Project 1 - Hash Function DIY Contest

Designed by Yiran "Lawrence" Luo, CSE 310 Spring 2024

- **Type:** Individual or a 2-member team.
- **Synopsis:** Design your own hash function to encode ordinary strings and materialize simple uniform hashing (as is indicated in Hash Table lecture).
- **Keywords:** Hash table, hash function
- **Programming Language:** C++
- **Requirements:** You may code on your own PC but you have to make sure your program compiles and runs correctly on general.asu.
- **Deliverables:** A zip/tar archive containing the following:
    - **main.cpp**
    - **hash.cpp and hash.h**
    - **Any new .cpp/.h files you create on your own for convenience**
    - **Makefile (A standard one is given to you)**
    - A **README.txt** with your name(s) and a custom team alias. This file briefly describes the functionality of your program.
- **Full score:** 20 pts
- **Deadline: Thursday, Feb 29th at 2359 hours**, after which late submissions will not be accepted. **No extension will be made because -**
- **+2 extra credits for the best-performing 5 teams/persons, detailed in the final section .**

Restrictions

You are ***only*** allowed to use the following functionalities in C++ (C99 standard):

- <string>
- <cmath>
- <stdio.h> and <stdlib.h>
- <iostream> <fstream> and <sstream>
- struct and class
- Variable and pointer types of: int, char, string, float, and double

Using any of the following functionalities **is prohibited** since they give you an unfair edge:

- <vector>, <map>, <set> or any predefined container library defined here.
- rand() or any randomizer as part of your hash function.

Your submission zip/tar/7z package **shall not exceed 5MB.**

- You shall not include your built binary executable.

Failure to comply with any of the above restrictions will result in a hefty penalty of 75%.

Overview

1. Download the starter package from Canvas, in which you are given the code templates and sample input cases. A gold standard demo is also provided and you may run it and see how the outputs should look like.

2. Your job is to design A) **A Chaining hash table** that categorizes and stores string tokens loaded from a single input file, and B) **Your own hash function** that sorts each string into a corresponding slot in your hash table.
    a. Each hash table slot is a linked list of the Node struct.
    b. You are free to design however you manipulate the input key inside the hash function using built-in math functions. But,
    c. Your hash function should always return the same hash value given the same input string in any occasion. In other words, <u>your hash function should never return different outputs in different situations given the same input. Thus if your hash function turns out to be inconsistent, your program would be considered incorrect.</u>

3. Once you fit all the strings from the input file into the hash table using your own hash function, print out the following information. The following example in `Courier font` shows the expected printed outputs w.r.t the first 5 slots and 9 specific tokens, when your hash function sorts each token by its first letter:

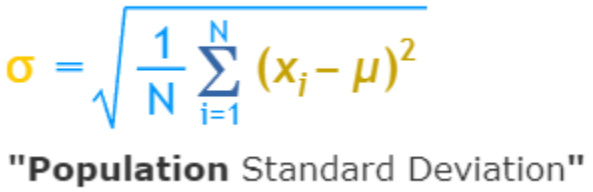
    a. The contents of your hash table's first 5 slots:

    ```
    ==== Printing the contents of the first 5 slots ====
    Slot 0: apple abandon Amazon Applebee
    Slot 1: banana barbaric boring Boeing
    Slot 2:
    Slot 3:
    Slot 4: elephant
    ```

    b. The lengths of every slot as in:

    ```
    ==== Printing the slot lengths ====
    Slot 0: 4
    Slot 1: 4
    Slot 2: 0
    Slot 3: 0
    Slot 4: 1
    ```

c. **The** [standard population deviation](#) (std-dev) of all the hash table slot lengths. Notice the N in the following formula is equal to our k.



"**Population** Standard Deviation"

This float-type number gives us the idea how fairly your hash function sorts the strings from a certain input file, e.g.:

```
==== Printing the standard variance ====
1.8330
```

Unlike the load factor, the smaller the standard deviation is, the better your hash function performs for each test case. An ideal hash function should always yield a low standard deviation given any input sequence.

Input format

You load the contents of a .txt file via <stdio> into an array of strings. (This part of code is called a 'tokenizer' **and is already provided as part of main.cpp**).

We assume the format of an input file **ALWAYS** follows these rules:

1. The first line specifies the number of hash table slots k (5 <= k <= 100)
2. The following lines consist of one token per line, in only upper and lower case letters. There will not be duplicate tokens in a single input file.
3. The number of tokens N does not exceed 500.

Example (sample_input.txt):

```
5
Amazon
Boeing
apple
Applebee
abandon
banana
elephant
boring
barbaric
```

Building and testing

You are supposed to use a Makefile to build your C++ program. A sample Makefile is given and you are free to alter it at will. Once a build is complete, you should be running your program with a such command in the shell prompt (assuming the executable is named `encoder`):

```
./encoder < inputs/input.txt
```

To test with another test case file, simply replace the path to your input file:

```
./encoder < inputs/input2.txt
```

You are more than welcome to create your customized input files/test cases and test the robustness of your hash function design.

Rubrics and Bonuses

We grade your submission based on the correctness of :

1. The implementations of the hash table and your customized hash function, **(4 pts)**
2. The 3 print-outs of your hash table information (the first 5 slots' contents, all slots' lengths, and the standard deviation), **(3 x 4 pts)** and
3. Turning in *all the required deliverables* for us to rebuild your executable successfully on general.asu **(4 pts)**.

When we are grading your work on output performance, we will be using a different set of test cases from the ones given to you in the starter kit.

The top 5 teams/persons that achieve the lowest average standard deviations with our test cases will be granted 2 extra credits (2% to the overall grade) and will be decorated in our Hall of Top 5 Benchmarkers.