CSCI 2270 – Data Structures

Guidelines for final project

Instructors: Maciej Zagrodzki, Christopher Godley

Hashing Implementation and Analysis

In this project, you will be implementing and comparing performance characteristics of Hash Tables.

You will create a Hash table with 10009 slots (hereafter denoted as TABLE_SIZE). Why 10009? Hash Tables are typically created with size equalling a prime number.

You need to implement Hash Tables with combinations of the following hash functions and collision resolution mechanisms:

Hash Functions to implement:

1. h(x) = x mod TABLE_SIZE …….. (mod stands for the modulus operator)
2. h'(x) = Floor(x / TABLE_SIZE) mod TABLE_SIZE….. (the Floor() operator returns the integer part of the quotient)

Collision Resolution Mechanisms to Implement:

1. **Chaining with a Linked List :** You will implement chaining with a LL to resolve conflicts.
2. **Chaining with a BST**: Here, the chaining mechanism will utilize a BST instead of a Linked list.
3. **Linear Probing**: Implement Linear Probing with step size = 1

4. **Cuckoo Hashing :** Here we will explore an alternative open addressing strategy known as cuckoo hashing.

   The main idea is to use two hash functions instead of one (we will use the ones mentioned above) . We will maintain two hash tables, each associated with one hash function.

   Lookup: When searching for an item, we will first examine the location obtained from the first hash function. If we do not find the item we are looking for, we look for it in the location obtained from the second hash function. Therefore, we need to examine two locations before declaring that an item does not exist.

   Delete: As mentioned above, we look for the item in its two potential locations and then remove it from the table (by setting the value in that slot to -1)

CSCI 2270 – Data Structures

Guidelines for final project

Instructors: Maciej Zagrodzki, Christopher Godley

Insert: We will first try to insert the item in the location obtained from the first hash function. If that gives us a collision, we try to insert the item in its second location. If both locations are full, we displace the element in the current item's first location and move the displaced item to its alternative location. As you can imagine, the displaced item's alternative location may be full as well. Therefore, we need to move that item to its alternative location. If needed, we continue this process of displacing items till an item has an empty alternative location.

If this process causes a cycle i.e. this process of displacing goes into an infinite loop, we choose two new hash functions and rebuild the hash table (sometimes called rehashing). To obtain new hash functions, we only need to increase the size of our table. The changed TABLE_SIZE means that we have two new hash functions.

More information can be found here : https://en.wikipedia.org/wiki/Cuckoo_hashing

Performance Evaluation

You will be provided with two datasets of integers to be inserted into the hash tables.

You will need to compare performance on the 3 operations of a hash table: insert, lookup and delete. To this end, you will measure the average time taken for each operation. This will be done by performing 100 instances of each operation and then averaging them. For example, to measure the time required for an insert, you will perform 100 inserts, record their time and take their average.

You will utilize both hash functions for each implementation of the hash table. Therefore, the two chaining mechanisms and the linear probing mechanisms will have two implementations each while cuckoo hashing will have only one implementation.

Since the performance of each scheme varies according to the load factor of the table, you also need to measure performance at different load factors. Thus, you will not insert the entire dataset at once. Rather, you will first insert enough elements so that the load factor becomes 0.1, then, you will perform 100 inserts, 100 lookups and 100 deletes, record their average times and only then insert additional elements from the dataset till the load factor becomes 0.2.
In this way, you will record average performance for each of the 3 operations at Load Factor = 0.1, 0.2, 0.5, 0.7, 0.9, 1.

CSCI 2270 – Data Structures

Guidelines for final project

Instructors: Maciej Zagrodzki, Christopher Godley

Note: For Cuckoo hashing, it may be the case that you never get to high load factors because as the table gets filled up, you may need to resize the table. In this case, you should record the number of times you needed to resize the table for each dataset as well.

The data obtained from the above process would be drawn as a series of graphs. You will, therefore, have one (time vs load factor) graph per dataset, per hash function, per collision resolution implementation. All graphs should have the same scale for clarity and ease of visual comparison.

Extra Credit (15% of total)

Create an associative array and compare its performance with the C++ STL implementation of map<string, int>.
Associative arrays are a generalization of standard arrays. Standard arrays use integers for indexing whereas associative arrays can use numeric as well as other data types. They are typically used to store (key, value) pairs. For those with experience in Python, Python's Dictionaries are a kind of associative array i.e they store (key, value) pairs.

