Ομαδική εργασία (Μέρος 3ο) 2024

Δομές Δεδομένων

Νικολέττα Ναντιέζντα Βόσσου, it2023113@hua.gr
Αθανάσιος Κολλάς, it2023030@hua.gr

Περιεχόμενα:

Εισαγωγή 1
Περιγραφή 1
- Διεπαφή Cache<K, V> 1
- Κλάση MultiPolicyCache 1
- Κλάση App 2
- Κλάση CacheReplacementPolicy 2
- Κλάση MultiPolicyCacheTest 2

Λειτουργίες την MultiPolicyCache 3
- put(K key, V value): 3
- get(K key): 4
- clear(): 4
- updateFrequency(Node node): 4

Λειτουργίες την App: 5
Ψευδοκώδικας για την προσθήκη νέου στοιχείου (put(Integer key, Integer value)): 5
Παράδειγμα Εκτέλεσης Κώδικα 7
- Χρήση LRU (Least Recently Used) 7
- Χρήση MRU (Most Recently Used) 8
- Χρήση LFU(Least Frequently Used) 8
- Εμφάνιση Hit/Miss Rate 10
- Τυχαία Στοιχεία 11

Συμπεράσματα 12

Αναφορά

Εισαγωγή

Το έργο αυτό πραγματεύεται την υλοποίηση δύο τύπων δομών δεδομένων προσωρινής αποθήκευσης: LRUCache (Least Recently Used Cache) και MRUCache (Most Recently Used Cache). Αυτοί οι μηχανισμοί χρησιμοποιούνται για τη βελτιστοποίηση της χρήσης μνήμης και χρόνου σε πολλές εφαρμογές.Η εφαρμογή συνοδεύεται από ένα πρόγραμμα με διεπαφή κονσόλας που επιτρέπει στον χρήστη να πειραματιστεί με τις λειτουργίες της cache.

Περιγραφή

Η εφαρμογή αποτελείται από πέντε βασικά μέρη:

- **Διεπαφή Cache<K, V>**

Αυτή η διεπαφή περιγράφει τις βασικές λειτουργίες μιας cache:

- get(K key): Επιστρέφει την τιμή που αντιστοιχεί σε ένα κλειδί.
- put(K key, V value): Εισάγει ή ενημερώνει ένα ζεύγος κλειδιού-τιμής.
- size(): Επιστρέφει τον αριθμό των στοιχείων που βρίσκονται αυτήν τη στιγμή στην cache.
- capacity(): Επιστρέφει τη μέγιστη χωρητικότητα της cache.
- clear(): Καθαρίζει όλα τα δεδομένα από την cache.

- **Κλάση MultiPolicyCache**

Η συγκεκριμένη κλάση υλοποιεί τη διεπαφή Cache<K, V> και περιλαμβάνει τα εξής χαρακτηριστικά:

- Χρήση μιας διπλά συνδεδεμένης λίστας για τη διαχείριση της σειράς των στοιχείων.

Η λίστα χρησιμοποιείται για την αποθήκευση των στοιχείων με τη σειρά χρήσης τους. Ο πιο πρόσφατα χρησιμοποιημένος κόμβος βρίσκεται στο τέλος της λίστας (tail), ενώ ο λιγότερο πρόσφατα χρησιμοποιημένος βρίσκεται στην αρχή (head).

- Έναν HashMap για την αποθήκευση των δεδομένων, που επιτρέπει γρήγορη αναζήτηση με βάση τα κλειδιά.

Ο χάρτης παρέχει γρήγορη πρόσβαση στα στοιχεία με βάση τα κλειδιά τους. Η λειτουργία του σε συνδυασμό με τη λίστα εξασφαλίζει χρόνο εκτέλεσης O(1) για τις λειτουργίες get και put (LRU, MRU) και O(logn) για την LFU.

