CSCI 2461 - Homework 6: Hashmap

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 key will be the word (char*). Values will be document id (name of document) and an integer 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 in each document, across multiple documents (files). It might still be useful to review in more detail how HashMaps do chaining before beginning this assignment.

Submission only on github, and you must document your code.

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 are required to do the following:

- Fill in the functions in **hashmap.c**
- In **test.c**,
  1. Fill in the main
  2. Fill in the functions in hashmap.c. (Your compiler may throw out warnings or errors if you compile the posted code (you will need to insert return 0 in the functions).
  3. 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)** , creates the hashmap and counts the number of times each word appears in each of the three files (*i.e.,* you will need to call various functions you implement in hashmap.c). It is safe to assume for this homework that the words in the files will all be lowercase.
  4. (Optional but highly recommended) Design and fill out the `test.c` test cases to test your implementation – check the correctness of your various functions (for example, hm_get when you try to get a word that does not exist should return -1).

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 num_occurences);`
- `void hm_remove(struct hashmap* hm, char* word, char* document_id);`
- `void hm_destroy(struct hashmap* hm);`

- `int hash(struct hashmap* hm, char* word);`

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

`hm_get` will return the number of times the word appears in the document . If the word is not found in the document then it returns -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 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 in the word then compute modulo the number of buckets to determine the bucket in the hashmap, *i.e.,* `sum % buckets` operation in C.

Example Execution

Assume the three file contents look like the following:

```
D1.txt: i love computer architecture
D2.txt: computer architecture rocks
D3.txt: 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, then insert it into the hashmap with a count of 1 by calling `hm_put`. If it is, get its current count from the HashMap, increment it, then put it back in, overwriting the existing count value.

A subset of the hashmap created after reading the three input files is shown in the figure below (the example in the figure does not use the actual hash function you may be using in your implementation):

Once you have read through all the words in all the files, test if your 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. A call to `hm_get(hm, "homework", D2)` should return -1 since the word is not in the document D2. 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

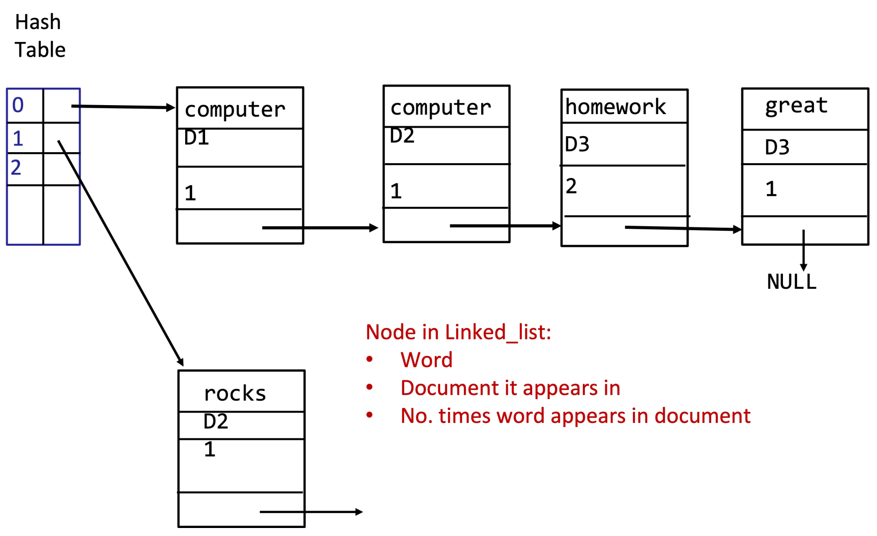

Figure 1: Example hashmap

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.