-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 κλειδιούτιμής.
- `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

```
LRUCache<Integer, String> cache = new LRUCache<>(3);
cache.put(1, "A"); cache.put(2, "B"); cache.put(3,
"C");
System.out.println(cache.get(1)); // Επιστρέφει "A"
cache.put(4, "D");
System.out.println(cache.get(2)); // Επιστρέφει null (αφαιρέθηκε το
2 λόγω LRU)
```

Επεξήγηση

1. Εισάγονται τα κλειδιά 1, 2, και 3.
2. Γίνεται πρόσβαση στο 1, το οποίο τοποθετείται στην κεφαλή της λίστας.
3. Όταν εισαχθεί το 4, αφαιρείται το 2, καθώς είναι το λιγότερο πρόσφατα χρησιμοποιημένο.

Παράδειγμα 2

```
LRUCache<String, Integer> cache = new LRUCache<>(2);
cache.put("A", 1); cache.put("B", 2);
System.out.println(cache.get("A")); // Επιστρέφει 1
cache.put("C", 3);
System.out.println(cache.get("B")); // Επιστρέφει null (αφαιρέθηκε
το B)
```

Επεξήγηση

1. Εισάγονται τα κλειδιά `"A"` και `"B"`.
2. Το `"A"` παραμένει στη μνήμη λόγω πρόσφατης πρόσβασης.
3. Όταν εισαχθεί το `"C"`, αφαιρείται το `"B"`.

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.

-ΜΕΡΟΣ 2ο * (19/12/24-7/01/25)

Υλοποίηση Λειτουργικότητας Hit/Miss

Στο δεύτερο μέρος της εργασίας, επεκτάθηκε η υλοποίηση της κρυφής μνήμης με τη δυνατότητα μέτρησης των επιτυχιών (hits) και αποτυχιών (misses).

1. Καταμέτρηση Hits και Misses

- Ένας hit σημειώνεται όταν το ζητούμενο στοιχείο υπάρχει ήδη στην κρυφή μνήμη.
- Ένας miss σημειώνεται όταν το στοιχείο δεν υπάρχει και απαιτείται η εισαγωγή του.

Για την υλοποίηση της λειτουργικότητας αυτής, προστέθηκαν οι μέθοδοι:
- getHitCount(): Επιστρέφει τον συνολικό αριθμό επιτυχιών.
- getMissCount(): Επιστρέφει τον συνολικό αριθμό αποτυχιών.

2. Δημιουργία Τυχαίου Σεναρίου

Ένα πρόγραμμα προσομοίωσης δημιουργήθηκε ώστε να παράγει τυχαία αιτήματα στην κρυφή μνήμη, βασισμένο στο σενάριο κατανομής 80/20. Σε αυτό:
- Το 80% των αιτημάτων αφορά συγκεκριμένα δεδομένα (hot keys).
- Το υπόλοιπο 20% αφορά τυχαία δεδομένα.

Παράδειγμα Εκτέλεσης:MRU

Total operations: 100,000
Cache Hits: 27,762
Cache Misses: 72,238
Hit Rate: 27.76%
Miss Rate: 72.24%

Παράδειγμα Εκτέλεσης:LRU

Total operations: 100,000
Cache Hits: 81,018
Cache Misses: 18,982
Hit Rate: 81.02%
Miss Rate: 18.98%

Πολλαπλές Στρατηγικές Αντικατάστασης

Για τη βελτίωση της υλοποίησης, προστέθηκε υποστήριξη για εναλλακτικές στρατηγικές αντικατάστασης.

1. Εισαγωγή της Πολιτικής MRU

Στη στρατηγική MRU (Most Recently Used), το πιο πρόσφατα χρησιμοποιημένο στοιχείο αφαιρείται πρώτο όταν γεμίσει η χωρητικότητα.

2. Επεκτάσεις στην Υλοποίηση

Προστέθηκε το enumeration CacheReplacementPolicy, που επιτρέπει την επιλογή πολιτικής αντικατάστασης κατά τη δημιουργία της κρυφής μνήμης. Το πρόγραμμα προσομοίωσης εκτελέστηκε ξανά με τη νέα πολιτική αντικατάστασης και τα αποτελέσματα συγκρίθηκαν με την πολιτική LRU.

3. Σύγκριση Αποτελεσμάτων

LRU:
Hit Rate: 81.02%
Miss Rate: 18.98%

MRU:
Hit Rate: 27.76%
Miss Rate: 72.24%

Συμπεράσματα για το Δεύτερο Μέρος

Η προσθήκη των λειτουργιών hit/miss και των εναλλακτικών στρατηγικών αντικατάστασης βελτίωσε την ευελιξία της κρυφής μνήμης. Η στρατηγική LRU

επιβεβαίωσε την αποτελεσματικότητά της για σενάρια 80/20, ενώ η στρατηγική MRU παρουσίασε διαφορετικά χαρακτηριστικά, χρήσιμα σε συγκεκριμένες περιπτώσεις χρήσης.