- Ειδικούς ψευδο-κόμβους (head και tail) για την εύκολη διαχείριση της λίστας.
- Μηχανισμούς για την εφαρμογή των πολιτικών LRU, LFU και MRU, δηλαδή είτε την απομάκρυνση του λιγότερο πρόσφατα χρησιμοποιημένου αντικειμένου, είτε την απομάκρυνση του συχνότερα πρόσφατα χρησιμοποιημένου αντικειμένου, είτε το αντικείμενο που έχει χρησιμοποιηθεί τις λιγότερες φορές.

Στο 2ο μέρος εργασίας έχουμε αλλάξει τον constructor (κάναμε η πρόσθεση τον μεταβλητή replacementPolicy).

Στο 3ο μέρος εργασίας αλλάξαμε την κλάση CacheReplacementPolicy, προσθέσαμε τον TreeMap freqMap (TreeMap<Integer, Map<Integer, Node>>). Δημιουργήθηκε επίσης μια νέα μέθοδος updateFrequency με παράμετρο συχνότητας στον κόμβο με αρχική τιμή 1, καθώς και αλλαγή στις μεθόδους get και put για την ανίχνευση της πολιτικής LFU και την εκτέλεση των απαιτούμενων ενεργειών για την υλοποίηση αυτής της πολιτικής. Είναι σημαντικό να σημειωθεί ότι έχουν υλοποιηθεί επίσης δοκιμές για τον νέο τύπο πολιτικής και έχει προστεθεί μια επιλογή για τη δοκιμή αυτής της πολιτικής από την πλευρά του χρήστη στην κλάση App.

- **Κλάση App**

Η κλάση App αποτελεί τη βασική κλάση εφαρμογής που παρέχει μια διεπαφή κονσόλας για τη διαχείριση και τον πειραματισμό με τις δομές δεδομένων LRU, LFU και MRU Cache. Ο σχεδιασμός της επιτρέπει τη διαδραστική εκτέλεση βασικών λειτουργιών, όπως εισαγωγή δεδομένων, ανάκτηση, εκκαθάριση, αξιολόγηση απόδοσης της cache κλπ.

- **Κλάση CacheReplacementPolicy**

Η κλάση CacheReplacementPolicy ορίζει τις στρατηγικές αντικατάστασης δεδομένων στη μνήμη cache. Οι διαθέσιμες στρατηγικές περιλαμβάνουν:

1. LRU (Least Recently Used)
2. MRU (Most Recently Used)
3. LFU (Least Frequenly Used)

- **Κλάση MultiPolicyCacheTest**

Η κλάση LruMruCacheTest περιλαμβάνει JUnit τεστ για να επαληθεύσει τη σωστή λειτουργία της cache με τις πολιτικές LRU και MRU.

Βασικές Δοκιμές:

1. **Δοκιμή Πολιτικής LRU (Least Recently Used):**

- **Σκοπός:** Επαληθεύει την απομάκρυνση του λιγότερο πρόσφατα χρησιμοποιημένου στοιχείου και επιβεβαιώνει ότι η cache παραμένει εντός της χωρητικότητας.
- **Διαδικασία**:
  - Προστίθενται στοιχεία στην cache.
  - Ελέγχεται η απομάκρυνση του στοιχείου με την χαμηλότερη συχνότητα πρόσβασης όταν υπερβαίνεται η χωρητικότητα.
  - Επαληθεύεται ότι το στοιχείο με την λιγότερη πρόσβαση απομακρύνεται σωστά.

2. **Δοκιμή Πολιτικής MRU (Most Recently Used):**

- **Σκοπός**: Επαληθεύει την απομάκρυνση του πιο πρόσφατα χρησιμοποιημένου στοιχείου και διασφαλίζει τη σωστή διαχείριση προσβάσεων.
- **Διαδικασία**:
  - Προστίθενται στοιχεία στην cache.
  - Ελέγχεται η απομάκρυνση του πιο πρόσφατα χρησιμοποιημένου στοιχείου όταν υπερβαίνεται η χωρητικότητα.

