Οικονομικό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής
Μάθημα: Οργάνωση Συστημάτων Υπολογιστών

Τετάρτη, 11 Δεκεμβρίου 2019

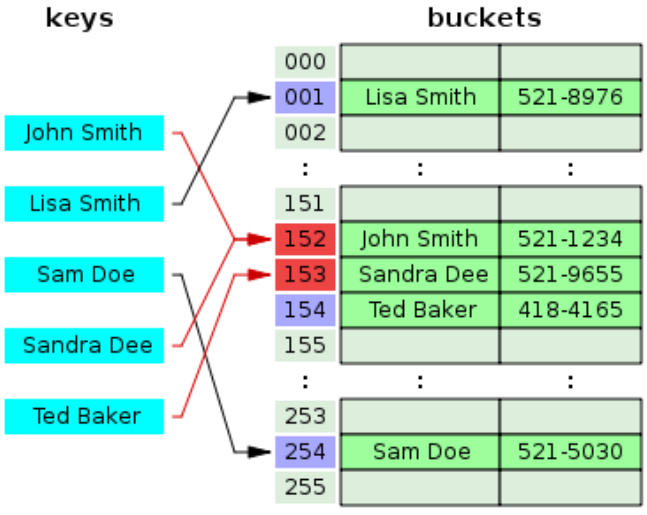
Εργασία 3η

Τίτλος: Υλοποίηση Πίνακα Κατακερματισμού (Hash Table) με χρήση Συμβολικής Γλώσσας Επεξεργαστή MIPS32

Έστω ότι θέλουμε να αποθηκεύσουμε 100 αριθμητικές τιμές σε έναν πίνακα. Ένας τρόπος θα ήταν η αποθήκευση των τιμών – κλειδιών να γίνει τυχαία (ή σειριακά). Στην περίπτωση αυτή όλες οι διαδικασίες (εισαγωγή, αναζήτηση, κλπ) απαιτούν χρόνο ανάλογο με το πλήθος των αποθηκευμένων δεδομένων. Ένας πιο αποδοτικός τρόπος αποθήκευσης των κλειδιών είναι η χρήση μιας συνάρτησης που θα μετασχηματίζει κάθε κλειδί σε μια θέση του πίνακα. Μια τέτοια συνάρτηση είναι η εξής:

Θέση πίνακα = αρχικό κλειδί % μέγεθος πίνακα

Έτσι, το κλειδί 253 θα αποθηκευτεί στη θέση 253%100 = 53. Με τον ίδιο τρόπο μπορεί να βρεθεί η θέση του πίνακα για οποιοδήποτε κλειδί. Μια τέτοια συνάρτηση που μετασχηματίζει ένα μεγάλο διάστημα τιμών σε ένα μικρότερο διάστημα ονομάζεται **συνάρτηση κατακερματισμού (hashing function)** και ένας πίνακας στον οποίο εισάγονται δεδομένα με μια τέτοια συνάρτηση μετασχηματισμού του κλειδιού ονομάζεται **πίνακας κατακερματισμού (hash table)**. Ο στόχος των πινάκων κατακερματισμού είναι να επιτευχθεί σταθερός χρόνος προσπέλασης, ανεξάρτητα από το πλήθος των δεδομένων. Στην περίπτωση που τα κλειδιά δεν είναι αριθμητικά, χρησιμοποιούμε τις ASCII τιμές των χαρακτήρων τους –τις οποίες μετά προσθέτουμε – για να τα μετατρέψουμε σε αριθμητικά:



Hash collision (open addressing with linear probing)[1]

Το πρόβλημα βέβαια είναι ότι σύμφωνα με τον προηγούμενο μετασχηματισμό δύο διαφορετικά κλειδιά μπορεί να οδηγηθούν στην ίδια θέση του πίνακα. Για παράδειγμα, τα κλειδιά 253 και 453 προκύπτει ότι θα αποθηκευτούν στη θέση 53. Αυτή η κατάσταση ονομάζεται **σύγκρουση** (collision). Τα δύο κλειδιά που διεκδικούν την ίδια θέση στον πίνακα ονομάζονται **συνώνυμα** (synonyms). Μία λύση στο πρόβλημα αυτό, είναι να εξετάσουμε τον πίνακα, να βρούμε μία άδεια θέση και να εισάγουμε εκεί το νέο κλειδί. Αυτή η

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

τεχνική ονομάζεται **ανοιχτή διευθυνσιοδότηση** (open addressing). Η πιο απλή μέθοδος ανοιχτής διευθυνσιοδότησης είναι η γραμμική εξέταση (linear probing). Η μέθοδος ψάχνει σειριακά στον πίνακα, ξεκινώντας από την επόμενη θέση από όπου συνέβη η σύγκρουση, μέχρι να βρει μια άδεια θέση, όπου και αποθηκεύει το νέο στοιχείο.

