Homework 7

CS 2461

Overview

This homework will prepare you for Project 5. You will implement a HashMap (which you will be able to reuse in Project 5) that handles collisions using chaining. A HashMap is a data structure that stores *"key,value"* pairs with fast insertion and retrieval times. For our implementation, our keys will be a combination of two `char*`, the word itself and the document id (name) the word appears in. Values will simply be an `int` representing the number of times that word appears in the document. The application of our HashMap will be to store the number of times a word appears across multiple files. It might still be useful to review in more detail how HashMaps do chaining before beginning this assignment.

Requirements

`Struct` definitions have been provided for you in **hashmap.h**, as well as function declarations. **Do not modify the hashmap.h file or the Makefile.** You you are required to do the following:

1. Fill in the functions in **hashmap.c**.
2. In **test.c**, fill in the main method with code which reads from the 3 given text files (**D1.txt**, **D2.txt**, and **D3.txt**) and counts the number of times each word appears in each of the three files. It is safe to assume for this homework that the words in the files will all be lowercase.

The functions you will need to implement are as follows (defined in **hashmap.h**):

- `struct hashmap* hm_create(int num_buckets);`
- `int hm_get(struct hashmap* hm, char* word, char* document_id);`
- `void hm_put(struct hashmap* hm, char* word, char* document_id, int value);`
- `void hm_remove(struct hashmap* hm, char* word, char* document_id);`
- `void hm_destroy(struct hashmap* hm);`
- `int hash(struct hashmap* hm, char* word, char* document_id);`

`hm_create` will allocate (hint hint) a new HashMap with the specified number of buckets.

`hm_get` will return the value associated with the key that is passed in within the HashMap that is passed in. If the element is not found, return `-1`.

`hm_put` will put the key value pair into the HashMap that is passed in. If the word and document id combination already exists within the HashMap, its value will be overwritten with the new one.

`hm_remove` will remove the key value pair in the HashMap that is associated with the given key.

`hm_destroy` will deallocate (free) the HashMap that is passed in and all of its elements.

`hash` will take the given word and document id and map them to a bucket in the HashMap. An easy way to do this is to sum the ASCII codes of all of the characters then modulo by the number of buckets.

Example Execution

Assume the file contents look like the following:

D1: `i love computer architecture`

D2: `computer architecture rocks`

D3: `this homework is a great homework assignment`

For each word in each file, you would do the following:

1. Check if the word / document id combination is already in the HashMap.
2. If it is not, put it in with a count of 1. If it is, get its current count from the HashMap, increment it, then put it back in, overwriting the existing value.

Once you have read through all the words in all the files, test you implementation works correctly by making calls to `hm_get`. For example, a call to `hm_get(hm, "homework", "D3")` should return `2` because the word "homework" appears twice in D3. Finally, you should deallocate your HashMap by calling `hm_destroy`.

Compiling and Running

To compile and test your code, run the following commands in the directory containing your code files. To ensure that your code compiles correctly, run `make`. To run your tests in **test.c**, run `./test` after you have run `make`. To remove a previous build, run `make clean`. **Do not modify** the Makefile. Your code must compile with the Makefile that is provided. **Your code will be tested on the SEAS shell, please ensure it compiles and runs there before submitting.**

Question

Is it more or less efficient to use a hash map or linked list when trying to search for an element? Explain and give an example for each scenario.