- Επαληθεύεται ότι το πιο πρόσφατο στοιχείο απομακρύνεται σωστά.

3. **Δοκιμή Πολιτικής LFU (Least Frequently Used):**

- **Σκοπός**: Επαληθεύει την απομάκρυνση του λιγότερο συχνά χρησιμοποιημένου στοιχείου και διασφαλίζει την ορθή λειτουργία της πολιτικής LFU.
- **Διαδικασία**:
  - Προστίθενται στοιχεία στην cache.
  - Ελέγχεται η συχνότητα πρόσβασης στα στοιχεία.
  - Επαληθεύεται ότι το στοιχείο με τη χαμηλότερη συχνότητα πρόσβασης απομακρύνεται σωστά όταν υπερβαίνεται η χωρητικότητα.

4. **Μετρητές Hit και Miss:**

- **Σκοπός**: Ελέγχει την ορθή καταγραφή προσβάσεων που βρίσκουν ή δεν βρίσκουν στοιχεία στη cache.
- **Διαδικασία**:
  - Ελέγχεται η σωστή καταγραφή των hit (επιτυχείς προσβάσεις) και miss (αποτυχημένες προσβάσεις) όταν γίνεται πρόσβαση σε υπάρχοντα και μη υπάρχοντα στοιχεία της cache.

5. **Stress Tests:**

- **Σκοπός**: Δοκιμάζει τη λειτουργία της cache με εκατομμύρια τυχαίες λειτουργίες και εξετάζει περιπτώσεις με "hot keys" σύμφωνα με την κατανομή 80/20.
- **Διαδικασία**:
  - Εκτελούνται εκατομμύρια τυχαίες προσθήκες και προσβάσεις στη cache.
  - Εξετάζεται η απόδοση της cache όταν χρησιμοποιούνται "hot keys", δηλαδή κλειδιά που χρησιμοποιούνται συχνότερα από το 80% των προσβάσεων.

6. **Edge Cases:**

- **Σκοπός**: Εξετάζει ακραίες περιπτώσεις, όπως την διαχείριση της cache όταν έχει μέγεθος 1 ή μηδενική χωρητικότητα.
- **Διαδικασία**:
  - Ελέγχεται η λειτουργία της cache όταν η χωρητικότητα είναι 1.
  - Ελέγχεται η σωστή διαχείριση εξαιρέσεων όταν η χωρητικότητα είναι 0 (θα πρέπει να πετάγεται εξαίρεση).

Λειτουργίες την MultiPolicyCache

- **put(K key, V value):**

Για την LRUCache:

Εάν το κλειδί υπάρχει ήδη στη cache, το στοιχείο ενημερώνεται και μετακινείται στο τέλος της λίστας.

Εάν το κλειδί δεν υπάρχει και η cache έχει φτάσει τη χωρητικότητά της, απομακρύνεται το στοιχείο στην αρχή της λίστας πριν προστεθεί το νέο.

Για την MRUCache:

Εάν το κλειδί υπάρχει ήδη στη cache, το στοιχείο ενημερώνεται και μετακινείται στο τέλος της λίστας.

Εάν το κλειδί δεν υπάρχει και η cache έχει φτάσει τη χωρητικότητά της, απομακρύνεται το στοιχείο στο τέλος της λίστας πριν προστεθεί το νέο.

Για την LFUCache (Least Frequently Used):

Εάν το κλειδί υπάρχει ήδη στη cache, το στοιχείο ενημερώνεται και η συχνότητά του αυξάνεται.

Εάν το κλειδί δεν υπάρχει και η cache έχει φτάσει τη χωρητικότητά της, απομακρύνεται το στοιχείο με τη χαμηλότερη συχνότητα πρόσβασης (ή το πιο παλιό στοιχείο αν υπάρχουν ισοδύναμα συχνά χρησιμοποιούμενα στοιχεία).

Η συχνότητα των στοιχείων παρακολουθείται μέσω του `freqMap`, το οποίο οργανώνει τα στοιχεία κατά συχνότητα.

