DomesDedomenwn

Το πρόγραμμα DomesDedomenwn προσομοιώνει την λειτουργία μιας μνήμης cache σε Java.

Ο σκοπός αυτής της προσομοίωσης είναι η μνήμη cache να έχει ένα σταθερό (σχετικά μικρό) μέγεθος και να δέχεται αντικείμενα. Όταν γεμίσει η μνήμη πρέπει να βγάζει ένα στοιχείο έξω από την μνήμα για να κάνει χώρο για ένα επόμενο. Το κριτήριο που καθορίζει το ποιος θα βγει για να κάνει χώρο αποφασίζεται με τον τύπο της cache (LRU, MRU, LFU). Ο τύπος της cache ορίζει το τι replacement priority έχει η συγκεκριμένη μνήμη.

Δουλειά του προγράμματος είναι:

1) να υλοποιεί τους τρεις τύπους μνήμης cache,
2) να κάνει τα κατάλληλα test για κάθε μνήμη
3) και να εμφανίζει τα αποτελέσματα της λειτουργίας hit/miss.

1. Τύποι Μνήμης Cache

Αυτό το πρόγραμμα υποστηρίζει 3 τύπους μνήμης cache:

1) LRU Cache
2) MRU Cache
3) FRU Cache

Επειδή χρειάζονται διαφορετικές μεταβλητές για την υλοποίηση του LRU και του MRU από την υλοποίηση του LFU, έχουμε φτιάξει δύο εσωτερικές κλάσεις, την CachingLruMru και την CachingLfu, όπου και οι δύο είναι υποκλάσεις της εσωτερικής διεπαφής Caching. Η δουλειά της CachingLruMru είναι να υλοποιεί το get και το put για τους τύπους LRU και MRU και η δουλειά της CachingLfu είναι να υλοποιεί το get και το put για τον τύπο LFU. Όταν καλείτε η get ή η put σε μία μνήμη cache, αυτή καλεί αυτόματα την get ή την put της CachingLruMru ή της CachingLfu ανάλογα με τον τύπο της.

Τύπος 1 (LRU Cache):

Ο πρώτος τύπος μνήμης cache είναι η LRU (least recently used) Cache που το στοιχείο που θα βγει για να κάνει χώρο για τον επόμενο είναι αυτό που χρησιμοποιήθηκε λιγότερο πρόσφατα.

Υλοποίηση και Order λειτουργιών της LRU Cache:

Ο τρόπος υλοποίησης της LRU θα γίνει με την βοήθεια της έτοιμης υλοποίησης κατακερματισμένης αποθήκευσης της Java, το HashMap και μια δικιά μας υλοποίηση μιας διπλά συνδεδεμένης λίστας με βοήθεια της εσωτερικής κλάσης Node. Η σύνδεση μεταξύ των δύο είναι εφικτή επειδή η διπλά συνδεδεμένη λίστα κουβαλάει το key και το value που χρειαζόμαστε για το αντικείμενο και το HashMap κουβαλάει σαν key το ίδιο key με το Node και σαν value δέχεται το ίδιο το Node. Έτσι για την αναζήτηση ενός κόμβου με το key χρησιμοποιούμε το HashMap και μας επιστρέφει το value το που είναι το ίδιο το Node οπότε η εύρεση κάποιου κόμβου με ένα key γίνεται σε O(1) χρόνο. Συνεπώς, O(1) χρόνου είναι και η διαγραφή γιατί αφού με την αναζήτηση (που είναι O(1) χρόνου) παίρνουμε τον κόμβο που θέλουμε να διαγράψουμε, το μόνο που μένει να κάνουμε είναι να αλλάξουμε το head και να πειράξουμε τα κατάλληλα next και prev για να προσαρμόσουμε την λίστα μας που επίσης είναι O(1) χρόνου. Κάθε διαγραφή γίνεται από την αρχή της λίστας (δηλαδή το head) κάθε φορά που γεμίζει η μνήμη cache. Τέλος, και η πρόσθεση στοιχείου είναι O(1) χρόνου γιατί γίνεται από το τέλος και το μόνο που κάνουμε είναι να πειράξουμε το tail με τον κόμβο που θέλουμε να βάλουμε (και το next και prev).

Συμπερασματικά, κάθε λειτουργία της LRU Cache γίνετε σε O(1) χρόνο.

