ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 4

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2021-2022

Διδάσκων: Ε. Μαρκάκης

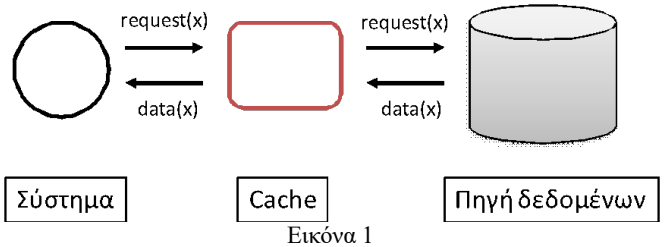
Προθεσμία παράδοσης: Τετάρτη, 2 Μαρτίου 2022 και ώρα 23:59.

Υλοποίηση LRU Cache

Στην εργασία αυτή ζητείται να υλοποιήσετε σε Java τις βασικές μεθόδους που απαιτούνται για τη διαχείριση μίας μνήμης Cache γενικού σκοπού που θα εφαρμόζει την πολιτική αντικατάστασης Least Recently Used – LRU.

Γενικά περί Caching

Στην Επιστήμη Υπολογιστών, οι μνήμες cache είναι βοηθητικές μνήμες που χρησιμοποιούνται για την προσωρινή αποθήκευση δεδομένων για μελλοντική χρήση. Σε γενικές γραμμές, μία cache χρησιμοποιείται σε περιπτώσεις που κάποιο πρόγραμμα/σύστημα αναζητά δεδομένα από μια πηγή δεδομένων (π.χ. αναζήτηση αρχείων από σκληρό δίσκο ή βάση δεδομένων, πληροφορίες από το Internet, δείτε εικόνα 1). Το πρόγραμμα αποθηκεύει την πληροφορία στην cache και την επόμενη φορά που αναζητά δεδομένα από την πηγή δεδομένων, τα αναζητά πρώτα στην cache. Αν τα δεδομένα είναι στην cache (*cache hit*), το σύστημα εξοικονομεί χρόνο καθώς δεν απαιτείται να ανατρέξει στην πηγή δεδομένων για να αναζητήσει τα δεδομένα (είναι πολύ πιο γρήγορο να πάρουμε τα δεδομένα από την cache παρά από την πηγή). Στην περίπτωση που τα δεδομένα δεν βρίσκονται στην cache (*cache miss*), το σύστημα ανακτά τα δεδομένα από την πηγή και τα αποθηκεύει στην cache για μελλοντικές αναζητήσεις.



Εικόνα 1

Για να είναι αποδοτική η χρήση μιας cache, θα πρέπει ο χρόνος αναζήτησης στην περιοχή μνήμης της cache να είναι αρκετά μικρότερος από τον χρόνο που απαιτείται για την ανάκτηση δεδομένων από την πηγή δεδομένων. Για παράδειγμα, μια cache στο σύστημα αρχείων του υπολογιστή χρησιμοποιεί την κύρια μνήμη (RAM) που είναι γρηγορότερη από το σκληρό δίσκο. Επίσης, οι web browsers αποθηκεύουν προσωρινά αρχεία web σελίδων (έγγραφα html, εικόνες, flash objects) στον τοπικό υπολογιστή για να γλιτώσουν χρόνο σε σχέση με το να τα μεταφέρουν από τον server που φιλοξενεί ένα web site. Με αυτό τον τρόπο, εξοικονομείται χρόνος για την ανάκτηση δεδομένων, με τίμημα το κόστος της μνήμης που απαιτεί η cache (ως συνήθως, έχουμε time vs. space tradeoff).
Η τεχνική του caching χρησιμοποιείται ευρύτατα στα συστήματα υπολογιστών. Παραδείγματα εφαρμογής caching είναι:

- Στα chip επεξεργαστών (CPU caching [1])
- Στα λειτουργικά συστήματα (Page cache [2])
- Στις βάσεις δεδομένων (Database caching [3])
- Στο World Wide Web (HTTP caching [4])

Λειτουργία μιας Read-only Cache

Μια cache μόνο για ανάγνωση χρησιμοποιείται ως ενδιάμεσος χώρος γρήγορης πρόσβασης. Στην εικόνα 2 φαίνεται η δομή της.

