Lab_hash : Hellish Hash Tables

Lab #11 – April 21 – 25, 2021

Welcome to Lab Hash!

Course Website: https://courses.engr.illinois.edu/cs225/sp2021/assignments/

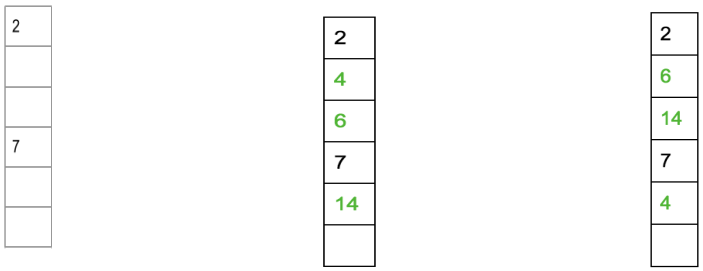
Overview

Hash tables are the most efficient data structure you will learn about in CS 225. On average, *insert*, *remove*, and *find* operations all run in average O(1) for hash tables no matter the size of the table or how many elements are in it. This is why hash tables are a favorite data structures among CS people; they are often the underlying structure to library data types (i.e. dictionaries or unorderedMap in C++).

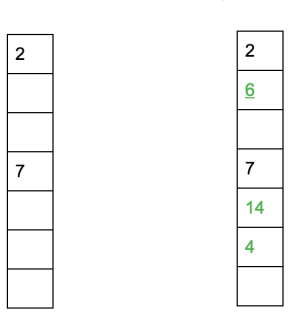
Linear Probing:

A **collision** in a hash table happens when the hash function **h** gives the same index **h(v)=h(w)** for different data values **v** and **w** as they are being inserted into the table. Linear Probing is one strategy to deal with collisions in a hash table. The way it works is: if **h(v) = i** causes a collision (index **i** is already occupied), then we increment **i++** until we find an empty spot in the table. Use the modulus function to "wrap around" and continue incrementing from the start of the table when the end is reached. The idea of **double hashing** is to add a second hash function that will be used as a step function to avoid clusters.

Exercise 1.1: (Linear Probing) We want to insert 4, 6, and 14 into the hash table below. Suppose that our hash function gives: h(4) = 1, h(6) = 0, and h(14)=2. How would the table look after inserting 4, 6 and 14 in that order? What about if we insert 6, 14, then 4?



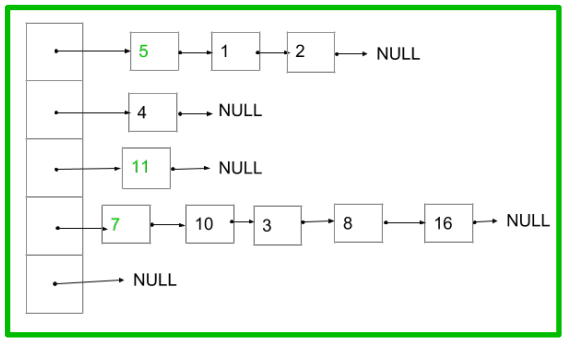
Exercise 1.2: (Double Hashing) Suppose that our first hash function gives: h(4) = 1, h(6) = 0, h(14) = 5. And suppose our second hash function (used as the step function) gives: h'(4)=2, h'(6)=1, h'(14) = 3. How would the table look after inserting 6, 4, and 14 in that order?



Separate Chaining:

In a hash table that uses separate chaining, instead of holding the inserted data values, each slot in the table holds a **pointer** to the head of a linked list where the actual data values are stored; this is called **open hashing**. Thus, if a collision occurs, then the new data value is simply inserted at the **head** of the linked list for that slot. For example: if **h** maps **a**, **b**, and **c** all to index **i**, then at index **i** we will have the linked list: c→ b→ a→ NULL.

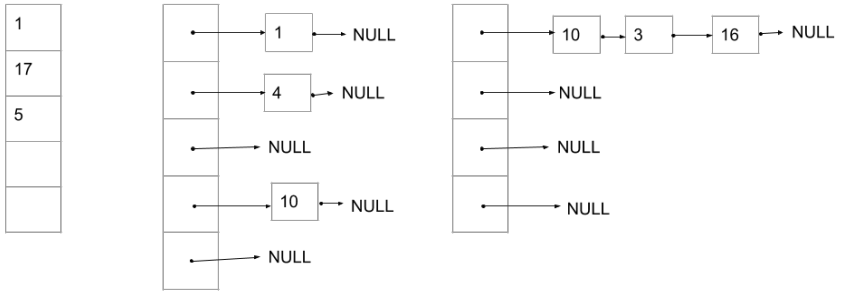
Exercise 2.1: We want to insert 5, 11, and 7 into the (overloaded) hash table below. Our hash function gives us: h(5)=0, h(11)=2, and h(7)=3. How will the hash table look after we insert these elements? Which of the elements causes a collision?



Exercise 2.2: What is the worst possible runtime for the insert() function if we are using separate chaining to resolve collisions? What about the worst possible runtime for the find() function?
insert() will always be O(1) because we insert into the head of the linked list. find() however can be O(n) at the worst.

Resizing the Table:
The load factor of a hash table is an important statistic when it comes to gauging when to resize the table. The load factor is calculated by dividing the number of elements inserted into the table by the size of the table. Once the load factor is greater than or equal to a chosen threshold, the size of table should be approximately doubled to the nearest prime number >= 2*(old table size). This is to help avoid the issue of clustering and achieving a better uniform distribution. Since **h** depends on the size of the table, the hash values of existing elements WILL change once we resize; so REMEMBER to re-insert (re-hash) every element into the table after resizing.

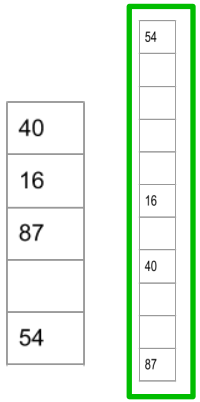
Exercise 3.1: For each of the hash tables below, calculate the load factor. If we have a load factor threshold of 0.7 should we resize these hash tables?



3/5 = 0.6 3/5=0.6 3/4=0.75 ← load factors

Y / N Y / N Y / N ← resize?

Exercise 3.2: Suppose that our hash function for the table below is a simple mod function: h(x) = x mod TABLE_SIZE and we use linear probing to resolve collisions. Our load factor threshold is 0.7, so we will need to resize the table. Draw the new resized table.



Closest prime number to 2*5 is 11; so the new hash function is: h(x) = x mod 11.

In the programming part of this lab, you will:

- Write the *insert* function for Linear Probing, and the *remove* and *resizeTable* functions for Separate Chaining
- Solve two fun puzzle problems using hash tables: *AnagramFinder* and *LogfileParser*

As your TA and CAs, we're here to help with your programming for the rest of this lab section! 🥰