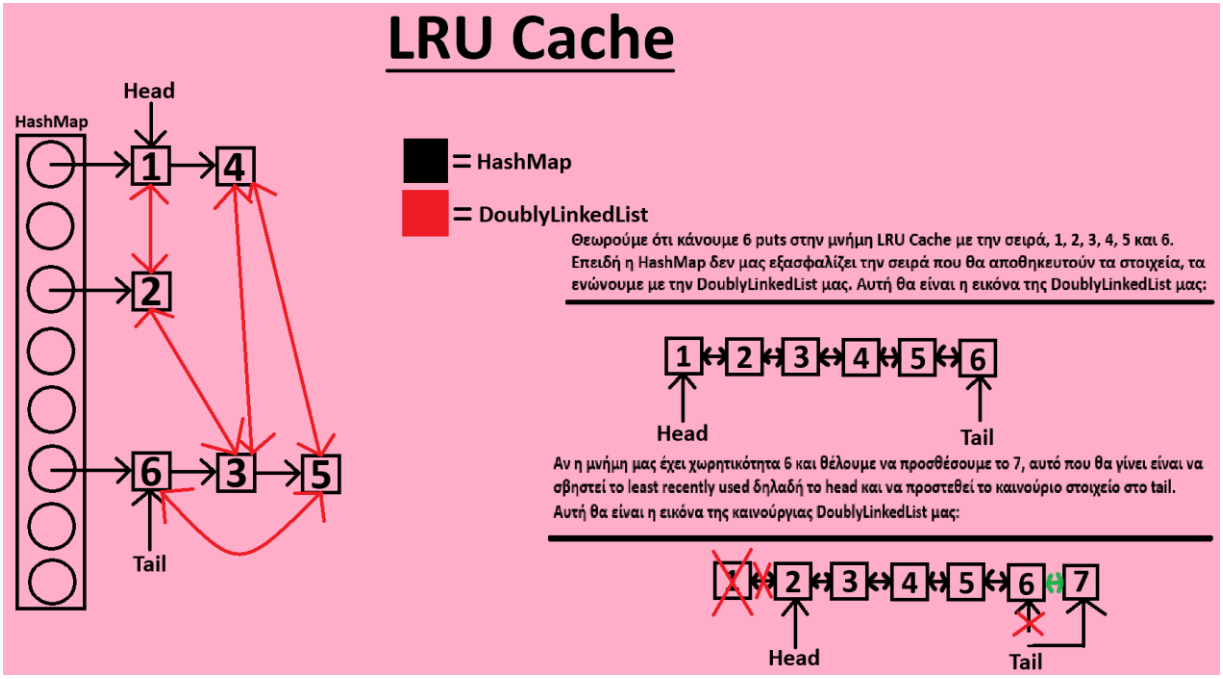
Λειτουργίες της LRU Cache:

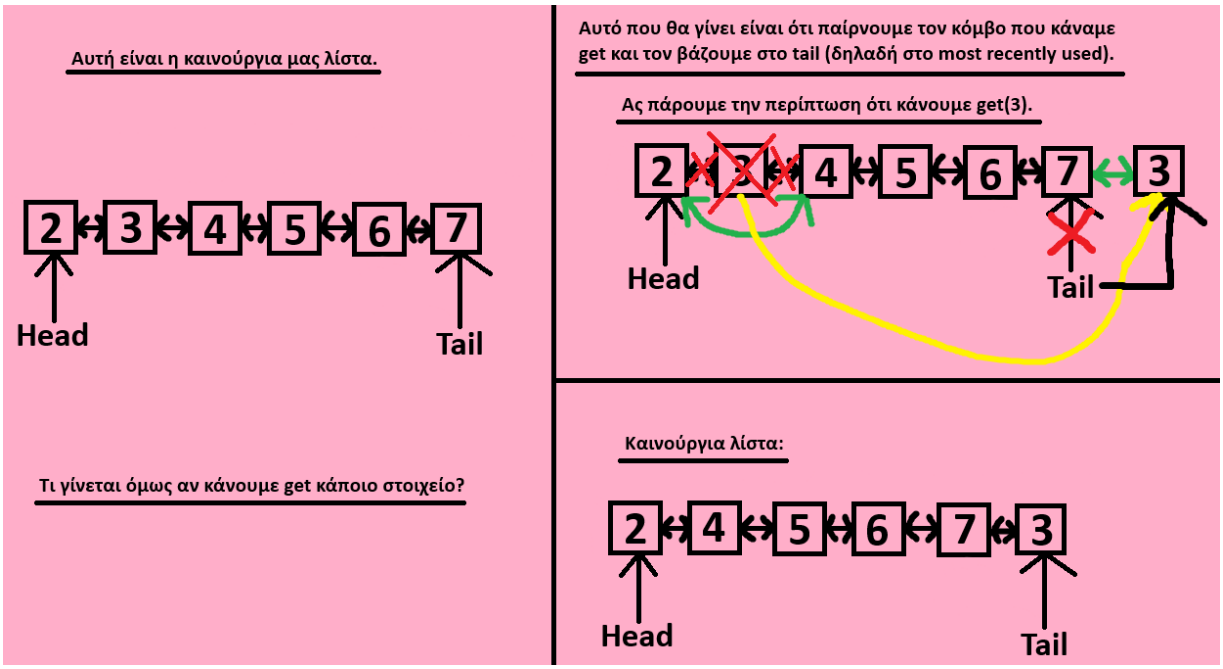
get: Δέχεται ένα κλειδί, τοποθετεί τον κόμβο με αυτό το κλειδί στο τέλος της λίστας (στο most recent) και επιστρέφει την τιμή του κλειδιού. Αν το κλειδί δεν υπάρχει στην μνήμη, επιστρέφει null.