- **get(K key):**

Εάν το κλειδί υπάρχει στη cache:

- Το στοιχείο μετακινείται στο τέλος της λίστας (για LRU και MRU) ή η συχνότητά του αυξάνεται (για LFU).
- Επιστρέφεται η τιμή του στοιχείου.
- Ο μετρητής hitCount αυξάνεται κατά 1.

Εάν το κλειδί δεν υπάρχει στη cache:

- Επιστρέφεται null.
- Ο μετρητής missCount αυξάνεται κατά 1.

- **clear():**

Καθαρίζει όλα τα στοιχεία της cache, επαναφέροντας τη λίστα και τον χάρτη στην αρχική τους κατάσταση.

Επαναφέρει τη λίστα και τον χάρτη στην αρχική τους κατάσταση. Οι κόμβοι αποσυνδέονται και η cache γίνεται κενή. Η χωρητικότητα και οι μετρητές (hitCount, missCount) επαναφέρονται στην αρχική τους τιμή.

- **updateFrequency(Node node):**

Αυξάνει τη συχνότητα του στοιχείου κατά 1 όταν το στοιχείο ανακτάται από τη cache. Το στοιχείο αφαιρείται από τη λίστα της παλιάς συχνότητας και προστίθεται στη λίστα της νέας συχνότητας. Η

συχνότητα παρακολουθείται με τη χρήση του freqMap, το οποίο είναι ένας χάρτης που οργανώνει τα στοιχεία κατά συχνότητα πρόσβασης.

Κλπ.

Λειτουργίες την App:

Η κλάση App υποστηρίζει τις εξής λειτουργίες

1. **Δημιουργία Cache**
   - Ο χρήστης μπορεί να επιλέξει:
     - Τη χωρητικότητα της cache.
     - Την πολιτική αντικατάστασης (LRU, LFU ή MRU).
   - Υπάρχει δυνατότητα δημιουργίας cache με τυχαία δεδομένα.
2. **Διαχείριση Cache**

   - Εισαγωγή (Put): Εισάγει ένα ζεύγος κλειδιού-τιμής στη cache.
   - Ανάκτηση (Get): Αναζητεί ένα κλειδί και επιστρέφει την αντίστοιχη τιμή, αν υπάρχει.
   - Εκκαθάριση (Clear): Καθαρίζει όλα τα δεδομένα από τη cache.
   - Εμφάνιση Μεγέθους (Size): Προβάλλει το τρέχον μέγεθος της cache.
   - Εμφάνιση Χωρητικότητας (Capacity): Εμφανίζει τη μέγιστη χωρητικότητα της cache.
3. **Αξιολόγηση Απόδοσης και Κανόνας 80/20**
   - Η κατανομή 80/20 περιγράφει μια κατάσταση όπου:
     1. 80% των προσβάσεων γίνεται σε 20% των δεδομένων (hot keys).
     2. 20% των προσβάσεων γίνεται στα υπόλοιπα 80% (cold keys).

     Στον δικό μας κώδικα η λογική των hot keys είναι η εξής:

