-ΜΕΡΟΣ 1ο * (8/12-19/12/2024)

IT2023140

IT2023101

IT2023024

Αναφορά Υλοποίησης Κρυφής Μνήμης LRU

Εισαγωγή

Η συγκεκριμένη εργασία που εκτελέστηκε από μια ομάδα 3 ατομων0 έχει σκοπό τη δημιουργία μίας κρυφής μνήμης (cache) με πολιτική αντικατάστασης Least Recently Used (LRU). Η υλοποίηση έγινε με χρήση γλώσσας Java και βασίστηκε στο συνδυασμό μίας δομής πίνακα κατακερματισμού (HashMap) και μίας διπλά συνδεδεμένης λίστας. Στόχος της υλοποίησης είναι η βέλτιστη διαχείριση μνήμης με χρόνο λειτουργίας O(1) τόσο για την εισαγωγής όσο και για την ανάκτηση δεδομένων.

Μεθοδολογία και Τρόπος Σκέψης

1. **Κατανόηση της Πολιτικής LRU**
    a. Η πολιτική Least Recently Used (LRU) απομακρύνει από την κρυφή μνήμη το στοιχείο που έχει χρησιμοποιηθεί λιγότερο πρόσφατα όταν γεμίσει η χωρητικότητα.
    b. Αυτό επιτυγχάνεται με τη χρήση μίας διπλά συνδεδεμένης λίστας για τη διατήρηση της σειράς πρόσβασης και ενός πίνακα κατακερματισμού για γρήγορη ανάκτηση δεδομένων.
2. **Σχεδίαση Δομών Δεδομένων**
    a. Χρησιμοποιήθηκε ένας πίνακας κατακερματισμού (`HashMap`) για την αποθήκευση των ζευγών κλειδιού-τιμής.
    b. Μία διπλά συνδεδεμένη λίστα χρησιμοποιήθηκε για τη διατήρηση της σειράς χρήσης των δεδομένων.
    c. Στο head της λίστας τοποθετούνται τα πιο πρόσφατα χρησιμοποιημένα δεδομένα, ενώ στο τέλος βρίσκονται τα παλαιότερα.
3. **Υλοποίηση της Κλάσης `LRUCache`**

a. Η κλάση υλοποιεί το interface `Cache<K, V>`.
b. Περιλαμβάνει μεθόδους `get(K key)` και `put(K key, V value)` για την ανάκτηση και εισαγωγή δεδομένων.
c. Προστέθηκαν εσωτερικές βοηθητικές μέθοδοι για την ενημέρωση της λίστας κατά την εισαγωγή και την αντικατάσταση δεδομένων.

4. **Οδηγός Υλοποίησης**
    a. **Εισαγωγή (put)**:
        i. Αν το στοιχείο υπάρχει ήδη, μετακινείται στην κεφαλή της λίστας.
        ii. Αν δεν υπάρχει και η κρυφή μνήμη έχει φτάσει στο μέγιστο μέγεθος, αφαιρείται το τελευταίο στοιχείο.
    b. **Ανάκτηση (get)**:
        i. Αν το κλειδί υπάρχει, το στοιχείο μετακινείται στην κεφαλή της λίστας και επιστρέφεται η τιμή του.
        ii. Αν το κλειδί δεν υπάρχει, επιστρέφεται `null`.

Περιγραφή Κώδικα

Κλάση LRUCache

Η κλάση `LRUCache` υλοποιεί τη λειτουργικότητα της κρυφής μνήμης. Ακολουθεί η βασική δομή του κώδικα:

Πεδία

- `capacity`: Το μέγιστο μέγεθος της κρυφής μνήμης.
- `map`: Ένας πίνακας κατακερματισμού για την αποθήκευση των ζευγών κλειδιού-τιμής.
- `head` και `tail`: Δείκτες για την κεφαλή και την ουρά της διπλά συνδεδεμένης λίστας.

Μέθοδοι

1. **`get(K key)`**
    a. Ελέγχει αν το κλειδί υπάρχει στη μνήμη.
    b. Αν το κλειδί υπάρχει, μετακινεί το στοιχείο στην κεφαλή της λίστας και επιστρέφει την τιμή του.
    c. Αν το κλειδί δεν υπάρχει, επιστρέφει `null`.
2. **`put(K key, V value)`**
    a. Εισάγει ένα νέο ζεύγος κλειδιού-τιμής στην κρυφή μνήμη.

b. Αν το στοιχείο υπάρχει ήδη, ενημερώνει την τιμή του και το μετακινεί στην κεφαλή της λίστας.
c. Αν η μνήμη έχει φτάσει στο μέγιστο μέγεθος, αφαιρείται το λιγότερο πρόσφατα χρησιμοποιημένο στοιχείο από την ουρά.

3. **Βοηθητικές Μέθοδοι**
   a. `removeNode(Node<K, V> node)`: Αφαιρεί έναν κόμβο από τη λίστα.
   b. `addToHead(Node<K, V> node)`: Προσθέτει έναν κόμβο στην κεφαλή της λίστας.

Παράδειγμα Εκτέλεσης

Είσοδος και Αποτελέσματα

Παράδειγμα 1

```
@Test
  void testBasicPutAndGet() {
     LRUCache<Integer, String> cache = new LRUCache<>(3);
     cache.put(1, "10");
     cache.put(2, "20");
     cache.put(3, "30");
     assertEquals("10", cache.get(1));
     assertEquals("20", cache.get(2));
     assertEquals("30", cache.get(3));
  }
```

Επεξήγηση

1. Δημιουργείται μια κρυφή μνήμη με μέγιστη χωρητικότητα 3.
   Εισάγονται τα ζεύγη (1, "10"), (2, "20") και (3, "30").
   Η μέθοδος get επιστρέφει τις τιμές:
   Για το κλειδί 1, επιστρέφει "10".
   Για το κλειδί 2, επιστρέφει "20".
   Για το κλειδί 3, επιστρέφει "30".

Παράδειγμα 2

```
@Test
void stressTest() {
   LRUCache<Integer, String> cache = new LRUCache<>(10000);

   for (int i = 0; i < 100000; i++) {
      cache.put(i, "Value" + i);

      if (i % 1000 == 0) {
         assertEquals("Value" + i, cache.get(i));
      }
   }

   for (int i = 0; i < 90000; i++) {
      assertNull(cache.get(i));
   }
}
```

Επεξήγηση

1. Δημιουργείται μια κρυφή μνήμη με χωρητικότητα 10.000.
2. Εισάγονται 100.000 ζεύγη κλειδιών-τιμών, π.χ., (0, "Value0"), (1, "Value1")...
3. Κάθε 1.000 επαναλήψεις, ελέγχεται ότι η τελευταία εισαγωγή παραμένει στη μνήμη.
4. Μετά την εισαγωγή όλων, τα πρώτα 90.000 κλειδιά ελέγχονται και επιβεβαιώνεται ότι έχουν αφαιρεθεί (επιστρέφουν null).

Συμπεράσματα

Η κρυφή μνήμη LRU που υλοποιήθηκε διαχειρίζεται αποδοτικά τα δεδομένα με χρήση των δομών πίνακα κατακερματισμού και διπλά συνδεδεμένης λίστας. Οι λειτουργίες ανάκτησης και εισαγωγής δεδομένων ολοκληρώνονται σε (ψιλοαναμενόμενο) χρόνο O(1), διατηρώντας τη σειρά πρόσβασης των δεδομένων.