put: Δέχεται ένα κλειδί και μια τιμή, φτιάχνει έναν κόμβο με αυτό το κλειδί και τιμή, τον τοποθετεί στο τέλος της (στο most recent) και τοποθετεί το κλειδί και τον κόμβο στην μνήμη. Αν το κλειδί υπάρχει ήδη στην μνήμη, ανανεώνει στον κόμβο με αυτό το κλειδί την παλιά τιμή με την καινούρια και τον τοποθετεί στο τέλος της λίστας (στο most recent).

put (σε γεμάτη μνήμη): Στην περίπτωση που γίνει put σε γεμάτη μνήμη, πριν την εισαγωγή του καινούριου κόμβου διαγράφεται από την μνήμη ο πρώτος κόμβος της λίστας (το head που είναι το least recent) για να κάνει χώρο για τον καινούριο κόμβο και στην συνέχεια ξεκινάει η λειτουργία put.

Σχήμα για LRU Cache:





Τύπος 2 (MRU Cache):

Work in progress...

Τύπος 3 (LFU Cache):

Work in progress…

2. Tests

Τώρα που εξηγήσαμε την υλοποίηση κάθε τύπου μνήμης cache, πάμε να δούμε τα tests που πρέπει να κάνει κάθε μνήμη για να θεωρηθεί σωστά υλοποιημένη.

Αυτό το πρόγραμμα υποστηρίζει 3 tests για κάθε μνήμη cache:

1) General test
2) Edge cases test
3) Stress test

Test 1 (general test): Ελέγχει την ορθότητα της υλοποίησης της μνήμης cache.

Test 2 (edge cases test): Ελέγχει τις ακριανές τιμές της μνήμης cache.

Test 3 (stress test): Ελέγχει την ανθεκτικότητα της μνήμης cache.

Αυτά τα 3 test γίνονται για κάθε τύπο μνήμης cache από μία φορά.

Τώρα θα δείξουμε πως υλοποιούνται αυτά τα tests.

Υλοποίηση test 1 (general test): Ο τρόπος που το general test ελέγχει την ορθότητα της υλοποίησης της μνήμης cache είναι ελέγχοντας αν η μνήμη βγάζει τα επιθυμητά αποτελέσματα ανάλογα τον τύπο της.

Υλοποίηση test 2 (edge cases test): Ο τρόπος που το edge cases test ελέγχει τις ακριανές τιμές της μνήμης cache είναι ελέγχοντας αν κάθε εισαγωγή καινούργιου στοιχείου αποθηκεύτηκε σωστά και αν διαγράφτηκε το στοιχείο που έπρεπε να διαγραφεί ανάλογα με τον τύπο της cache.

Υλοποίηση test 3 (stress test): Ο τρόπος που το stress test ελέγχει την ανθεκτικότητα της μνήμης cache είναι φορτώνοντας την μνήμη με έναν πάρα πολύ μεγάλο πλήθος στοιχείων και στην συνέχεια ελέγχοντας αν η μνήμη δεν “έσκασε” και αν τα στοιχεία που έμειναν στην μνήμη ήταν αυτά που έπρεπε να μείνουν ανάλογα με τον τύπο της cache.

3. Λειτουργία Hit/Miss

Work in progress...