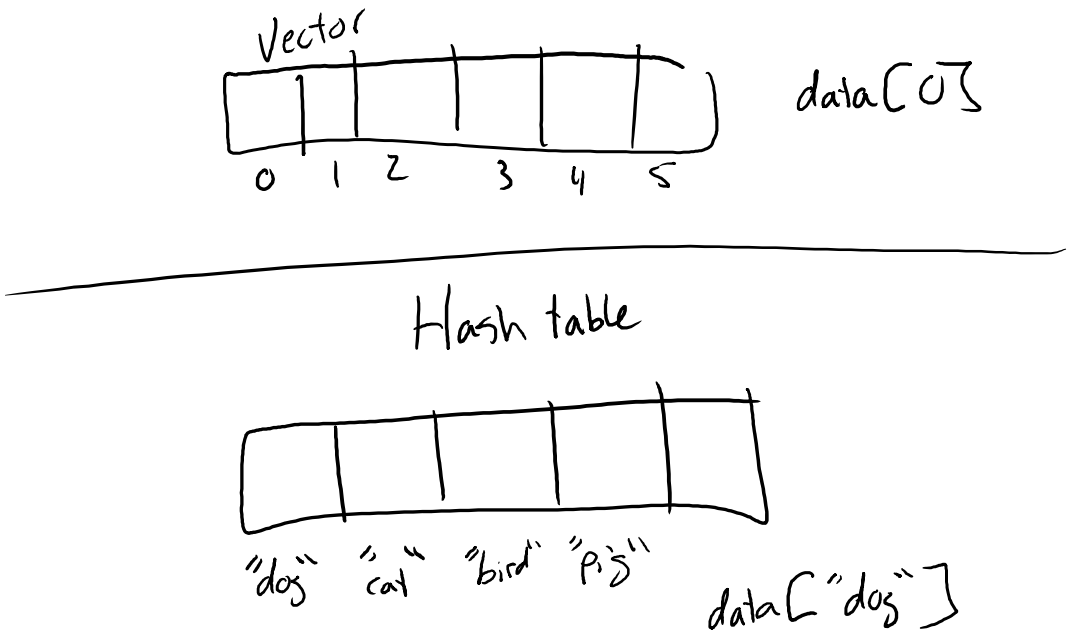
2019-10-03 Hashtables

Thursday, October 3, 2019 9:00 AM

- A hashtable is a vector-like structure that allows a programmer to use non-numeric keys for indexing.



- In C++, the HT is called an "unordered map" which references the fact that our keys are not ordered in any predictable way (e.g. key 0 may not come before key 1; "bird" may or may not come after "dog")
- From a programmer's perspective, operating on a HT is pretty easy, especially when the HT is a first-order data type (e.g. Python, JavaScript, PHP)
- In C++, we have some caveats:
  - We can only have one key type per HT
  - Looping cannot be done with a "class" FOR loop (requires FOR AUTO)
- In C++, we also have a "map" data structure
  - This is **very** different than an unordered map. Maps in C++ are trees, not HTs
- On average, HTs are just as fast as vectors
  - O(1) insert
  - O(1) lookup
  - O(1) remove (FASTER than vector)
- Downside of HTs: take up more memory: Up to 2x space required vs vector.

Key factors when designing a HT

1. How do you convert from a non-integer into an integer (hashing)
   a. Hashes should be fast and generate a wide spread of values
2. What do you do when two keys hash to the same index?
   a. **This is inevitable**
   b. These are called collisions
   c. Two broad strategies:

        i. put multiple things in same box (2D vector - separate chaining)
        ii. Find new box somehow (e.g. go one over) -> open addressing
3. How full should we allow our HT to become before resizing?
    a. The more full at HT is, the more likely #2 will happen, which slows down our HT
    b. As a HT gets more full, its performance shifts from O(1) to O(N) for the "big three" operations
    c. This is called "load factor" and ranges from 0 (empty) to 1 (full)
    d. Some HTs cannot have load factor more than .5 otherwise, they get stuck in an infinite search loop (e.g. quadratic probing)