Όνομα	Δεδομένα
X	00000000011111100000000111111
Y	00011100011110001001110000000
Z	00001111010101001110101111001
...	

Εικόνα 2

Η cache αποθηκεύει τα δεδομένα της εκάστοτε εφαρμογής, μαζί με κάποιο κλειδί που χρησιμοποιείται ως όνομα. Ας θεωρήσουμε στα πλαίσια της εργασίας ότι η cache αποθηκεύει αντικείμενα, τα οποία έχουν ένα πεδίο που χρησιμοποιείται ως κλειδί. Η δομή της cache πρέπει να υποστηρίζει κυρίως 2 λειτουργίες:

1. LookUp(x): αναζητά στο χώρο μνήμης της cache αντικείμενα με το κλειδί x. Αν τα δεδομένα υπάρχουν, τα επιστρέφει στο πρόγραμμα πελάτη, διαφορετικά επιστρέφει την τιμή null.

2. Store(x, Object): αποθηκεύει στην cache τα δεδομένα με το κλειδί x.

Ο χώρος μνήμης που καταλαμβάνει μια cache είναι συνήθως αρκετά μικρότερος σε σχέση με το χώρο που καταλαμβάνει η πηγή δεδομένων. Όταν ο χώρος της cache γεμίσει με εγγραφές, κάθε φορά που καλείται η λειτουργία Store, η cache πρέπει να διαγράψει μια εγγραφή προκειμένου να αποθηκεύσει τα νέα δεδομένα. Ο αλγόριθμος που χρησιμοποιείται για την επιλογή της εγγραφής που θα διαγραφεί λέγεται Πολιτική Αντικατάστασης ή Στρατηγική Αντικατάστασης (Replacement Policy ή Replacement Strategy), και παίζει πολύ σημαντικό ρόλο για την επίδοση της cache.
Η πιο διαδεδομένη πολιτική αντικατάστασης είναι η πολιτική Least Recently Used (LRU). Σύμφωνα με αυτή την πολιτική, όταν η cache χρειάζεται να διαγράψει μια εγγραφή, διαγράφει την εγγραφή στην οποία η τελευταία προσπέλαση είναι η λιγότερο πρόσφατη. Ενδεικτικά, μια υλοποίηση της πολιτικής LRU μπορεί να γίνει με τη χρήση timestamps: η cache διατηρεί, πέρα από τα δεδομένα, ένα timestamp για κάθε εγγραφή. Στο timestamp αυτό αποθηκεύει την χρονική

στιγμή που έγινε η τελευταία προσπέλαση της εγγραφής. Κάθε φορά που καλείται η λειτουργία LookUp, αν η εγγραφή υπάρχει στη μνήμη, τότε η cache ενημερώνει το αντίστοιχο timestamp με την τρέχουσα ώρα και μετά επιστρέφει τα δεδομένα στο πρόγραμμα πελάτη. Όταν καλείται η λειτουργία Store και η μνήμη της cache είναι γεμάτη, η cache επιλέγει την εγγραφή με το χρονικά παλαιότερο timestamp και την διαγράφει ώστε να δημιουργηθεί χώρος για τα νέα δεδομένα (σημειώστε ότι αυτός δεν είναι ο αλγοριθμικά βέλτιστος τρόπος να υλοποιηθεί μια LRU cache).

Υλοποίηση

Στόχος της εργασίας είναι να υλοποιήσετε μια cache γενικού σκοπού στην γλώσσα Java. Η cache θα εφαρμόζει την πολιτική αντικατάστασης LRU. Για την δομή σας, δίνεται ένα Java interface το οποίο περιγράφει όλες τις συναρτήσεις που θα υποστηρίζει η δομή σας (διαθέσιμο στο παράρτημα, αλλά και στο eclass). Το interface θα χρησιμοποιεί τα Java Generics ώστε η δομή να είναι γενικής χρήσης.

