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).

Δουλειά του προγράμματος είναι να υλοποιεί και τους τρεις τύπου μνήμης cache και να κάνει και κατάλληλα tests για την κάθε μία.

Υπάρχουν τρία είδη test που μπορούμε να κάνουμε για κάθε μνήμη cache.

Tests:

Test 1 (general test): Αυτό το test ελέγχει την ορθότητα της υλοποίησης της μνήμης cache.

Test 2 (edge cases): Αυτό το test ελέγχει της ακριανές τιμές όπως αν βγήκε από την μνήμη cache το σωστό αντικείμενο ή αν έγινε σωστή διαχείριση των κόμβων της λίστας.

Test 3 (stress test): Αυτό το test ελέγχει ανθεκτικότητα της μνήμης. Πιο συγκεκριμένα όπως γίνεται σε αυτό το test, φορτώνεται ένας πολύ μεγάλος αριθμός από αντικείμενα στην μνήμη cache (στην περίπτωσή μας φορτώνει 1.000.000 αντικείμενα) και ελέγχει αν έγινε σωστή διαχείριση της μνήμης.

ΤΥΠΟΙ ΜΝΗΜΗΣ CACHE:

Τύπος 1 (LRU Cache):

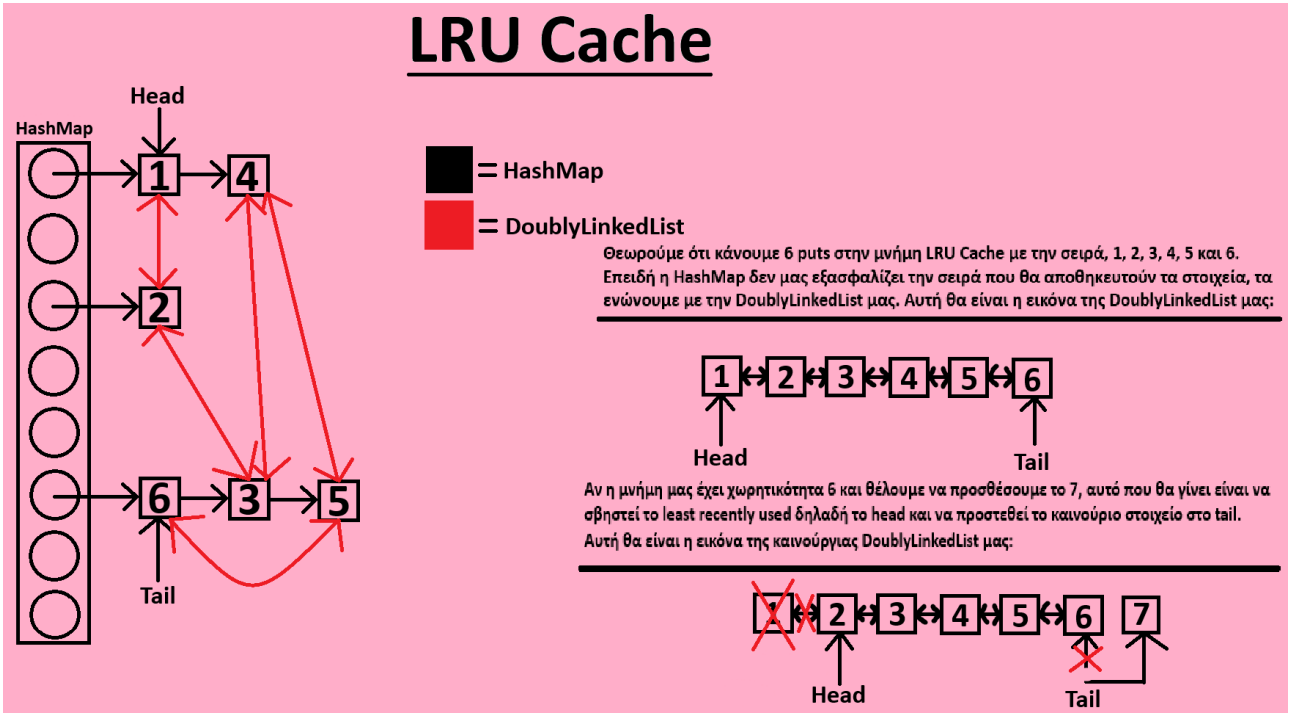
Ο πρώτος τύπος της cache είναι η LRU Cache (least recently use) που το αντικείμενο που θα βγει για να κάνει χώρο είναι αυτό που χρησιμοποιήθηκε λιγότερο πρόσφατα.

Ο τρόπος που θα το υλοποιήσουμε αυτό είναι με την βοήθεια της έτοιμης υλοποίησης πίνακα κατακερματισμού της Java το HashMap και μια δικιά μας υλοποίηση μιας διπλά συνδεδεμένης λίστας σε μία εσωτερική κλάση Node. Η σύνδεση μεταξύ των δύο είναι εφικτή η διπλά συνδεδεμένη λίστα (Node) κουβαλάει το κατάλληλο key και value και το HashMap κουβαλάει σαν

key το ίδιο key με το Node και σαν value δέχεται το ίδιο το Node που βρίσκεται με αυτό το key. 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 κάπου κόμβου με ένα key γίνεται σε O(1) χρόνο. Συνεπώς, O(1) είναι και η διαγραφή γιατί αφού με την αναζήτηση (που είναι O(1)) παίρνουμε τον κόμβο που θέλουμε να διαγράψουμε και το μόνο που μένει είναι να πειράξουμε τα κατάλληλα next και prev για να προσαρμόσουμε την λίστα μας που επίσης είναι O(1). Τέλος, και η πρόσθεση στοιχείου είναι O(1) 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.

Σχήμα για LRU Cache:



<u>Τύπος 2 (MRU Cache):</u>

Work in progress…

<u>Τύπος 3 (LFU Cache):</u>

Work in progress…