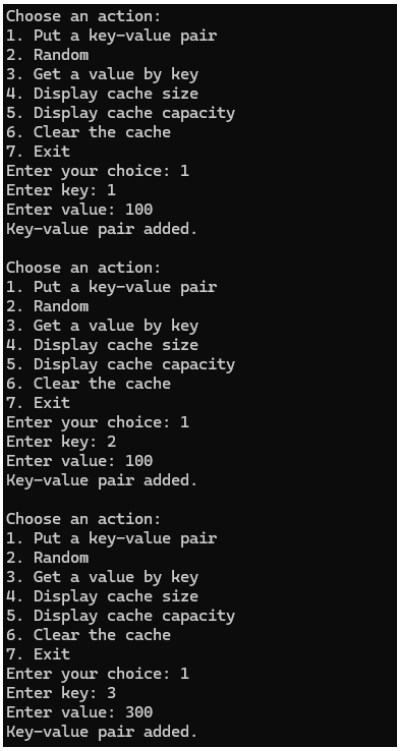
     Το μισό του πίνακα είναι από την κρυφή μνήμη μας και το άλλο μισό είναι από την random.
   - Παρέχεται δυνατότητα εκτέλεσης τυχαίων αναζητήσεων για την αξιολόγηση της αποτελεσματικότητας της cache.
   - Υπολογίζονται και εμφανίζονται στατιστικά όπως:
     - Cache Hits: Πόσα αιτήματα εξυπηρετήθηκαν από την cache.
     - Cache Misses: Πόσα αιτήματα δεν βρέθηκαν στην cache.
     - Hit Rate: Ποσοστό επιτυχημένων αναζητήσεων.
     - Miss Rate: Ποσοστό αποτυχημένων αναζητήσεων.

Ψευδοκώδικας για την προσθήκη νέου στοιχείου (put(Integer key, Integer value)):

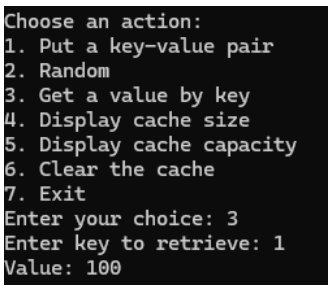
1. Έλεγχος αν το κλειδί υπάρχει ήδη στη cache:
    - Εάν υπάρχει:
        1. Ελέγχουμε αν έχει επιλεγεί η πολιτική LFU και αν ναι, ενημερώνουμε τη συχνότητα (UpdateFrequency(Node node)).
        2. Αφαιρούμε τον αντίστοιχο κόμβο από τη λίστα.
        3. Εισάγουμε το νέο στοιχείο στον χάρτη (nodeMap).
2. Διαχείριση χωρητικότητας:
    - Εάν το μέγεθος της cache είναι ίσο με τη μέγιστη χωρητικότητα:
        - Εάν η πολιτική αντικατάστασης είναι LRU:
            1. Αφαιρούμε τον κόμβο που βρίσκεται στην αρχή της λίστας (head.next).
        - Εάν η πολιτική αντικατάστασης είναι MRU:
            1. Αφαιρούμε τον κόμβο που βρίσκεται στο τέλος της λίστας (tail.prev).
        - Εάν η πολιτική αντικατάστασης είναι LFU:
            1. Βρίσκουμε το πρώτο κλειδί στο freqMap που αναφέρεται σε ένα αντικείμενο/χάρτη αντικειμένων με τη χαμηλότερη συχνότητα και το αποθηκεύουμε στο leastFreq.
            2. Αναζητούμε το χάρτη αντικειμένων ή το ένα αντικείμενο με το κλειδί leastFreq και το αποθηκεύουμε στο nodesLeastFreq.
            3. Παίρνουμε το πρώτο αντικείμενο από τη λίστα και το αποθηκεύουμε στο keyToDel.
            4. Διαγράφουμε το αντικείμενο με το κλειδί keyToDel από το χάρτη nodesLeastFreq και αν ο χάρτης είναι τώρα κενός, διαγράφουμε όλα τα στοιχεία με το κλειδί leastFreq από το freqMap.
            5. Αφαιρούμε τον κόμβο με το κλεδί nodeToRemove.
3. Προσθήκη του νέου στοιχείου:
    - Εάν η πολιτική αντικατάστασης είναι LFU:
        - Δημιουργούμε νέο κόμβο με το κλειδί και την τιμή.
        - Αν δεν υπάρχει ο κλειδί newFreq στο freqMap, δημιουργούμε έναν νέο χάρτη και τον προσθέτουμε με το κλειδί newFreq.
        - Στη συνέχεια, προσθέτουμε το node με το κλειδί node.key και την τιμή node στον χάρτη που αντιστοιχεί στο newFreq.
        - Προσθέτουμε τον κόμβο στο τέλος της λίστας (κοντά στο tail).
4. Ενημέρωση του χάρτη:
    - Εισάγουμε το νέο στοιχείο στον χάρτη (nodeMap).

