King Fahd University of Petroleum & Minerals
College of Computer Science and Engineering
Information and Computer Science Department
ICS 202 – Data Structures

Hashing

Objectives

The objective of this lab is to study hashing.

Outcomes

After completing this Lab, students are expected to:

- Be familiar with hash tables
- Apply hash functions

Lab Exercises

The idea of this lab is to implement Hashtables. You are expected to comeup with two classes:

1. class `Entry` – this class represents an entry in a hashtable. It has three components:
   a. an Object (called `dataObject`) that holds the data.
   b. a String (called `status`) that can have one of these values: “E” (empty), “D” (deleted) or “O” (occupied).
   c. A `toString` method that prints the contents of the `dataObject`
   d. A hash method that gets the hashCode of the object (This is from the built-in java hashCode).

2. class `Hashtable` – this class represents the hashtable. Its main data structure is an array of `Entry` objects. (of size ‘`n`’ which you may specify in the constructor). Your class should support the following methods:
   a. Insert an object into the hashtable of entries.
   One signature could be `public void insert(Object o)`
   Initially you can assume that the data object to be inserted is not in the hashtable. Later on, you can improve your code by checking for duplicate objects (Do not allow duplicate objects to be inserted). You may also create a method `public int findNextVacantSlot(Object o)`

   b. Retrieve (or find) an object
   c. Delete an object
   d. toString() – print the contents of the hashtable.

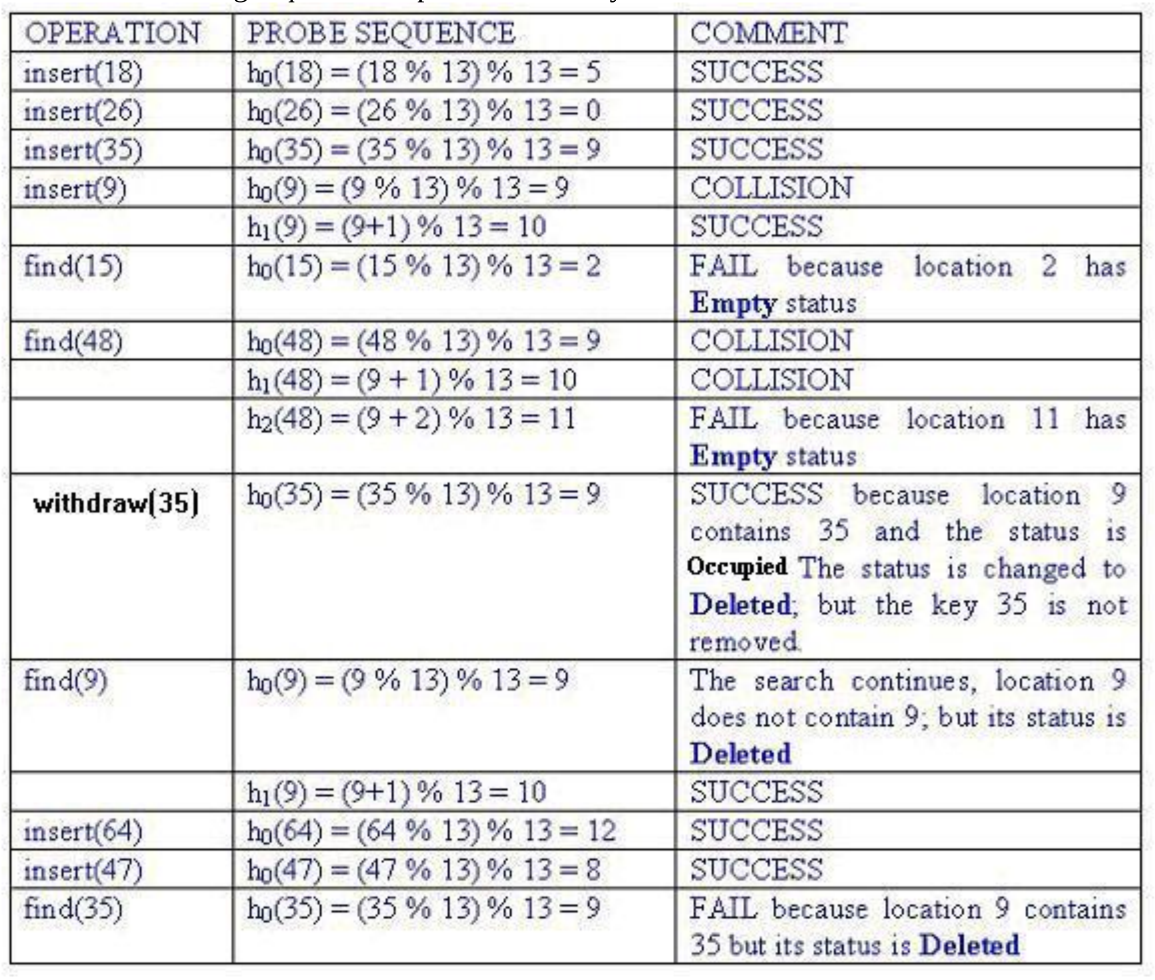
   Use a simple remainder hash function.
   Make sure you hash the **dataObject’s hashcode** before insertion, deletion and retrieval.
   Furthermore use linear probing for the probe sequence.

3. Write a main method to test your program with this example (given in the picture).

.

4. Testing

Use the following sequence of operations to test your hashtable.



OPERATION	PROBE SEQUENCE	COMMENT
insert(18)	$h_0(18) = (18 \% 13) \% 13 = 5$	SUCCESS
insert(26)	$h_0(26) = (26 \% 13) \% 13 = 0$	SUCCESS
insert(35)	$h_0(35) = (35 \% 13) \% 13 = 9$	SUCCESS
insert(9)	$h_0(9) = (9 \% 13) \% 13 = 9$	COLLISION
	$h_1(9) = (9+1) \% 13 = 10$	SUCCESS
find(15)	$h_0(15) = (15 \% 13) \% 13 = 2$	FAIL because location 2 has **Empty** status
find(48)	$h_0(48) = (48 \% 13) \% 13 = 9$	COLLISION
	$h_1(48) = (9 + 1) \% 13 = 10$	COLLISION
	$h_2(48) = (9 + 2) \% 13 = 11$	FAIL because location 11 has **Empty** status
withdraw(35)	$h_0(35) = (35 \% 13) \% 13 = 9$	SUCCESS because location 9 contains 35 and the status is **Occupied** The status is changed to **Deleted**; but the key 35 is not removed.
find(9)	$h_0(9) = (9 \% 13) \% 13 = 9$	The search continues, location 9 does not contain 9; but its status is **Deleted**
	$h_1(9) = (9+1) \% 13 = 10$	SUCCESS
insert(64)	$h_0(64) = (64 \% 13) \% 13 = 12$	SUCCESS
insert(47)	$h_0(47) = (47 \% 13) \% 13 = 8$	SUCCESS
find(35)	$h_0(35) = (35 \% 13) \% 13 = 9$	FAIL because location 9 contains 35 but its status is **Deleted**

5. More testing:
   Test your hashtable for the following operations (assume the size is 13)
   Insert(1), Insert(2), Insert(3), Insert(4), Insert(17), Insert(30), Insert(5), Insert(6), Insert(7), Insert(8), Insert(9), Insert(22), Insert(35), Delete(4), Insert(4)