L3 PROBLEM 4 (5/5 points)

Imagine we implement a dictionary, all of whose keys are U.S. social security numbers (9 digit numbers). If we used a list with 10^9 elements to represent the dictionary, we could do lookups in constant time. Of course, if the dictionary contained entries for only 20 people (or, for that matter, only $3 * 10^8$ people), this would waste quite a lot of space.

Which gets us to the subject of **hash functions**. A hash function maps a large space of inputs (e.g., all natural numbers) to a *smaller* space of outputs (e.g., the natural numbers between 0 and 5000). The space of possible outputs should be much smaller than the space of possible inputs!

A hash function is "many-to-one", that is, multiple different inputs are mapped to the same output. When two inputs are mapped to the same output, it is called a **collision**. A good hash function produces a uniform distribution, i.e., every output in the range is equally probable, which minimizes the probability of collisions.

Remember the intDict code from the previous video? `intDict` uses a simple hash function (modulus) to implement a dictionary with integers as keys. The basic idea is to represent instances of class `intDict` by a list of buckets, where each bucket is a list of (key, value) tuples. By making each bucket a list, we handle collisions by storing all of the values that hash to that bucket.

Collisions are inevitable when implementing hash tables, because generally we are mapping a really big set of inputs to a much smaller set of buckets. Thinking about the way that `intDict` implements hashing - making each bucket a list, and assigning each element to one bucket's list - what is the expected average length of the list in each bucket of the hash table when the hash table is 10 buckets big for the following number of unique insertions (that is, no two elements inserted are equal)?

1. Hash table size = 10 buckets; number of unique insertions = 10

   Expected *average* length of the list for each bucket =

   1

   Answer: 1

2. Hash table size = 10 buckets; number of unique insertions = 20

   Expected *average* length of the list for each bucket =

   2

   Answer: 2

3. Hash table size = 10 buckets; number of unique insertions = 100

   Expected *average* length of the list for each bucket =

   10

   Answer: 10

EXPLANATION:

The average length of the list for each bucket will be the number of elements inserted divided by the number of buckets available (this is assuming tha we have a good hash function)

There are many other ways to handle collisions, some considerably more efficient than using lists. However, this is probably the simplest mechanism, and it works fine if the hash table is big enough and the hash function provides a good enough approximation to a uniform distribution.

4. Consider the class `intDict`, particulary the method `getValue`. If there are no collisions, what would the complexity of `getValue` be?

   O(1) O(1)

   EXPLANATION:

   Without collisions, each key has only one value associated with it.

   Obviously, the drawback of making a hash table so huge there's no collisions is the space required for the hash table. If a set has `10^9` elements, and each element takes up even just 1 byte of space, that's still a 1 GB hash table!

5. So let's reduce the number of buckets... to 1! What would the complexity of `getValue` be if *n* elements were hashed to the same bucket?

   O(n) O(n)

   EXPLANATION:

   With only one key, all values are part of a very long list! Looking up one value means you must traverse the entire list, at worst case.

OK, so we save on space, but lookup takes forever. So, when creating hash tables, we try to optimize both the size of the table (as small as possible) and lookup time for elements (as short as possible). It turns out that by making the hash table large enough, we can reduce the number of collisions sufficiently to allow us to treat the complexity of lookup as almost O(1). I.e. we can trade space for time. But what is the tradeoff? We'll look at this using the tools of probability, in the next problem.

Check Hide Answer

Show Discussion New Post