For instance, suppose we have stored pairs of student names and roll numbers in an array indexed with the student's name, we would access the roll number of a student named "John" like so : my_assoc_array["John"]. This would give us an integer denoting John's roll number.

Additional details can be found here : https://www.geeksforgeeks.org/associative-arrays-in-cpp/

Your job is to create an associative array based on hash tables to map strings to integers. The main idea is to apply a hash function to the string and use that to index into a hash table. At that slot in the hash table, we would store the integer value. We will utilize open addressing as the collision resolution mechanism.

You will compare the performance of inserting a (key, value) pair and deleting a value in your implementation vs the C++ STL implementation of map<string, int> on the dataset provided. You can use different hash functions and compare performance.

You will get **up to 15%** of the project grade as extra credit for implementing associative arrays and conducting a performance comparison. Your project report should include documentation answering the above questions for the extra credit section separately.

CSCI 2270 – Data Structures

Guidelines for final project

Instructors: Maciej Zagrodzki, Christopher Godley

Project submission (to be submitted by December 8th on Moodle)

You are required to upload everything as a zipped folder as a submission on Moodle.
Turn in:

1. Your entire project including all the codes, header files, and any other additional files you've used. (**75% of the grade**)
2. A short report (2-3 pages) (**25% of the grade**)
    - Methodology: you can include some pictorial representations to explain the flow or design of your project. (Use https://lucidchart.com to create flow diagrams). We would request you to add references if you used any.
    - Results: add some of the final outputs which you achieved through your project.

We would assign some portion of the grades for following the coding standards: indentation, space, new lines, meaningful variables and method names. It is recommended to use Github for your project (although it's not mandatory). Make sure you have your code backed up regularly since last moment laptop problem excuses will not be granted.

Interview grading (Floating deadline: Last two weeks)

Interview Grading will be held in the last two weeks of the semester. Each team will set up a time to meet their respective TA. Depending on when this meeting is scheduled, the team would be expected to have completed between 85-100% of the project. Exactly what constitutes a certain percentage would be communicated by the TAs.

Interview grading slots will be released on Moodle. You need to set up an appointment with your TA for the same. Students will get graded for their code in their interview. Interview grading is mandatory for everyone.

Expectations:
We expect you to have a clear picture of what you did and how did you do the project.

- Failing to explain the methodology will give you poor grades.
- You will be graded for each functions you are supposed to do.
- You have to explain your result. For example, why your hashing function is providing bad performance, how can you improve it.

CSCI 2270 – Data Structures

Guidelines for final project

Instructors: Maciej Zagrodzki, Christopher Godley

Written requirements:

25% of the overall project grade

Your project needs to include a 2 - 3 pages paper that describes your project and the results of your analysis of the three implementations. Your paper needs to include the following sections:

Purpose:
What is the purpose of this project? What are you evaluating?

Procedure:
What data structures are you using for your implementations? This section should include a description of how each implementation works. Also include what metric you will be using to measure the runtime differences between the implementations.

Data:
You can plot a graph of datasets (integer value vs index) used in this project. Describe the distribution of values in the dataset.

Results:
Describe how each implementation performed. This section should include graphs obtained from your implementations. You also need to include the mean and standard deviations of the runtimes (at each load factor) of the 100 instances of running an operation for each implementation.
Do your implementations perform as expected on the datasets? Why or why not? Include a description of why your implementation performs like it does. You can use additional datasets or hash functions of your choice (in addition to the ones mandated above) to better illustrate the differences in your implementations.

CSCI 2270 – Data Structures

Guidelines for final project

Instructors: Maciej Zagrodzki, Christopher Godley

Timing Code:
One simple way to time your code in C++ is using the following method:

```
#include<iostream>
#include<ctime>
using namespace std;
int main(){
   int startTime, endTime;
   double execTime;
   startTime = clock();
   /*
   EVALUATION CODE GOES HERE
   */
   endTime = clock();
   execTime = (double)(endTime-startTime)/CLOCKS_PER_SEC;
   cout << "execution time: " << execTime << endl;
   return 0;
}
```

Work can be individual or in pairs:

Students can work individually or in teams of two. It is assumed that every team will write their own code and produce their own write up. Start off early. Since we have very limited time, there will be no extensions in any of the project evaluations.

If a team dispute arises, it will not be mediated by anyone on the teaching team. (For example, if student A agrees to work with student B, then student B decides to not do their part, Student A still needs to find a way to complete the entire project.)

Submission:

Submit your code and report as FinalProject.zip to the Final Project Submit link on Moodle by December 8th.