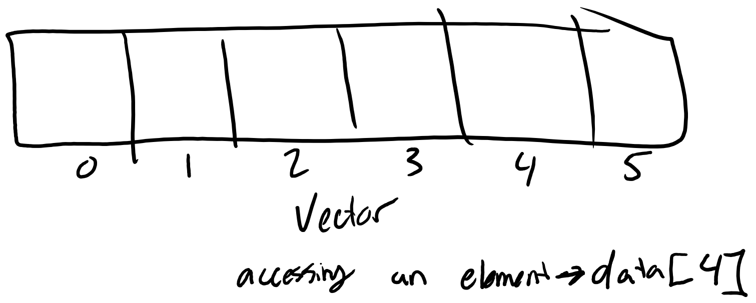
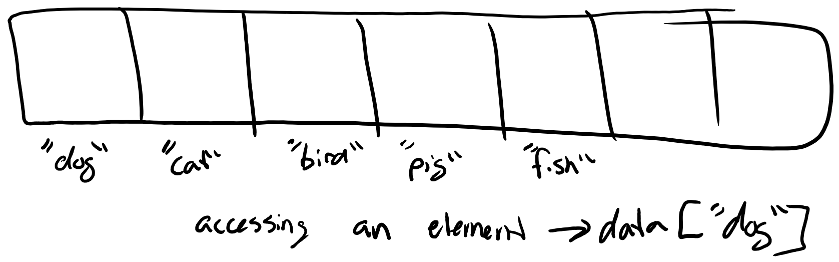
2018-03-22 Hashtables Part 1

Thursday, March 22, 2018 9:00 AM

- A hash table is a vector-like structure that allows a programmer to use non-numeric indices (keys)





- From a programmer's perspective, using hashtables is very easy. More or less just like a vector. (C++ makes this a little more difficult)
- In C++ the unordered_map is a hash table. DO NOT USE MAP. It is not a hash table! (unless you know why you're using a map)
- Key considerations:
  - As it says in the C++ name, hash tables are unordered
    - Items are added to a HT based on a hash value, which from a programmer's perspective is random
  - Hashtables have O(1) lookup, O(1) insert, O(1) remove - **just like a vector!**
  - Because there is no order in a hashtable, you **cannot** use standard FOR loops (e.g. for int I = 0; I < n; i++)
    - Instead, suggest use new C++ 11 for auto loop.

Three important factors that make up a hash table

- Hashing function. Somehow, the hash table must convert a non-numeric key into an integer. This is process is called hashing. The quality of the hashing function has a direct impact on the speed of the HT.
- Collision resolution mechanism. It is (mostly) mathematically impossible to guarantee that two non-integers will not have the same hashed integer value. The collision resolution mechanism resolves collisions so that both items can exist in the same array even though they have the same hashed value.
- Load factor. A fancy term for the "fullness" of the hash table. It ranges from 0 to 1 (think 0% to 100%). Generally, higher load factors slow down the hash table (i.e. 100% load factor is slow). The ideal load

factor depends on the collision resolution mechanism.

Hashing functions

- A hashing function should ideally create very large numbers, with a reasonable spread between like keys in a very short timeframe.
- Bad hashing functions create more collisions, which slow the HT
- A hashing function that takes a really time to come up with a good number also slows the HT

A really bad hashing function

```
int keyFromString(string s){
  int hash = 0;
  for(auto ch : s){
    hash += (int) ch;
  }
  return ch;
}
```

- no spread of like data (eg "a", "b")
- lots of keys hash to same value (eg "ac" and "bb" same)
- small hash values

A slightly better example

```
int keyFromString(string s){
  int hash = 0;
  for(int I = 0; I < s.size(); i++){
    hash += ((I + 1) * (2^s[i] * s[i] * s[i] + s[i])) << 2;
  }
  return hash;
}
```