Η υλοποίησή σας θα πρέπει να είναι όσο το δυνατόν βέλτιστη ως προς τον χρόνο εκτέλεσης των λειτουργιών της cache, δίνοντας έμφαση στο χρόνο που απαιτεί η πράξη LookUp. Για την αποδοτικότερη λειτουργία, η υλοποίηση της cache απαιτεί συνδυασμό κάποιων δομών δεδομένων που καλύπτονται στο μάθημα. Για παράδειγμα, αφού δίνουμε έμφαση στο να γίνεται γρήγορα η αναζήτηση, αυτό πρέπει να σας παραπέμψει σε δομές που έχουν τέτοιες ιδιότητες (δέντρα διαφόρων ειδών, κατακερματισμός, και γενικότερα το υλικό που καλύπτεται στα κεφάλαια 12-14 του βιβλίου σας). Παράλληλα, τυχόν χρήση timestamps μπορεί να σας παραπέμψει στη χρήση ουράς προτεραιότητας/σωρού (κεφάλαιο 9). Διάφοροι συνδυασμοί από δομές είναι εφικτοί για την υλοποίηση (δεν είναι ομως όλοι το ίδιο αποδοτικοί). Οι εργασίες θα συγκριθούν μεταξύ τους ως προς το χρόνο εκτέλεσης και η καλύτερη εργασία θα έχει κάποια περαιτέρω επιβράβευση.

Η υλοποίησή σας θα πρέπει στον κατασκευαστή να παίρνει ως όρισμα το μέγιστο πλήθος των εγγραφών που θα αποθηκεύει η cache ώστε το πρόγραμμα πελάτης να καθορίζει το μέγεθος της. Για τη συγγραφή της δομής μπορείτε να επαναχρησιμοποιήσετε κώδικα που έχετε γράψει στη διάρκεια των εργασιών ή έτοιμο κώδικα από τα εργαστήρια. Σε καμία περίπτωση όμως **δεν μπορείτε να χρησιμοποιήσετε έτοιμες δομές δεδομένων είτε από την βιβλιοθήκη της Java είτε από έτοιμες βιβλιοθήκες.** Ο εκπαιδευτικός στόχος της εργασίας είναι να το υλοποιήσετε μόνοι σας.

Για την εκπόνηση της εργασίας συστήνεται να ψάξετε σε διάφορες πηγές (βιβλιογραφία, web) και να βρείτε ιδέες για διάφορους τρόπους υλοποίησης της cache (με πίνακες, με λίστες, με δέντρα, με διάφορους συνδυασμούς κλπ) ώστε να αποφασίσετε ποιος είναι ο αποδοτικότερος τρόπος και να τον υλοποιήσετε. Η αναζήτηση πληροφοριών και η επιλογή για το πώς θα υλοποιήσετε τις διάφορες μεθόδους είναι μέρος της εργασίας.

Δοκιμαστικά Δεδομένα

Για να επαληθεύσετε την ορθότητα της δομής, καθώς και τον υπολογιστικό χρόνο που χρειάζεται, έχει αναρτηθεί στο eclass έτοιμος κώδικας που προσομοιώνει το πρόγραμμα-πελάτη και την πηγή δεδομένων. Η default τιμή για το μέγεθος της cache στο πρόγραμμα πελάτη είναι 100, αλλά μπορείτε να χρησιμοποιήσετε κι άλλες τιμές.

Η πηγή δεδομένων αποτελείται από ένα απλό δομημένο αρχείο κειμένου με πολλές εγγραφές (από 100.000 έως 1.000.000) και ένα πρόγραμμα που διαβάζει το αρχείο αυτό. Το πρόγραμμα πελάτης παίρνει ως είσοδο ένα αρχείο που περιγράφει το φόρτο (workload): μια ακολουθία από κλειδιά για τα δεδομένα που πρέπει να αναζητηθούν. Το σύνολο των αναζητήσεων θα είναι πολλαπλάσιο του συνόλου των εγγραφών.

