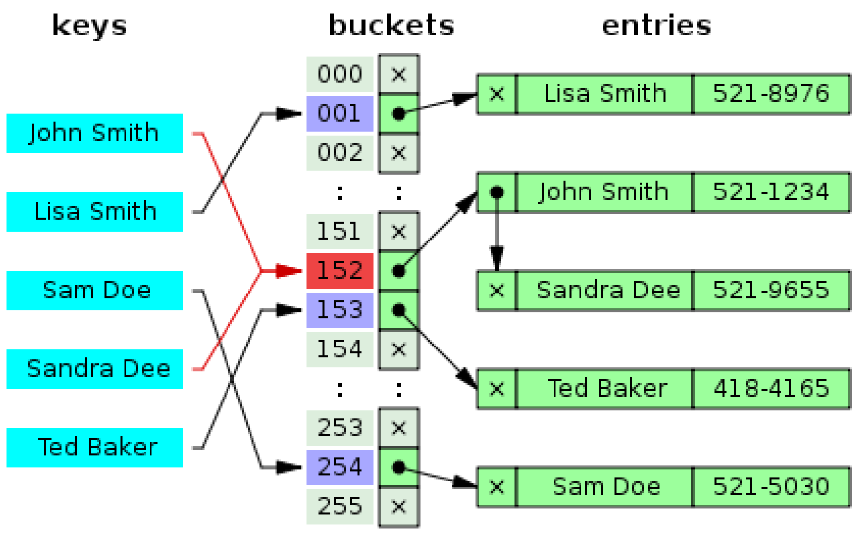
Comp 202 - Spring 2022
Homework #2
Due date - 23:59 April 06, 2022

Homework Description

In this assignment, you will be implementing a **Hash Table** in Java from scratch. You might have noticed the standard **HashTable** implementation in Java is a generic class that works with many types. You do not, however, need to make your code generic; for the purposes of this assignment, you will use **String** as your key and **Object** as the value.



As a brief reminder, hash tables use a hashing function to compute numeric indices from keys and use them to index into their *internal buckets*, allowing fast lookup. However, since collisions can occur in the hash algorithm, the buckets do not point directly to the values; they instead point to linked list data structures that hold key-value pairs of the same key.

Define a class CrashTable and a private inner class EntryNode. The latter will be your singly linked list:

Define the following attributes in EntryNode:

- String key
- Object value
- EntryNode next

Define EntryNode[] table as an instance variable. This will be an array of your buckets.
Define a MAX_CONFLICTS=10 constant (explained below)

Computing the hash

The hashing algorithm you will be implementing in this assignment is Polynomial rolling hashing with the prime number **89**. You need to modulo the resulting hash with the size of the table array in order to avoid going outside of the array boundary. Notice how the hash values are dependent on the size of the table.

Your Hash Table implementation will be slightly different than how it would normally be implemented. We want our table to use very little memory, so our table array will initially be set to **8** elements. This will cause many conflicts as you can imagine, but we allow a certain number of conflicts (MAX_CONFLICTS) for each bucket. When any bucket reaches the limit, you need to double the size of the table, and rehash every key with the new size.

Implement the following methods in CrashTable:

private static int hash(String key) [10%]
This method will be used by the instances to compute numerical hashes of given keys.

public Object get(String key) [20%]
Do the following steps to implement the method. Note that some steps are repeated for other methods, and will not be repeated:

1. Compute the hash of the given key
2. Look up the index in the bucket. Return null if no such index exists, or the bucket is empty.
3. If the bucket contains an EntryNode at the given index, traverse through the linked list to find a node whose key is equal to the given key.
4. If found, return it, otherwise, return null

public Object get(String key, Object def) [5%]
Same as the get method above, but returns the supplied default value if the key is not found.

public Object put(String key, Object value) [20%]
This method inserts a new key-value pair to the table. Similar to the get method, compute the hash of the key and insert the appropriate objects. Note that you may need to resize the table array (and rehash!) if the bucket will exceed MAX_CONFLICTS after the insertion.
If a value already exists at the given key, return it before overwriting it, otherwise return null

public Object remove(String key) [10%]
This method will remove the key-value pair from the table associated with the given key. You are not required to resize the table after this operation.

public String[] getKeys() [10%]
This method will return a String array of every key in the table.

public void resize(int size) [15%]
Resizes the table to a given size. Do not forget to rehash!

Submission Details

1. Accept the invitation at this link: https://classroom.github.com/a/HQtDfMyh
2. Choose your students ID from the list to link to your GitHub account if you haven't already. If you have no GitHub account, create one.
3. A personal repository with the starter code will be created. Ensure that your repository is private.
4. Make sure to commit and push all your changes to GitHub.
5. To avoid possible issues with GitHub code submission, submit a copy of your project to Blackboard as well. Submit a single .zip file named ID_NAME_SURNAME.zip.

Grading Criteria

Point allocation for each method is listed alongside its name. 10 additional points are reserved for overall correct implementation of classes.

Template Code

Template code with placeholders for required methods has been provided to you on GitHub Classroom. You are free to modify the project structure as you see fit as long as the method signatures are left intact.

You are also provided a `table.txt` file that contains String-String key-value pairs. Code for reading and parsing the file into a List in `[ [ “Key”, “Value” ] ]` format is provided in Main.java. You may use this list to populate your Hash Tables for testing.

A GitHub autograder is included in the assignment that compiles and runs your code. This is only a cursory test, however, and does not check the correctness of your implementation. The test will pass if your code is free of compilation errors and can execute without runtime errors.

The template code was tested in Java 11 and 17.

Code of Conduct

Same rules that are mentioned in the course syllabus also apply to this assignment. Please refer to that course syllabus for the code of conduct. Good luck!