Παράδειγμα Εκτέλεσης Κώδικα

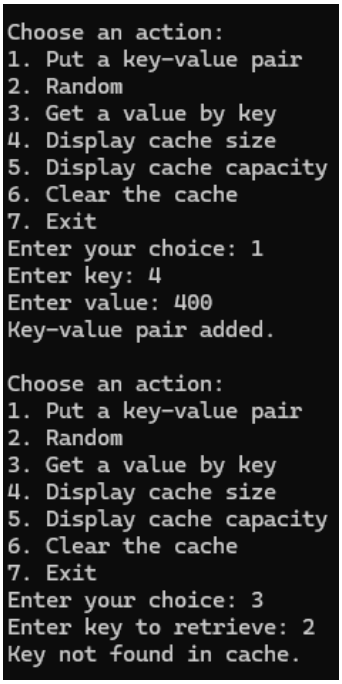
- **Χρήση LRU (Least Recently Used)**



1 - Προσθήκη 3 στοιχείων



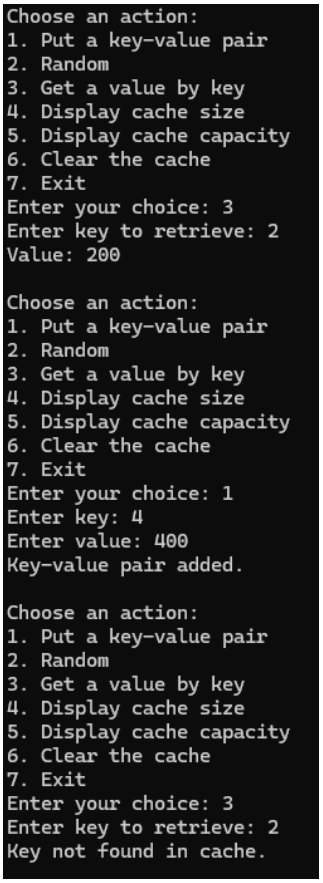
2 - Πρόσβαση στο στοιχείο με κλειδί 1



3 - Προσθήκη νέου στοιχείου, ξεπερνώντας τη χωρητικότητα (το 2 διαγράφηκε λόγω LRU)

- **Χρήση MRU (Most Recently Used)**

Πάλι κάνουμε την Προσθήκη 3 στοιχείων. Επομένως κάνουμε πρόσβαση στο στοιχείο με κλειδί 2 και προσθήκη νέου στοιχείου, ξεπερνώντας τη χωρητικότητα (το 2 διαγράφηκε λόγω MRU):



4 – MRU

- **Χρήση LFU(Least Frequently Used)**

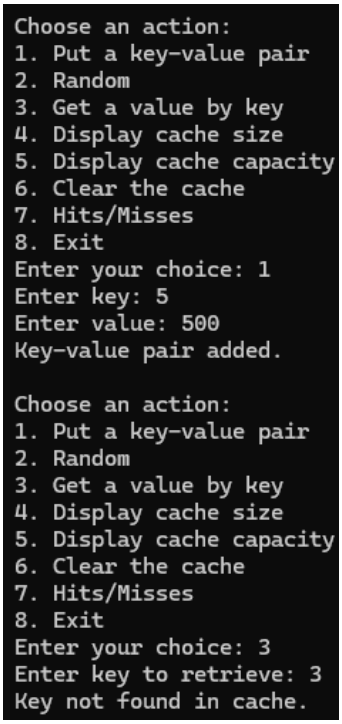
Δημιουργούμε 3 αντικείμενα: (1,100), (2,200), (3,300). Η συχνότητα είναι αυτόματα 1, επομένως, αν αποφασίσουμε να προσθέσουμε έναν ακόμη κόμβο (4,400), θα αφαιρεθεί ο πρώτος από όλους τους κόμβους με συχνότητα 1, δηλαδή (1,100). Αν πρώτα αυξήσουμε τη συχνότητα χρήσης για (4,400) και (2,200), τότε κατά την προσθήκη του επόμενου κόμβου, θα αφαιρεθεί (3,300).

```
Choose an action:
1. Put a key-value pair
2. Random
3. Get a value by key
4. Display cache size
5. Display cache capacity
6. Clear the cache
7. Hits/Misses
8. Exit
Enter your choice: 1
Enter key: 4
Enter value: 400
Key-value pair added.

Choose an action:
1. Put a key-value pair
2. Random
3. Get a value by key
4. Display cache size
5. Display cache capacity
6. Clear the cache
7. Hits/Misses
8. Exit
Enter your choice: 3
Enter key to retrieve: 1
Key not found in cache.
```

