Deb’s Epic Hash Table Assignment

In this (ungraded) assignment, we’re going to be looking at the time complexities of some hash table operation. Note that this assignment is not representative of the types of questions that will appear on the exams.

If you have no idea what’s going on, feel free to ask your peers or a TA for help. If you do know what’s going on, now is a good time to make friends. We will not be releasing the answers to this assignment - focus on understanding how you arrive at each answer rather than memorizing the results.

Let ‘n’ be the number of elements in the hash table

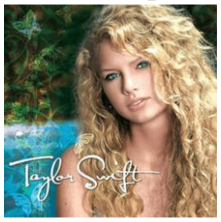
The efficiency of hash table operations is closely related to how well the hash function is implemented. A valid hash code is not always a good hash code, and we’re going to be looking at a (sort of) real world example to see why a having a good hash code is so important.

Imagine you’re driving towards your unrealistically large mansion in your extremely expensive ~~Rolls-Royce~~ Tesla, with your significant other in the passenger seat. They tell you with tears in their eyes that after the 27th update this week, iTunes has crashed. Armed with your CS 1332 knowledge, you decide to implement a piece of software to replace iTunes. As a diligent student, who was somehow awake during recitation, you decide that it’s probably a good idea to keep track of the set of all the albums in the music collection using a hash table.

You’ve almost finished writing the code for the hash set when your phone rings ominously. It turns out to be your enthusiastic cousin from UGA who (somehow) found out what you were up to and wants to lend a hand. Most of the code has been written, but in order to make him feel included, you let him implement the hash code function. The hash code he uses is as follows:

Hash code of album = number of letters in the album artist’s first name

Naturally, the love of your life’s music collection looks something like this:

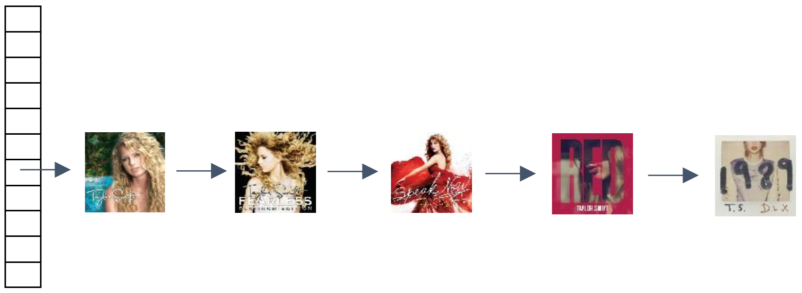


Your implementation uses external chaining with a linked list for collision resolution.

Terminology can be confusing

The terminology for collision resolution methods can be confusing. Remember:
External Chaining <--> Closed Addressing <--> Open Hashing
Probing <--> Open Addressing <--> Closed Hashing
A nice way to remember the second one is with the word ‘POACH’:
Probing <--> **O**pen **A**ddressing <--> **C**losed **H**ashing
It’s fairly simple to infer the rest of the terminology from this information

You put all of the albums into the hash table, and it ends up looking like this:



Note that the length of the used array is 11 and the compression function is simply a mod by the length of the array.

As we can see, the hash code of every album is the same. Is this a good hash function? ___

What is the time complexity of the 'get', 'contains' and 'remove' operations? ___

For the 'add' operation, it may be tempting to simply add to the front, but we must be careful to look through the list, because we cannot have duplicates. What is the time complexity of the 'add' operation? ___

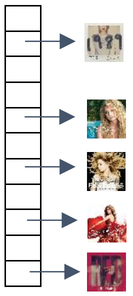
Is this the average or worst case? _________

Would any of these time complexities change if we had used probing instead? ___

In the real world we can assume, on average, that the we are using good hash functions. Since your current hash function is leading to lots of collisions, you decide to swap it out for a new one. You phone your ridiculously attractive and tragically underpaid former CS 1332 TA, who gives you the following hash function:

Hash code of album = year of album's release

There are now no collisions and the hash table looks much less scary:



What's the time complexity of the 'get', 'contains' and 'remove' operations now? ___

What about 'add'? (remember to account for resizing) __________

The hash function you have used works very well for the data you have, (Any self-respecting CS 1332 TA's knowledge of Taylor Swift music is impeccable) but what if Ms. Swift had decided to date 10 boys in one year? This would have led to 10 albums with the same release year and thus the same hash code. In the real world, more complicated hash functions are generally used in order to avoid collisions as much as possible. The excellent average case time complexities of hash table operations are what makes them so popular.