Computer Science 136  
Data Structures  
Lecture #24 (November 22, 2021)

A lecture on design.

1. Questions?

2. The design of a new Hashtable, that keeps its keys in insertion order. A Dict, after the similar object from Python.

   (a) The motivation:
      i. The desire to quickly traverse the values of a Hashtable in insertion order.
      ii. In the Hashtable class, iteration traverses the entire table, looking for entries that are not null and not reserved.
      iii. We’re looking for an approach that allows us to only traverse the entries of the table that are being used. Why inspect unused entries?

   (b) The approach:
      i. (N.B. This is a “clean room” design: I’ve not looked at Python’s implementation.)
      ii. We’d like to keep a linked list of key-value pairs inserted.
          A. A doubly-linked list allows us to add and remove values very quickly.
          B. We’ll use a dummy-node approach, and make the list circular. This will avoid excess storage and the use of null pointers. The head node will be identifiable: it contains a null reference to a key-value pair.
          C. We will use null pointers to keep track of reserved entries in the table; these entries were in the linked list, but now are not.
      iii. We’ll use the hash table vector to store elements of the linked list, located by hashing in the normal manner.

   iv. We need to be very careful about thinking about when we’re adding versus updating a key-value pair. The first involves adding an entry to the linked list; the other does not.

   v. The remove method removes an entry from the Dict and the shadow list, but it remains in the table as a reservation.

(c) The implementation:
   i. The DictEntry class. This is a single entry in the dictionary, and also and entry in the shadow list. We can iterate across entries, from the head node.
   ii. The DictIterator class. This is the iterator that traverses the shadow list, and is exposed through Dict iteration.
   iii. The Dict class. A mixture of traditional HashTable, and updated iteration.

(d) Testing: WordFreq.