ΑΝΑΦΟΡΑ 3ης ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Δομές δεδομένων και αρχείων

Μιχάλης Γαλάνης
2016030036

ΣΚΟΠΟΣ

Στη τρίτη και τελευταία εργαστηριακή άσκηση εξετάζουμε τη μέθοδο του **γραμμικού κατακερματισμού (linear hashing**) στο δίσκο. Συγκεκριμένα, μελετάμε την απόδοση **εισαγωγής** και **αναζήτησης στοιχείων** σε σελίδες δίσκου.

ΕΙΣΑΓΩΓΗ

Και η εισαγωγή αλλά και η αναζήτηση στοιχείων βασίζονται στη συνάρτηση **hash function,** η οποία δέχεται ως όρισμα ένα **κλειδί (key)**, και επιστρέφει ένα index στο οποίο θα εισαγχθεί (ή θα αναζητηθεί αντίστοιχα) το κλειδί. Για τις ανάγκες τις άσκησης η συνάρτηση αυτή υπολογίζει το υπόλοιπο της διαίρεσης του αριθμού με τον αριθμό των σελίδων **(mod based hash function)**.

Επειδή αναφερόμαστε σε σελίδες δίσκου, οι δύο παραπάνω βασικές πράξεις σύμφωνα με τις ανάγκες τις εκφώνησης έχουν ως εξής:

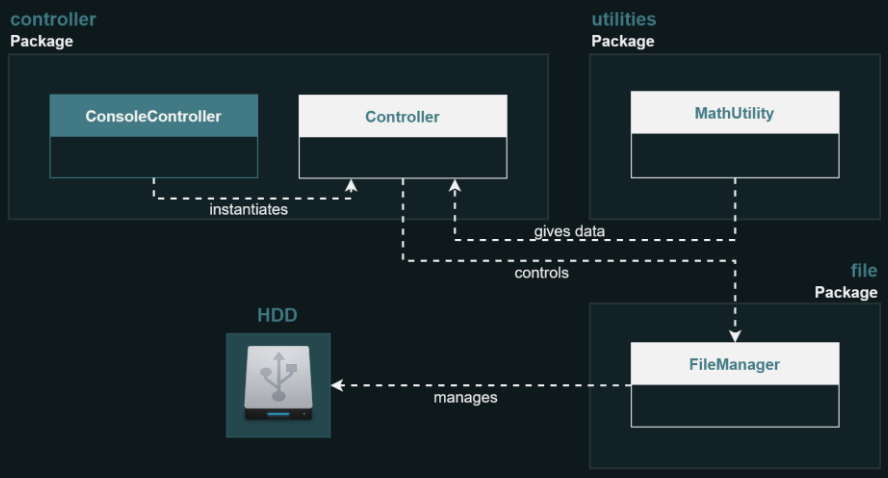
- **Αναζήτηση στοιχείου:** Η συνάρτηση αυτή δέχεται ως όρισμα ένα κλειδί, καλεί τη συνάρτηση hash function η οποία δείχνει σε ποια σελίδα πρέπει να αναζητηθεί το κλειδί. Ύστερα με μια απλή επανάληψη ελεγχουμε τα 128 στοιχεία της σελίδας αυτής για ταυτοποίηση με το κλειδί.
- **Εισαγωγή στοιχείου:** Παράγεται αρχικά ένα **κλειδί** ,στην περίπτωσή μας από μια γεννήτρια τυχαίων αριθμών. Εκτελείται η αναζήτηση στοιχείου με το συγκεκριμένο κλειδί (βλ. Πάνω). Αν το κλειδί δεν υπάρχει στη σελίδα αυτή, το εισάγουμε στην αμέσως επόμενη κενή θέση (της κύριας σελίδας αν υπάρχει διαθέσιμος χώρος, αλλιώς στη σελίδα υπερχείλησης).

Υπάρχει ένας συντελεστής διάσπασης (**u**) ο οποίος υπολογίζει το ποσοστό χώρου των συνολικών στοιχείων που έχουν συμπληρωθεί (συμπεριλαμβανομένου και των σελίδων υπερχείλησης). Για να αποφύγουμε **collisions**, όταν το **u** ξεπεράσει μια συγκεκριμένη επιθυμητή τιμή αποφασίζουμε να διασπάσουμε τις σελίδες.

Διάσπαση σελίδων συνεπάγεται το διαχωρισμό κλειδιών μεταξύ δυο σελίδων με βάση μια ανανεωμένη συνάρτηση **hash.** Αργότερα αναλύεται το πως εφαρμόστηκε αυτό στον κώδικα.

ΔΟΜΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Το πρόγραμμα αποτελείται απο 3 πακέτα και 4 κλάσεις οργανωμένα με τον τρόπο που φαίνεται σε επόμενο σχήμα και αναδεικνύεται η σχέση και ο ρόλος των κλάσεων/πακέτων:



Παρατηρήσεις: Η σημαντικότερη κλάση είναι η **Controller** καθώς εκεί εκτελούνται όλες οι λειτουργείες που σχετίζονται με το γραμμικό κατακερματισμό. Η **ConsoleController** περιλαμβάνει τη main() και ξεκινάει το πρόγραμμα καλώντας δυο objects του Controller (για τα δύο πειράματα). Η **MathUtility** είναι κλάση με μαθηματικά εργαλεία, κυρίως χρησιμοποιείται ως γεννήτρια τυχαίων αριθμών και η **FileManager** παρέχει λειτουργίες διαχείρισης αρχείων που κατασκευάστηκαν στη πρώτη εργαστηριακή άσκηση.

ΕΠΕΞΗΓΗΣΗ ΕΡΩΤΗΜΑΤΩΝ

Εκτέλεση πειραμάτων

Για την άσκηση χρειάστηκε να εκτελέσουμε 2 πανωμοιότυπα πειράματα με διαφορετικούς συντελεστές διάσπασης (**u > 80%** και **u > 50%** αντίστοιχα).

Η κλάση **Controller** διαθέτει μια μέθοδο για εισαγωγή στοιχείων `insertKeys()` η οποία λειτουργεί σύμφωνα με τον τρόπο που περιγράφηκε παραπάνω. Διαθέτει επίσης σύμφωνα με τα παραπάνω, και μία συνάρτηση `searchKey(int key, int page)` για αναζήτηση στοιχείων.

Η εκτύπωση των αποτελεσμάτων γίνεται με τη `displayStats()`.

Ο υπολογισμός του συντελεστή διάσπασης σε κάθε εισαγωγή γίνεται με τη `calculateOccupancyFactor()` και φαίνεται παρακάτω:

```
private double calculateOccupancyFactor(){
    double totalPrimaryNumbers = primaryFile.getNumberOfPages() * (primaryFile.getBUFFER_SIZE() - 1);
    double totalOverflowNumbers =  overflowFile.getNumberOfPages() * overflowFile.getBUFFER_SIZE();
    return storedValues / (totalPrimaryNumbers + totalOverflowNumbers);
}
```

Ενώ για τη συνάρτηση `hashFunction(int key)`, επειδή έχουμε αρχικά 10 κύριες σελίδες `(INITIAL_PAGES = 10)` και επεκτείνονται στις 40, η συνάρτηση αυτή για να χρησιμοποιείται για όλα τα κλειδιά ανα πάσα στιγμή χρειάστηκε ένας multiplier ωστε να υπολογιστεί το κατάλληλο υπόλοιπο για κάθε κλειδί κάθε φορά:

```
//Returns index of page the key has to be stored in.
private int hashFunction(int key){
    int multiplier = calculateMultiplier();
    return key % (((primaryFile.getNumberOfPages() > key % (multiplier * INITIAL_PAGES)) ? multiplier : multiplier / 2) * INITIAL_PAGES);
}

//Calculates multiplier needed for hash function.
private int calculateMultiplier(){
    int multiplier = 1;
    while (primaryFile.getNumberOfPages() > multiplier * INITIAL_PAGES){
        multiplier *= 2;
    }
    return multiplier;
}
```

Η διάσπαση σελίδων γίνεται με την `instantiateSplit()`. Κατα τη γνώμη μου η διάσπαση αποτέλεσε τη πιο δύσκολη λειτουργία της άσκησης καθώς έπρεπε να ξαναπεράσουμε τη κάθε τιμή (από τη σελίδα που γινόταν η επέκταση (κύρια και υπερχείλησης)) από τη **hash function** η οποία έκρινε αν τιμή αυτή καθεαυτή πρέπει να μετακινηθεί στη νέα σελίδα ή όχι. Μάλιστα, γίνεται πιο πολύπλοκη όταν για κάθε 10^n επέκταση έπρεπε όλες οι προηγούμενες –ανά 10 σελίδες (κύριες και υπερχείλησης)- να επανεξεταστούν για τη μετακίνηση κλειδιών στη τελευταία.

