2020/10/07 - Hashtables

06 Tháng Mười 2020 11:08 CH

SYNOPSIS

- Go over Lab 4.

LAB 4

- On Canvas, go to the "Lab 4" assignment and read the "lab4.html" file attached. All lab details will be there.

- There is template code, but it isn't located in the cosc140 folder for the lab. Instead, go to Canvas and view the "hashtable2_handout". Copy the second page (hashtable_gp.cpp), as well as the hash_table<string>::hash function on the first page.
  - More on this below...

SUBMISSION COMMAND

- tar -cvf lab4.tar Hashtable.cpp

HASHTABLE (80 POINT PART)

- Follow the instructions above to get lab template code.

- Remove all templates. This hashtable should be for strings only... with a twist.

- Make a private struct in hash named "key_line". This will store a key, as well as a vector of ints for all line numbers that "key" occur in input. We'll get to that in a bit. =)

  - Make key_line::inuse(), which will return TRUE if the key is not empty. FALSE otherwise.

  - Make key_line::operator==(), which will return TRUE if a STRING (yes, a STRING) is equal to key. FALSE otherwise.

    - PRO-TIP: Make this function "const" if you don't want a textbook of errors thrown at your face... (and i'm not joking...)

  - Overall,

```
class hash_table {
  private:
    struct key_line {
      bool inuse();
      bool operator==(const string &) const;

      string key;
      vector<int> lines;
    };

  public:
    /* getting there */
};
```

- Think what needs to be done... we want a hash_table which stores strings and the lines where they occur.

(Ex N = 13

FILE "test.txt"

This is a test

indeed a test tbqh

HASHTABLE

INDEX	KEY	LINES VECTOR
0	This	1
1	test	1 2
2		
3		
4	is	1
5		
6	a	1 2
7		
8		
9		
10		
11	indeed	2
12	tbqf	2

* This is a toy example to show how your program will store data. It may not be 100% accurate.

- How to achieve this? Change the hash_table table vector to vector<key_line> type.

  - Must change all functions to account for this. All comparisons checking if table[i] is blank should be replaced with the !table[i].inuse() call. And comparing with a string is table[i] == "A STRING" due to operator overload. Opposite is !(table[i] == "...").

- Modify nextprime() to compute the next prime number DOUBLE of the input number... plus 1
  - Basically N = (N * 2) + 1

- Modify insert() to work as such:
  - If table[index] isn't in use, set key and increment num_inuse.
  - Call std::find on the lines vector to see if the line number has already been added in. If not, add it in.
  - If num_inuse ≥ max_inuse, resize.

- Write hash_table::find to return a const vector<int> &. It looks in the table and returns the line numbers in the key_line struct. If the key isn't in the hash table, a blank vector is returned.
  - HINT: This can be done in one lines. No new blank vector is needed. If key isn't in the table, table[...].lines is guaranteed to exist and is blank.

- Add a new private integer into hash_table, named "collisions". Set it to 0 in constructor. It works like this:

- Add an extra argument to qprobe which, if true, increments collisions per call of the while loop in that function.

- The only time this new argument is true is in the insert function. It is false in every other call.

- Add a bool and three functions:
  - private bool named "show_stats" (or something like that). Set it to FALSE by default in constructor.

  - public function "set_showstats" to set show_stats to TRUE.

  - private function "showload". Prints out a "**N = " + table.size() + " load = " + ratio + "\n" where ratio is num_inuse / table.size() as doubles. 2 decimal places. So 0.12, 0.34, etc.

  - public function "insert_done", which just calls "showload". Dont ask me why. It should only print if "show_stats" is TRUE.

HASHTABLE (50 POINT PART)

- Modify hash_table constructor to take an int. This will set the initial size of the hash table internal vector.
  - Instead of setting the size to the int passed in, set it to nextprime(that int).

- Modify main to allow for command line arguments. Specifically:
  - -N <number>
    Initial hash table size
  - -f <file_path>
    Path to text file to read in to hash table.
  - -showstats
    Make program print additional information.

- Do not assume an order to these arguments. They can be used in ANY order.
  - Except the obvious. A number must come after "-N" and a path must come after "-f".

- Start reading data from file into hash table.
  - Keep an integer called "line_num". Read each line in the file via "getline".
  - Make a function named "remove_punctuation", takes a char as argument, and returns a char. If ispunct(c) is true, return a SPACE. Otherwise, return the character.
    - #include <ctype.h>
    - It sounds weird, but this lets you use std::transform on the string to remove all punctuation.
    - C++11 Lambdas are OK. Tell me if you get deducted and I'll happily fix it.
  - As hinted above, use std::transform with your new function to remove all punctuation.
  - Put the filtered line into an istringstream, then read each word from it. Toss those into the hash table.

- Start accepting input from stdin (cin). Read in words. Search the hash table via find, use the const vector<int> & it returns to report lines where the word occurred.

- How will you print out the original unmodified lines? Simple. Store a vector<string> in main and insert each line read via getline prior to any filtering.

- After cin is exhausted, if the hash table is to print stats, have it print in a destructor (hash_table::~hash_table) when main terminates.

  - Print a newline before anything.

  - Nothing much to say for this. Just match the solution executable.

- Done.