CS280 – HashMaps
March 9, 2016

http://azrael.digipen.edu/~mmead/www/Courses/CS280/Hashing-1.html

Homework Questions

- We are returning a pointer to a vector for the hashmap
- Writing code with memory leaks will get you fired!
- We aren't going to gain anything by object allocating an array since it is already contiguous, just use new for the underlying array for the hasmap

Hash Tables

- Keep in mind, we have been looking at the average case
  - Worse case could still be really bad!
- SuperFast hash is so much faster because it uses all bitwise operations
- If we know how many items we will have and what load factor, we don't have to grow the table since we can calculate its size
- Changing the table size or load factor can affect hash functions performance and collision avoidance a lot!
- Hash functions are almost entirely mathematical
- Double hashing gains better performance in general. Even good performance when we have a high load factor like 95%
  - Remember, only using the secondary hash function when we have a collision
  - Also keep in mind, use a different hash function for secondary, otherwise you will keep getting collisions and are basically just doing linear probing at that point

- Double hashing lets you save some memory by having longer insertion times
    - Another example of memory vs. speed tradeoff
- You have to look at distributions to find the worst case
    - Ask yourself if you can live with the worst case
    - Example: *Average case is 3 comparisons, but once in a while a search takes 50. Do you mind having a rare spike for overall good performance or will this spike cause issues? If it causes issues, better make the table bigger!*
- Open addressing overhead is much more expensive memory-wise because empty slots are the size of the data you are storing instead of simply being a pointer
    - However, closed addressing is not cache friendly while open addressing is
        - Yet another example of memory vs. speed tradeoff.
        - Closed addressing is more memory efficient but slower
        - Open addressing is less memory efficient but cache friendly and faster
- If you know the ID's ahead of time, you can run a hash function generator to give you a hash function that will yield **perfect** hashing (no collisions)!