CS532: Homework 6

Hash Tables with Chaining

Question 1: Basic Code

You will be creating a hash-table that uses a doubly-linked list for resolving collisions. Create a class called `ht_element` for storing a key, a next pointer and a prev pointer.

Create a class for `chained_hash` that includes an insert, search and delete method with arguments consistent with the textbook in Section 11.2. Insert and delete should take as input an object of type `ht_element`, and search should return an object of that type. Insert does not need to check that the key is not already in the list. If search cannot find an object with the requested key, it should return None.

The initializer for this class should take a size for the table. Your code should then round up the size to a power of 2. Save this value as part of your object as `self.p`, which will be needed in a subsequent question. The actual size should be saved with the hash table object, in a variable called `size`. The array for holding the pointers to the linked list should be called `T`.

Create a method called `hash_function` for the hash function that uses the division method from the textbook. You can assume that the keys are integers.

Create your own test code to make sure it works. Then, test your code with the unittest code in hw6test.py.

Hand in your code, and state whether it passed all of the tests in hw6test.py.

Question 2: Satellite Data

Create a class `my_element` that inherits `ht_element` and write `test2` that demonstrates you can store and retrieve satellite data in your hash table. For your satellite data, use name and address.

Your initializer for `my_element` should take as input the name and address and initialize those fields in your `my_element` object. Your initializer should also call the initializer for `ht_element`. Look up on the web how you can use `super()` to refer to the inherited class.

Hand in your code for `my_element`, and hand in code that demonstrates that it works. You do not have to turn in the output of the demonstration code.

Question 3: Multiplication Method

Create the class `mult_hash`, which is similar to `chain_hash`, except it’s hash function will use the multiplication method given in the textbook using Knuth’s suggestion for $A = (\sqrt{5} - 1)/2$. Assume that the word size is 32 bits, and keys will be smaller than this. Just as with the previous version, you will input to the initialization routine the minimum number of hash entries you want to use.

You want to do as much work in the initialization routine, and as little work in the hash function as possible. In the initializer, first compute $p$ as you did in the previous version, and save this value (as it is needed by the hash function). You can assume the word size is 32 bits, so do not pass this value in. In the initialization code, compute the value of $s$, which will be an integer.

In your hash function, you will compute $k \times s$, then computing the rest of the key using only bit-wise ‘and’ and bit-wise shifts to compute the answer. Do not use modulo or division or a second multiplication.

During debugging, verify that you are computing all of the intermediate values correctly, and that you get
the right answer for the example given in the textbook.

Set up class `mult_hash` so that it inherits `chain_hash`. Your new `hash_function` will override the one in `chain_hash`. Your new initializer should call the initializer in `chain_hash` to compute \( p \) and create \( T \) so that this is done only once. As the insert, search and delete have not changed from your previous version, class inheritance means you do not have to re-specify these.

Forests for Disjoint Sets

**Question 4: Code**

Implement the code for representing a disjoint set with a forest (Chapter 21.3). Your code should be as faithful as possible to the pseudo code given, which implements union by rank and path comprehension. Thus `make_set` and `union` should not return a value.

The code will differ from the textbook as you will be creating a single class `ds_element`, with the textbook functions as object methods. Since you are using a class, you will need a class initializer, which must be separate from `make_set`. Your class objects should include the variables `self.v` (value), `self.p` (parent pointer) and `self.rank`. Value should be set at initialization of the element, while parent and rank should be set to `None`.

Note, in the disjoint-set forest implementation, sets are not explicitly kept as they are in the linked-list implementation. They are implicitly stored via the parent pointers of the underlying objects.

Hand in your code. Explain why it would make more sense to subsume the `make_set` function as part of the class initializer.

**Question 5: Time Efficiency**

Section 21.4 gives a proof of the runtime of \( m \) disjoint set operations on \( n \) elements. Do an empirical examination of this. So that there is just one dimension that needs to be varied, keep \( n = m/3 \). First do the \( n \) make-set operations (these make-set operations count towards the \( m \) set operations). Second, do the following in a loop until you have the rest of the required operations: (a) randomly pick the first object, (b) use find-set to find its representative, (c) randomly pick a second object until you find one that is not in the same set as the first object, (d) union the two sets. Each loop through (a) through (d) will result in at least 3 operations, but more when you find that the second object is in the same set as the first. Remember, each find-set and each union count towards the total.

Hand in your code and a chart/table showing your results. Comment on how the time needed is varying with \( m \).

**Question 6: No Heuristics**

Create a second class in which you do not include the union by rank and path compression heuristics.

Hand in your code.

**Question 7: Time Efficiency**

Repeat your time analysis but with the version without heuristics. For your analysis code, have it take as an input which class it should use.
Hand in your new analysis code, and give your results when not using the heuristics, and discuss how they differ from using the heuristics.