

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).

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.

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

LRU Cache

The diagram illustrates an LRU Cache implementation. On the left, a vertical array of 10 circles represents the **HashMap**. Three of these circles are connected by red arrows to nodes in a **DoublyLinkedList**. The top circle points to node 1, the second circle to node 2, and the eighth circle to node 6. The **DoublyLinkedList** consists of nodes 1, 2, 3, 4, 5, 6, and 7. Node 1 is at the **Head** and node 6 is at the **Tail**. Red arrows show the bidirectional links between adjacent nodes. A legend indicates that black squares represent the **HashMap** and red squares represent the **DoublyLinkedList**.

Head

HashMap

1

2

3

4

5

6

7

Tail

Legend:

- Black square = HashMap
- Red square = DoublyLinkedList

Θεωρούμε ότι κάνουμε 6 puts στην μνήμη LRU Cache με την σειρά, 1, 2, 3, 4, 5 και 6. Επειδή η HashMap δεν μας εξασφαλίζει την σειρά που θα αποθηκευτούν τα στοιχεία, τα ενώνουμε με την DoublyLinkedList μας. Αυτή θα είναι η εικόνα της DoublyLinkedList μας:

Head

1

2

3

4

5

6

Tail

Αν η μνήμη μας έχει χωρητικότητα 6 και θέλουμε να προσθέσουμε το 7, αυτό που θα γίνει είναι να σβηστεί το least recently used δηλαδή το head και να προστεθεί το καινούριο στοιχείο στο tail. Αυτή θα είναι η εικόνα της καινούργιας DoublyLinkedList μας:

Head

2

3

4

5

6

7

Tail

Diagram illustrating a doubly linked list structure. The nodes are 2, 4, 5, 6, 7, and 3. The Head pointer points to node 2, and the Tail pointer points to node 3. Bidirectional arrows connect adjacent nodes, indicating that each node contains pointers to both the previous and next nodes in the sequence.

Τύπος 2 (MRU Cache):

Work in progress...

Τύπος 3 (LFU Cache):

Work in progress...