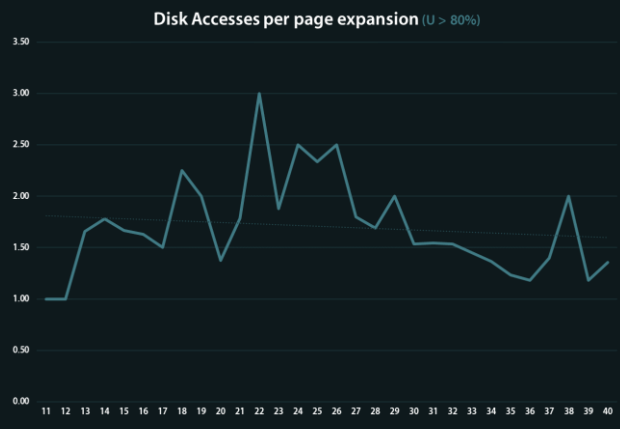
Τεκμηρίωση αποτελεσμάτων

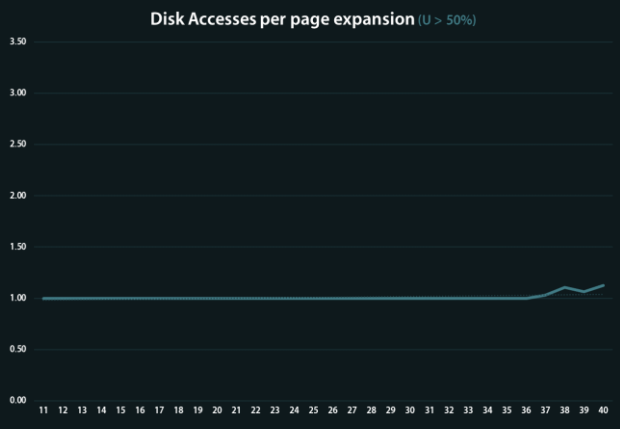
Παρακάτω φαίνονται τα αποτελέσματα στη κονσόλα για συντελεστή διάσπασης (**u > 80%** και **u > 50%** αντίστοιχα):

```
Linear Hashing Stats ( u > 80.0% ):
First Column represents the number of current expansion.
Second Column represents average number of disk accesses per page expansion.
- - - - - - -
11, 1.000000
12, 1.000000
13, 1.655809
14, 1.777778
15, 1.666667
16, 1.629032
17, 1.500000
18, 2.250000
19, 2.000000
20, 1.373860
21, 1.783784
22, 3.000000
23, 1.878049
24, 2.500000
25, 2.333333
26, 2.500000
27, 1.800000
28, 1.692308
29, 2.000000
30, 1.533333
31, 1.545455
32, 1.532786
33, 1.451613
34, 1.367347
35, 1.235294
36, 1.179487
37, 1.400000
38, 2.000000
39, 1.183062
40, 1.355263
- - - - - - -
```

```
Linear Hashing Stats ( u > 50.0% ):
First Column represents the number of current expansion.
Second Column represents average number of disk accesses per page expansion.
- - - - - - -
11, 1.000000
12, 1.000000
13, 1.000000
14, 1.000000
15, 1.000000
16, 1.000000
17, 1.000000
18, 1.000000
19, 1.000000
20, 1.000000
21, 1.000000
22, 1.000000
23, 1.000000
24, 1.000000
25, 1.000000
26, 1.000000
27, 1.000000
28, 1.000000
29, 1.000000
30, 1.000000
31, 1.000000
32, 1.000000
33, 1.000000
34, 1.000000
35, 1.000000
36, 1.000000
37, 1.031250
38, 1.104712
39, 1.062500
40, 1.126984
- - - - - - -
```

Αν προσπαθήσουμε να απεικονίσουμε τις παραπάνω τιμές, παράγονται τα παρακάτω διαγράμματα:





Παρατηρούμε ότι στη περίπτωση του **(u > 50%)** ο μέσος όρος προσβάσεων στο δίσκο για κάθε επέκταση σελίδας σε σχέση με **(u > 80%)** είναι γενικά μικρότερος και αυτό συμβαίνει διότι για **u > 50%** γίνεται πιο συχνά διάσπαση σελίδων, οπότε οι περισσότερες τιμές που εισάγονται, εισάγονται σε νέες κύριες σελίδες και όχι σε σελίδες υπερχείλησης που απαιτούν περισσότερες προσβάσεις στο δίσκο. Μάλιστα το 50% είναι αρκετά μικρό ποσοστό ώστε σχεδόν ποτέ να μη χρειαστεί να εισαχθούν κλειδιά σε σελίδες υπερχείλησης.

Σύμφωνα με τα παραπάνω, συμπεραίνουμε ότι όσο μικρότερο είναι το ποσοστό που πρέπει να ξεπεράσει ο συντελεστής διάσπασης τόσο πιο γρήγορος είναι ο αλγόριθμος αφού απαιτεί λιγότερες προσβάσεις δίσκου. Μειονέκτημα αυτού είναι όμως οτι για ένα συγκεκριμένο αριθμό σελίδων, μειώνεται ο αριθμός εισαγωγών άρα αυξάνεται ο χώρος που καταναλώνουμε. (Σύμφωνα με την κονσόλα, εισαγωγές για u > 80%: **5264** και για u > 50%: **2669)**