CS-UY 1134: Data Structures and Algorithms
Lab 10: Hash Tables and Maps

Friday, August 9, 2019

Instructions

This lab will focus on hash tables.

Try to solve all of the problems by the end of the lab time. As added practice, it is a good idea to try writing your code with pen and paper first, as you will have to do this on the exam.

When you type up your code, write some test cases. Try multiple test cases for each function you have to implement, and consider edge cases carefully.

As a reminder, you *may* (and are encouraged to!) discuss these problems with your classmates during lab and help each other solve them. However, make sure the code you submit is your own.

What to submit

If you believe you have solved all the problems, let a TA or the professor know and they will check your work. This is a good chance to get feedback on problems similar to what you might see on the exam.

If you don't finish by the end of the lab, you may submit something on NYU Classes by Sunday August 11, 11:55 PM.

Either when a TA has checked your solutions, or by the Sunday submission deadline, submit one ".py" file containing your solutions to the programming problems, and one additional file containing your answers to the written problems. Submit your files to the "Lab 10: Hash Tables" assignment on NYU Classes.

1 Written Problems

Complete the following written problems. You do not have to submit your work, but do them anyway and make sure that you can answer similar problems.

1. Perform the following operations to an initially empty hash table with table size $N = 4$. Use the Muliply-Add-Divide compression method, which maps an integer k to:

$$h(k) = [(ak + b) \mod p] \mod N$$

where p is a very large prime number, a is a randomly chosen number from $\{1, 2, \ldots, p-1\}$, b is a randmoly chosen number from $\{0, 1, 2, \ldots, p-1\}$, and N is the size of the table.

For this problem, use $p = 101$, $a = 1$, and $b = 25$. The items you will insert/delete do not have any values associated with them (they are just integers). Note that you should resize/rehash the table whenever $n >= N$.

(a) Insert 6

(b) Insert 50
(c) Insert 301
(d) Delete 6
(e) Insert 100
(f) Insert 37
(g) Insert 12
(h) Insert 227
(i) Insert 92
(j) Delete 37
(k) Insert 88
(l) Delete 100

2 List Intersection

Write a function that, given two lists, `lst1` and `lst2`, returns a new list containing all the elements that appear in both `lst1` and `lst2`.

Give your implementation in the function

```
def list_intersection(lst1, lst2):
```

As an example, the call `list_intersection([5, 6, 1, 10], [8, 1, 9, 5, 3, 8])` could return `[1, 5]`. The order of the elements does not matter, but it should not contain duplicates.

Implementation Requirement: Your function should run in $\Theta(n)$ **average time**.

Also include as a comment in your code: What is the **worst-case** running time of your code? **Seriously. Answer this question if you want full credit.**

Think about: What if you wanted to modify your code to achieve the fastest worst-case running time? How would your implementation change? How would this affect the average running time?

3 Find Mode

The **mode** of a list is the element which occurs the most often. For example, the *mode* of the list `[1, 3, 2, 1, 2, 1]` is 1.

Write a function:

```
def mode_of_list(lst):
```

that takes a list and returns the mode of that list. Your implementation should optimize the **average case** running time.

Analyze the **worst-case** running time of your code. What data structure could you use to improve the **worst case** running time? What would be the average case running time of this alternate solution? **Include your answers to these questions as comments in your code**.

4 Two Sum on Unsorted List

Earlier in the summer, you implemented a worst-case *linear* time algorithm for the *two sum* problem on a sorted list. This problem asked to find, given a sorted list and a target number, two elements in the list that sum to the target number.

Now, write an implementation to solve the same problem for an *unsorted list*. That is, implement the function:

```
def two_sum(lst, target):
```

that takes as input an *unsorted* list of numbers `lst` and a single number, `target`, and returns a pair of indexes (as a tuple), indicating the indices in `lst` of two numbers whose sum is `target`.

For example, if `lst` is the list `[-5, 2, 8, -3, 7, 1]`, then the call `two_sum(lst, -1)` should return `(1, 3)` because `lst[1] + lst[3]` is -1.

If there is no pair of numbers in `lst` that sum to `target`, then your function should return `(None, None)`.

5 Anagrams

Two words are *anagrams* if the letters of one can be arranged to make the other. For example: "enraged" and "angered" are anagrams.

Write a function:

```
def is_anagram(str1, str2):
```

that takes two strings, `str1` and `str2`, and returns `True` if `str1` is an anagram of `str2` (and false otherwise). You can assume that neither string contains spaces. Do not worry about whether `str1` or `str2` are valid English words, simply determine if `str2` can be obtained by rearranging the letters of `str1`.

Attempt to acheive the best possible **average case** running time. You may use additional space if necessary.