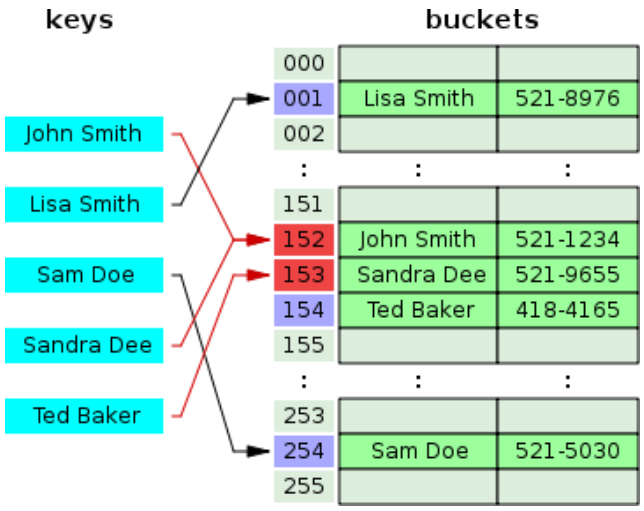
Athens University of Economics and Business
Department of Informatics
Class: Computer Systems Organization
Instructor: Spiros Voulgaris
Assistants: Christos Kalergis, Togantzi Maria
Academic Year: 2019-2020

Implementation of Hash Table using MIPS32

Let's assume we want to store 100 numerical values in an array. One way to do this is by storing the values-keys randomly or linearly. In this case every procedure (import, search.) has O(n) time complexity (where n is the number of numerical values). A more efficient way to store the keys is using a function that transforms every key in an array position. For example:
Array position = original key % array size
This way, key 253 will be stored in position 253%100=53. Following the same way, we can find the position of any key. A function like this one that transforms a large range of values is called hashing function and the array that is used to store data using a hashing function is called hash table. The goal of hash tables is to achieve an O(1) access time . If the given key in not of numeric type, we use the ASCII values of its characters -which we then add- to transform it to numeric.



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

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

The problem is that according to the last transformation, two different keys can go to the same array position. For example, keys 253 and 453 will be stored in position 53 .This is called a collision. The two keys that claim the same position are called synonyms. A solution for this problem is to search the array, find an open position and store our new key there. This technique is called open addressing. The simplest function to implement open addressing is linear probing. This function searches the array linearly starting from the collision position until it finds an empty position to store our key.

Using MIPS32 assembly language, write a program that implements an array that can store up to 10 keys and functions for importing and searching numeric keys in the array.

The program that is pictured below, solves this problem in Java.

```
public class Test{

    static final int N = 10;
    static int keys = 0;

    public static void main (String[] args){  //main function
        int key,pos,choice,telos = 0;         //initial variables
        Scanner sc= new Scanner(System.in);
        int [] hash = new int [N];            //hash table
        for (int i =0; i<N; i++){
            hash[i] = 0;
        }
        do{
            System.out.println("\nMenu");
            System.out.println("1. Insert Key");
            System.out.println("2. Find Key");
            System.out.println("3. Display Hash Table");
            System.out.println("4. Exit");
            System.out.println("\nChoice?");
            choice = sc.nextInt();
            if (choice == 1) {
                System.out.println("Give new key (greater than zero): ");
                key = sc.nextInt();
                if (key>0){
                    insertkey(hash,key);
                }
                else{
                    System.out.println("key must be greater than zero");
                }

            }
            if (choice == 2) {
                System.out.println("Give key to search for: ");
                key = sc.nextInt();
                pos = findkey(hash,key);
                if (pos == -1){
                    System.out.println("Key not in hash table.");

                }
                else{
                    System.out.println("Key value = " +hash[pos]);
                    System.out.println("Table position = " +pos);

                }
            }
            if (choice == 3 ){
                displaytable(hash);
            }
            if (choice == 4){
                telos = 1;
            }

        } while (telos == 0);
    }
```

```
static void insertkey(int[] hash, int k) {  //inserts a key if it is not present in the hash table
    int position;                           //and there is free space
    position = findkey(hash,k);
    if (position != -1) {
        System.out.println("Key is already in hash table.");
    }
    else{
        if (keys<N){
            position = hashfunction(hash,k);
            hash[position] = k;
            keys++;
        }
        else{
            System.out.println("hash table is full");
        }
    }
}

static int hashfunction(int[] hash, int k) {  //calculates the position of the hash table, the
    int position;                             //the specific key must be stored
    position = k%N;
    while (hash[position] != 0) {
        position++;
        position %= N;
    }
    return position;
}

static int findkey(int[] hash, int k){  //searhes for a key in the hash table
    int position, i = 0, found = 0;     //returns the position of the key if key is found, else -1
    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;
    }
}

static void displaytable(int[] hash) { //displays the current hash table
    int i;
    System.out.println("\npos key\n");
    for (i=0; i<N; i++){
        System.out.println(" " +i + " " + hash[i]);
    }
}
```

Comments

- Your solution should implement exactly this Java program.

Instructions

- Run your program on QTSPIM or MARS.