Ζητείται

Να γραφεί, στη συμβολική γλώσσα του επεξεργαστή MIPS32, ένα πρόγραμμα που υλοποιεί έναν πίνακα κατακερματισμού 10 θέσεων και τις μεθόδους της εισαγωγής και αναζήτησης αριθμητικών κλειδιών σε αυτόν. Παρακάτω δίνεται η περιγραφή του αλγορίθμου που επιλύει το παραπάνω πρόβλημα σε γλώσσα Java:

```
import acm.program.*;

/**
 * @author M. Togantzi
 * @author C. Kalergis
 */


public class hash extends Program {

    final int N = 10;    // καθολικές
    int keys = 0;        // μεταβλητές

    public void run ()  {                        // κύριο πρόγραμμα

        int key, pos, choice, telos = 0;         // μεταβλητές προγράμματος
        int [] hash = new int [N];               // πίνακας κατακερματισμού
        for (int i=0; i<N; i++) hash[i] = 0;     // αρχικοποίηση πίνακα
        do  {
            println(" Menu");
            println("1.Insert Key");
            println("2.Find Key");
            println("3.Display Hash Table");
            println("4.Exit");
            choice = readInt ("\nChoice?");
            if (choice == 1) {
                    key = readInt ("Give new key (greater than zero): ");
                    if (key>0) insertkey(hash, key);
                    else println ("key must be greater than zero");
            }
            if (choice == 2) {
                    key = readInt ("Give key to search for: ");
                    pos = findkey(hash, key);
                    if (pos == -1)
                        println("Key not in hash table.");
                    else {
                        println("Key value = " + hash[pos]);
                        println("Table position = " + pos);
                    }
                }
            if (choice == 3) displaytable(hash);
            if (choice == 4) telos = 1;
        } while (telos == 0);
    }
```

```
    void insertkey(int[] hash, int k) {     // τυπικές παράμετροι (διεύθυνση πίνακα και κλειδί)

        // Εισαγάγει ένα κλειδί στον πίνακα κατακερματισμού εφόσον το κλειδί δεν υπάρχει ήδη στον
        // πίνακα και υπάρχει ελεύθερη θέση. Καλεί τη findkey που ελέγχει αν το κλειδί βρίσκεται ήδη
        // στον πίνακα και τη hashfunction που υπολογίζει τη θέση εισαγωγής του κλειδιού στον πίνακα.

        int position;                       // τοπική μεταβλητή

        position = findkey(hash, k);
        if (position != -1) println("Key is already in hash table.");
        else
            if (keys<N){
                position = hashfunction(hash, k);
                hash[position] = k;
                keys++;
            }
            else println ("hash table is full");
    }

    int hashfunction (int[] hash, int k) { // τυπικές παράμετροι (διεύθυνση πίνακα και κλειδί)

        // Υπολογίζει τη θέση εισαγωγής του κλειδιού στον πίνακα και την επιστρέφει

        int position;                       // τοπική μεταβλητή

        position = k % N;
        while (hash[position] != 0) {
            position++;
            position %= N;
        }
        return position;
    }

    int findkey(int [] hash, int k) {   // τυπικές παράμετροι (διεύθυνση πίνακα και κλειδί)

        // Αναζητεί ένα κλειδί στον πίνακα κατακερματισμού. Αν το βρει επιστρέφει τη θέση του
        // διαφορετικά επιστρέφει -1

        int position, i = 0, found = 0; // τοπικές μεταβλητές
        position = k % N;
        while (i < N && found == 0) {
            i++;
            if (hash [position] == k)
                found = 1;
            else {
                position++;
                position %= N;
            }
        }
        if (found == 1)
            return position;
        else
            return -1;
    }

    void displaytable(int [] hash) {    // τυπική παράμετρος (διεύθυνση πίνακα)

        // Εμφανίζει τον πίνακα κατακερματισμού

        int i;                          // τοπική μεταβλητή
        println("\npos key\n");
        for (i=0; i<N; i++)
            println(" " + i + " " + hash[i]);
    }
}
```

Παρατηρήσεις

- Η λύση σας θα πρέπει να υλοποιεί ρητά το παραπάνω πρόγραμμα (java).
- Η εργασία μπορεί να υλοποιηθεί από μικρές ομάδες φοιτητών (δύο φοιτητές – όχι πάνω από δύο). Εναλλακτικά, σε περίπτωση μη εύρεσης συναδέλφου, μπορείτε να υλοποιήσετε την εργασία ατομικά.
- Η αξιολόγηση της παρούσας εργασίας ενδέχεται να γίνει σε συνδυασμό με προφορική εξέταση της κάθε ομάδας.

Οδηγίες

- Ονομάστε το αρχείο που περιέχει το πρόγραμμα, με τον αριθμό του φοιτητικού σας μητρώου και κατάληξη **.txt** (για παράδειγμα 3100000-3101111.txt).
- Εκτελέστε το πρόγραμμά σας στον προσομοιωτή **SPIM** και βεβαιωθείτε ότι δεν έχει συντακτικά ή άλλα σφάλματα.
- Τεκμηριώστε το πρόγραμμά σας με τα κατάλληλα σχόλια, έτσι ώστε να είναι κατανοητό. Ως σχόλια μπορείτε να χρησιμοποιείστε και εντολές από τον κώδικα java, όπως δίνεται παραπάνω. Στην πρώτη γραμμή του προγράμματος γράψτε, ως σχόλιο, τα προσωπικά σας στοιχεία (Επώνυμο, Όνομα, Αριθμό Μητρώου).
- Να υποβάλετε το αρχείο που περιέχει το **πρόγραμμα,** στο eclass μέχρι την **Τετάρτη, 8/1/2020**. Εκπρόθεσμες εργασίες, με μικρή υπέρβαση του ορίου υποβολής, θα γίνουν δεκτές από το eclass, αλλά θα έχουν αντίστοιχη μείωση του βαθμού.
- Στην περίπτωση της ομαδικής υλοποίησης, η εργασία υποβάλλεται από το ένα μόνο μέλος της ομάδας (μια φορά) αλλά στο όνομα του αρχείου και στα σχόλια αναφέρονται τα στοιχεία όλων των μελών της ομάδας.