **totalOperations** - Ο συνολικός αριθμός των λειτουργιών που εκτελούνται (100.000).
- **cacheCapacity** - Η μέγιστη χωρητικότητα της cache (2.000).
- **eightyPercentKeys** - Πίνακας που περιέχει 1.000 κλειδιά, τα οποία αντιπροσωπεύουν το 80% των αιτημάτων που θα ανακτηθούν από την cache.

- **twentyPercentKeys** - Πίνακας που περιέχει 1.000 κλειδιά, τα οποία αντιπροσωπεύουν το 20% των αιτημάτων που θα ανακτηθούν από την cache.
- **cache** - Αντικείμενο τύπου cache που χρησιμοποιείται για την αποθήκευση δεδομένων και την εκτέλεση αιτήσεων ανάκτησης. Εφαρμόζει την πολιτική αντικατάστασης **LRU**.

Μέθοδοι:

main:

- Περιγραφή: Η κύρια μέθοδος εκτέλεσης του προγράμματος που δημιουργεί και εκτελεί το σενάριο δοκιμής για την cache.
- **Λειτουργίες**:
  - Αρχικοποιεί την cache με χωρητικότητα 2.000 και πολιτική αντικατάστασης **LRU**.
  - Εισάγει 2.000 κλειδιά στη cache (1.000 κλειδιά με πιθανότητα 80% και 1.000 κλειδιά με πιθανότητα 20%).
  - Εκτελεί 100.000 αιτήσεις προς τη cache (εκ των οποίων το 80% είναι αιτήσεις για τα κλειδιά 0-999 και το 20% για τα κλειδιά 1000-1999).
  - Εκτελεί αιτήσεις για κλειδιά που δεν υπάρχουν στη cache (αστοχίες).
  - Εκτυπώνει τα αποτελέσματα: τον αριθμό των συνολικών αιτήσεων, τον αριθμό των επιτυχιών και αποτυχιών, τα ποσοστά επιτυχιών και αποτυχιών.

Πρόσθετες δοκιμές:

-testMRURemoval:

Σκοπός:

Δοκιμάζει τη συμπεριφορά εξώθησης MRU (Most Recently Used).

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική MRU

-Εισάγει τέσσερις κόμβους.

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

-Ελέγχει αν τα υπόλοιπα στοιχεία είναι μέσα στην cache

-testMRUKeyUpdate:

Σκοπός:

Δοκιμές ενημέρωσης της τιμής για ένα υπάρχον κλειδί

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική MRU

-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.

-testMRUOrder:

Σκοπός:

Επαληθεύει την σειρά που είναι τα δεδομένα στην cache

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική MRU

-Εισάγει τρία ζεύγη κλειδιού-τιμής

-Καλεί το πρώτο ζεύγος ώστε να αλλάξει θέση και προσθέτει ένα ακόμα ζεύγος ώστε να διώξει το πρώτο 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

-testMRUEdgeCase:

Σκοπός:

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

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική MRU

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

-testMRUStressTest:

Σκοπός:

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

Βήματα:

-Δημιουργεί μια cache με MAX_CACHE_CAPACITY και πολιτική MRU

-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).

3ο Μέρος Εργασίας

Πρόσθετοι τύποι:

CacheReplacementPolicy: Προσθήκη πολιτικής **LFU (Least Frequently Used)** κατά την οποία αφαιρείται το στοιχείο που έχει χρησιμοποιηθεί λιγότερες φορές.

Αλλαγές σε υπάρχοντες τύπους:

CacheImpl:

Δημιουργία ενός καινούργιου πεδίου τύπου CacheType (interface) το οποίο ενθυλακώνει τη λογική των πολιτικών της κρυφής μνήμης. Το interface CacheType δηλώνει τρεις μεθόδους (access, insert & evict).

Γίνεται δημιουργία εσωτερικών κλάσεων οι οποίες υλοποιούν τις μεθόδους του CacheType για τις τρεις πολιτικές (LRU, MRU & LFU).

Node: Προσθήκη ενός καινούργιου πεδίου **frequency** το οποίο χρησιμοποιείται για την υλοποίηση της πολιτικής **LFU**. Ενώ προστίθενται και οι μέθοδοι **getFrequency** η οποία επιστρέφει την τιμή του frequency και **incrementFrequency** η οποία αυξάνει το frequency κατά μία μονάδα και καλείται κάθε φορά που χρησιμοποιούμε το στοιχείο.

