Lab 5: Hash Tables

Week of March 16, 2020

Objective

To complete a program that compares two different hash table implementations of ADT Table.

The Nearly-Complete Program

File `Lab05.java` contains a nearly-complete application that uses two Table implementations: the first is a hash table using linear probing and the second is a hash table using separate chaining. It inserts all the items in input file `Lab05Input.txt` into both tables and then prints some statistics to allow a comparison. (Input file `Lab05Input.txt` is available on UM Learn with `Lab05.java`.)

In this application, an item consists of a `String`. The `String` is the key — there is no auxiliary data associated with the key in this application. The input file contains one item per line.

The file `Lab05.java` contains four classes:

1. The application class (named `Lab05`), which contains `main()` and the method it calls.
2. The `TableWithLP` class, which implements ADT Table with a hash table that uses linear probing for collision resolution.
3. The `TableWithSC` class, which implements ADT Table with a hash table that uses separate chaining for collision resolution. In this implementation, each table slot is a pointer to a `Node` which is a pointer to the first `Node` in a linked list of keys that hash to this position, — that is, each table slot is an unordered linked list with no dummy nodes.
4. The `private Node` class inside the `TableWithSC` class. This class implements an ordinary linked-list node with `public` instance members `item` and `next`.

You will complete two methods in the `TableWithSC` class.

Exercise

Complete the program in the file `Lab05.java` by adding TWO method bodies (details below). Do not change any of the code that is already written in `Lab05.java` — just add the two required method bodies. You can write helper methods for these two methods, if needed.

The two methods you will write the bodies of:

`insert`**:** There are two inserts, the first one is already implemented. You will implement the second insert. This method is passed a key (a `String`). It is supposed to insert key into the hash table, using separate chaining to resolve any collisions.

If the key is already in the table, then the method should print an error message and not insert the key — duplicates are not allowed.

If the key is not in the table, then the method should insert the key and increment instance member `numberItems`, which contains the total number of items currently stored in the table.

`search`**:** This method is passed a key (a `String`). It does a proper hash table search for the key, returning `true` if it finds the key and `false` if it doesn't.

Of course, it assumes that the hash table is using separate chaining to resolve collisions.