

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

Tests

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

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

Test 2 (edge cases): 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): Ελέγχει την ανθεκτικότητα της μνήμης. Πιο συγκεκριμένα ελέγχει αν η μνήμη μπορεί να αντέξει την είσοδο πολλών παραπάνω στοιχείων από την χωρητικότητα της.

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

Τύπου Μνήμης Cache

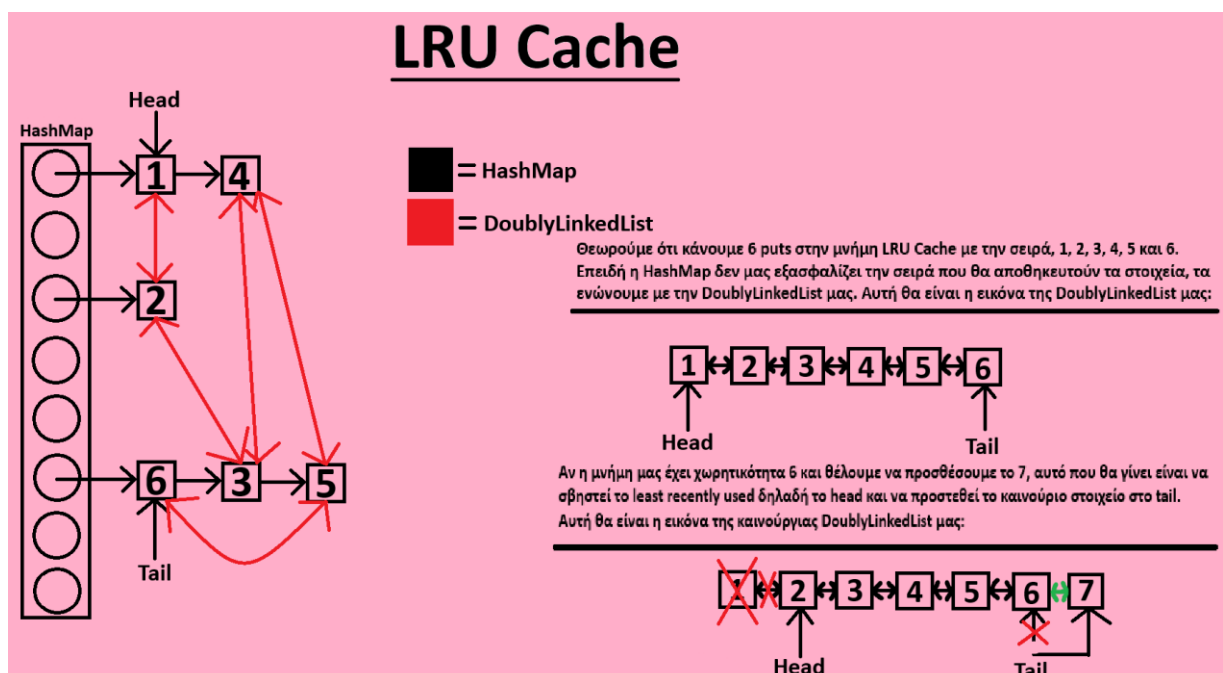
Τύπος 1 (LRU Cache):

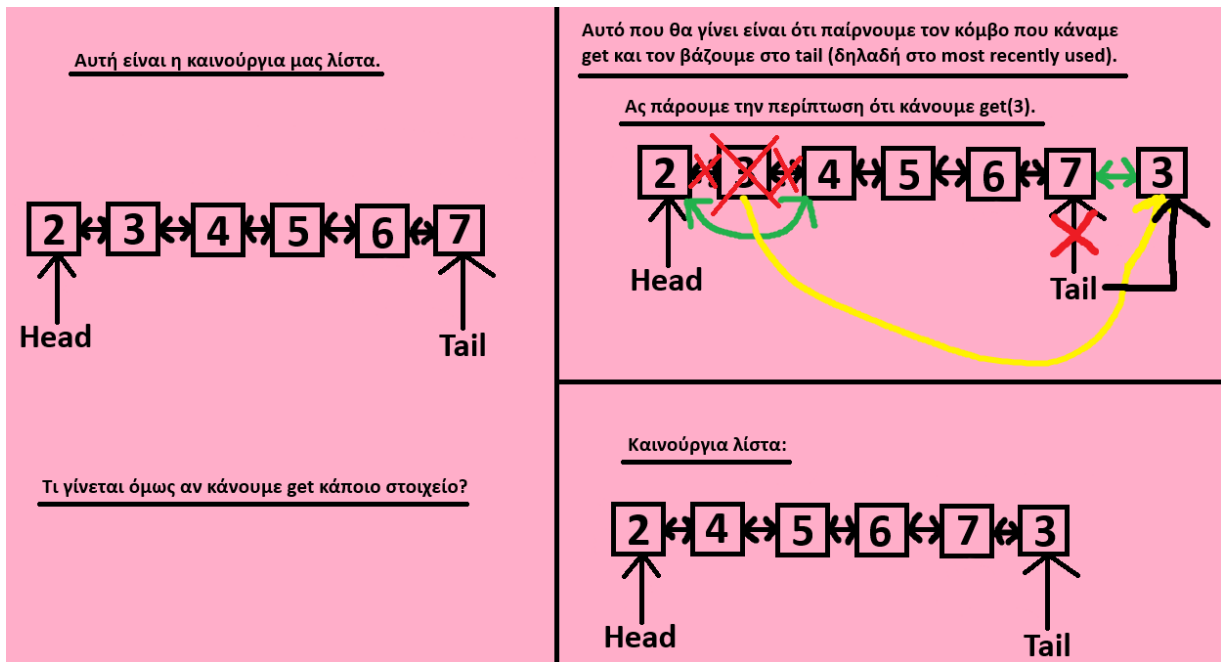
Ο πρώτος τύπος μνήμης cache είναι η LRU (least recently used) Cache που το αντικείμενο που θα βγει για να κάνει χώρο είναι αυτό που χρησιμοποιήθηκε λιγότερο πρόσφατα.

Ο τρόπος υλοποίησης της LRU θα γίνει με την βοήθεια της έτοιμης υλοποίησης πίνακα κατακερματισμού 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, $O(1)$ είναι και η διαγραφή γιατί αφού με την αναζήτηση (που είναι $O(1)$) 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 $O(1)$. Κάθε διαγραφή γίνεται από την αρχή της λίστας (δηλαδή το head) κάθε φορά που γεμίζει. 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 γίνεται σε $O(1)$ χρόνο.

Σχήμα για LRU Cache:





Τύπος 2 (MRU Cache):

Work in progress...

Τύπος 3 (LFU Cache):

Work in progress...