5 - πρώτη περίπτωση

```
Choose an action:
1. Put a key-value pair
2. Random
3. Get a value by key
4. Display cache size
5. Display cache capacity
6. Clear the cache
7. Hits/Misses
8. Exit
Enter your choice: 3
Enter key to retrieve: 4
Value: 400

Choose an action:
1. Put a key-value pair
2. Random
3. Get a value by key
4. Display cache size
5. Display cache capacity
6. Clear the cache
7. Hits/Misses
8. Exit
Enter your choice: 3
Enter key to retrieve: 2
Value: 200
```

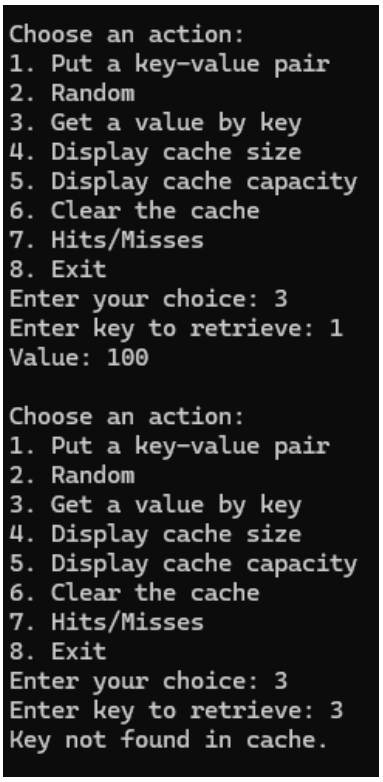
6 - δεύτερη περίπτωση



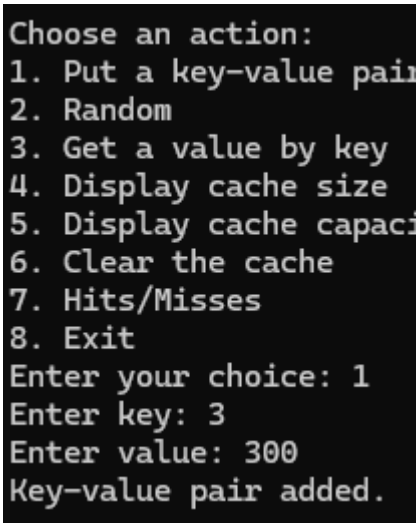
7 - δεύτερη περίπτωση

- **Εμφάνιση Hit/Miss Rate**

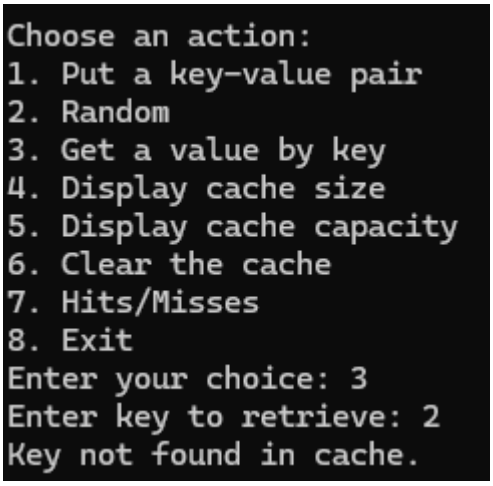
Για αυτό το παράδειγμα πρεπεί να αλλάξουμε λίγο τον κώδικα: να προσθέσουμε case(7. Hits/Misses). Προσθήκη στοιχείων στην κρυφή μνήμη με το μέγεθος 2 — (1,100),(2,200).