Προϋποθέσεις για συμμετοχή

1. Θα πρέπει να έχετε παραδώσει τις υπόλοιπες 3 εργασίες του μαθήματος. Ασχοληθείτε με αυτή την εργασία μόνο αν έχετε χρόνο και διάθεση, και δεν θα σας αποσπάσει από τις

υπόλοιπες υποχρεώσεις σας στο μάθημα και στην εξεταστική. Η εργασία έχει αξία 0.5 μονάδες. Δεν θα βαθμολογηθούν όμως εργασίες που δεν έχουν υλοποιήσει όλες τις μεθόδους του interface. Η υλοποίηση πρέπει να είναι **πλήρης** (έστω κι αν δεν είναι η πιο αποδοτική σε χρόνο).

2. Μπορείτε να είστε σε ζευγάρια ή μόνοι σας. Δεν χρειάζεται να είστε με τον ίδιο συνεργάτη που είστε στις υπόλοιπες εργασίες.

Παραδοτέα

Τα παραδοτέα της εργασίας είναι:

- Ο πηγαίος κώδικας (source code). Τοποθετήστε σε ένα φάκελο με όνομα **src** τα αρχεία java που έχετε φτιάξει. Φροντίστε να συμπεριλάβετε όποια άλλα αρχεία πηγαίου κώδικα απαιτούνται για να μεταγλωττίζεται η εργασία σας.

- Γράψτε μία αναφορά σε pdf αρχείο με όνομα project4-report.pdf, στην οποία θα περιγράφετε την λειτουργία της δομής σας. Στην αναφορά αυτή θα αναφέρετε
    a. πως λειτουργεί η δομή σας και σε ποιες δομές του μαθήματος βασιστήκατε.

    b. θα αναφέρετε ρητά το υπολογιστικό κόστος των πράξεων της cache, και θα δικαιολογήσετε πώς καταλήξατε να χρησιμοποιήσετε τη συγκεκριμένη υλοποίηση (την προτιμήσατε σε σχέση με κάποια άλλη και γιατί). Μπορείτε επίσης να συμπεριλάβετε διαγράμματα ή γραφικές παραστάσεις με την επίδοση του προγράμματος σας

    c. Αναφορές σε πηγές (βιβλία, επιστημονικά άρθρα, ιστοσελίδες) από όπου αντλήσατε πληροφορίες.

Πηγές:

[1] http://en.wikipedia.org/wiki/CPU_cache

[2] http://en.wikipedia.org/wiki/Page_cache

[3] http://en.wikipedia.org/wiki/Database_cache

[4] http://en.wikipedia.org/wiki/Web_cache

Όλα τα παραπάνω αρχεία θα πρέπει να μπουν σε ένα αρχείο zip. Το όνομα που θα δώσετε στο αρχείο αυτό θα είναι ο αριθμός μητρώου σας πχ. 3030056_3030066.zip ή 3030056.zip (αν δεν είστε σε ομάδα). Στη συνέχεια, θα υποβάλλετε το zip αρχείο σας στην περιοχή του μαθήματος «Εργασίες» στο e-class. Υποβάλετε ένα αρχείο ανα εργασία. Δεν χρειάζεται υποβολή και από τους 2 φοιτητές.

Παράρτημα

Διεπαφή Cache (διαθέσιμη και στο eclass)

```
/**
 * Defines the interface for a Cache data structure
 * The cache stores items of type V with keys of type K.
 * Keys must be unique
 * */
public interface Cache<K, V> {

	/**
	 * Look for data associated with key.
	 * @param key the key to look for
	 * @return The associated data or null if it is not found
	 */
	public V lookUp(K key);

	/**
	 * Stores data with associated with the given key. If required, it evicts a
	 * data record to make room for the new one
	 * @param key the key to associate with the data
	 * @param value the actual data
	 */
	public void store(K key, V value);

	/**
	 * Returns the hit ratio, i.e., the number of times a lookup was successful divided by the
number of lookup
	 * @return the cache hit ratio
	 */
	public double getHitRatio();

	/**
	 * Returns the absolute number of cache hits, i.e. the number of times a lookup found data
in the cache
	 */
	public long getHits();

	/**
	 * Returns the absolute number of cache misses, i.e. the number of times a lookup returned
null
	 */
	public long getMisses();

	/**
	 * Returns the total number of lookups performed by this cache
	 */
	public long getNumberOfLookUps();

}
```