CS280 Notes 3/9

by Christian Sagel

Questions:
Clear: When deleting nodes from every list, do we first iterate over all those nodes, delete the data, then when finished just clip the pointers?
HeadNodes; You provide a GetTable method, but since it's a ChHTHeadNode* it will only point at the first list, not at the whole table: *We will provide a pointer to the array.*
We only delete the table once we grow the table or the destructor gets called.
When deleting the nodes, delete the data, then the node.

AVL Tree Review
Mead is giving us 10 extra points for getting 100%.
Work on your games when you can, but don't blow your academic assignments over them! Mead was unhappy with how many people did so poorly with it. (⅔)
You cannot write code with bugs in the memory and expect not to get fired! We have been taught how to use the tools.
Historically, the first 3 assignments were the hardest.
Mead dumbed down the assignments down to BList.

For extra credit, you need perfect output.
The difficulty of the assignments varies widely depending on the student's strength.

Hash Functions and Performance
Mead wants us to be inspired to use hash tables.
With a sparsely populated table, we get very good performance.
Mead researched on what was the worst case for some of these hash functions.

Mead's revision of RS and Universal hash functions were 3 times faster.
Superfast went so fast because it was all bit-shifting.
Coding, decoding. Network packets get encrypted so often with so many packets going out so you want fast hash functions.

Open addressing (linear probing)
You may want to reserve the size of the data right away.
Mead's revised functions had less collisions than the original ones.
The SuperFast function had the least amount of comparisons too.
Throw memory at the problem to get better performance.

CS330 can be taught in a more mathematical way.
Superfast is available on the public domain. Google has their own hash function too.
Performance dramatically increases with double hashing to resolve collision.

We can’t delete with double hashing because we have to use the markdown method.
We only calculate the second hash function only if there's a collision.
With double hashing, we can have more full tables for the same performance. You pay for the extra hash but save memory.
Short tails: The data is a few hops away from the head.
With double hashing the tails were made much more reasonable.

How do we get 2 comparisons on a chain version? On average the length of the lists are gonna be 3. So we must set the load factor to 3 on a chain version to get on average 3 comparisons.
In chaining, you don’t have the long tails.
The open-addressed version brings a lot of overhead, but it has locality of reference.
Array-based structures bring nice cache performance whereas node-based don’t.