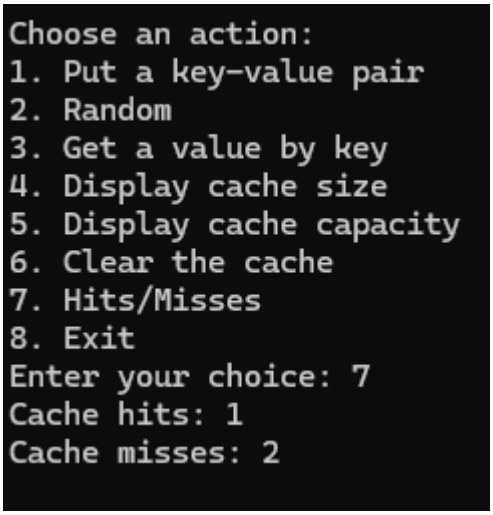
8 - 1 hit + 1 miss



9 - Προσθήκη νέου στοιχείου

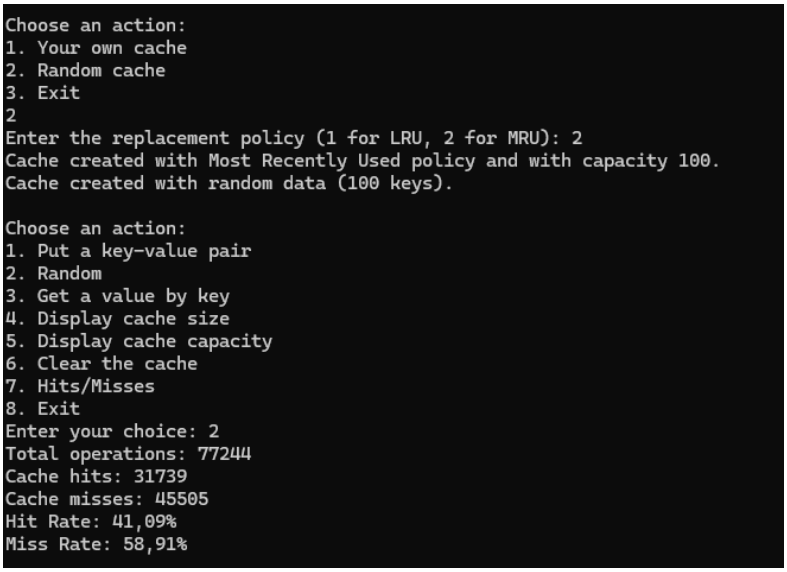


10 - 1 miss



11 - αποτελέσματα

- **Τυχαία Στοιχεία**



12 - Random σεναριο

Δημιουργία την κρυφή μνήμη (Random cache, το μεγεθος είναι 100) και το run του σεναριου Random. Βλέπουμε την εμφάνιση του ζιτούμενου 2.1:

“Total operations: 100000

Cache Hits: 52300

Cache Misses: 47700

Hit Rate: 52.30%

Miss Rate: 47.70%”

Συμπεράσματα

Η MultiPolicyCacheπροσφέρει:

- Αποδοτική διαχείριση δεδομένων με χρόνο πρόσβασης O(logn).
- Ευέλικτη διεπαφή που επιτρέπει τη χρήση της σε διαφορετικές εφαρμογές.
- Ένα αξιόπιστο παράδειγμα για την κατανόηση της πολιτικής LRU, LFU και πολιτικής MRU.