LFUCache:

Κλάση η οποία υλοποιεί την πολιτική **Least Frequently Used**. Η υλοποίηση γίνεται με την χρήση της δομής **TreeMap** η οποία αντιστοιχεί κάθε συχνότητα (key) σε μία λίστα **CustomLinkedList** (value) η οποία περιέχει όλα τα στοιχεία (τύπου Node) με τη συγκεκριμένη συχνότητα.

Η κλάση **TreeMap** στην Java υλοποιείται ως Δυαδικό Δέντρο Red-Black, το οποίο διατηρεί τα στοιχεία του ταξινομημένα σύμφωνα με τη φυσική τους σειρά άρα είναι άμεση η επεξεργασία των στοιχείων με τη μικρότερη συχνότητα. Σύμφωνα με αυτη τη πολιτική όταν η κρυφή μνήμη "γεμίσει" πρέπει να απομακρύνουμε το στοιχείο με την μικρότερη συχνότητα, στη περίπτωση που είναι μόνο ένα τότε απλά το διαγράφουμε, εάν υπάρχουν παραπάνω από ένα στοιχεία με την ίδια συχνότητα τότε αυθαίρετα διαγράφουμε το στοιχείο που βρίσκεται στη κεφαλή της λίστας (το πιο παλιό).

Επιπλέον χρησιμοποιείται και μη δομή **HashMap** η οποία μας επιτρέπει να βρίσκουμε ένα συγκεκριμένο node σε σταθερό χρόνο, έτσι η διαγραφή ενός στοιχείου από μία λίστα συχνότητας επίσης παίρνει σταθερό χρόνο.

Μέθοδοι:

Οι μέθοδοι της κλάσης LFUCache όπως και των υπόλοιπων κλάσεων LRUCache και MRUCache είναι οι τρεις μέθοδοι που δηλώνονται στο interface CacheType.

LRUCache & MRUCache:

-access: μεταφέρει ένα ήδη υπάρχων στοιχείο στο τέλος της λίστας

-insert: προσθέτει ένα καινούργιο στοιχείο στο τέλος της λίστας

-evict: διαγράφει το στοιχείο που βρίσκεται στη "κεφαλή" της λίστας οταν η πολιτική είναι **LRU** ενώ όταν η πολιτική είναι **MRU** διαγράφει το στοιχείο που βρίσκεται στην "ουρά" της λίστας.

LFUCache:

-access: μεταφέρει ένα στοιχείο από τη λίστα της συχνότητας που είχε στην λίστα της με την καινούργια συχνότητά του.

-insert: προσθέτει ένα καινούργιο στοιχείο στη λίστα της συχνότητας του στοιχείου

-evict: διαγράφει το στοιχείο με τη μικρότερη συχνότητα

Απόδοση αλγορίθμων:

Αλγόριθμος	**get** χρονική πολυπλοκότητα	**put** χρονική πολυπλοκότητα
LRU	$O(1)$ (αναμενόμενη)	$O(1)$ (αναμενόμενη)
MRU	$O(1)$ (αναμενόμενη)	$O(1)$ (αναμενόμενη)
LFU	$O(logn)$	$O(logn)$

Πρόσθετες δοκιμές:

-testLFURemoval:

Σκοπός:

Δοκιμάζει τη συμπεριφορά εξώθησης LFU (Least Frequently Used).

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική LFU

-Εισάγει τέσσερις κόμβους.

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

-Ελέγχει αν τα υπόλοιπα στοιχεία είναι μέσα στην cache

-testLFUKeyUpdate:

Σκοπός:

Δοκιμές ενημέρωσης της τιμής για ένα υπάρχον κλειδί.

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική LFU

-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.

-testLFUOrder:

Σκοπός:

Δοκιμάζει τη συμπεριφορά εξώθησης LFU (Least Frequently Used).

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική LFU

-Εισάγει τέσσερις κόμβους.

-Ελέγχει αν το τρίτο στοιχειό είναι null (διότι λόγο της χωρητικότητας της cache θα έχει αφαιρεθεί το LFU).

-Ελέγχει αν τα υπόλοιπα στοιχεία είναι μέσα στην cache.

-testLFUEdgeCase:

Σκοπός:

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

Βήματα:

-Δημιουργεί μια cache με MIN_CACHE_CAPACITY και πολιτική LFU

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

-testLFUStressTest:

Σκοπός:

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

Βήματα:

-Δημιουργεί μια cache με MAX_CACHE_CAPACITY και πολιτική LRU

-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).