CS280 Notes 2/29

by Christian Sagel

Next assignment will be delivered on Monday. Today we will talk Hashing.

Hashing

A radically new data structure, implemented with arrays, linked lists, etc.
Most of our algorithms deal with inserting, deleting and finding items in a data structure.
Searching through arrays has complexities of O(N) and O(lg N).
Locating an item y index is O91)
Searching through binary trees has complexities O(N) and O(lg N)

What we need to be able to do is find an item without having to compare it to other items.
He has waited so long to introduce the hashing data structure because it has some glaring drawbacks. (It is not sorted). The old sets and maps were implemented with red black trees. The new ones are implemented with hash maps. He's done extensive research and written GUIs with them.

It becomes about time vs space trade-off. By adding some extra memory you can get constant-time behavior.
Another name for a hash table is associate array or dictionary.
The idea is to associate some number with data, that we use as an index to look into an array.
We look at the range of the struct and find the universal set U (range of values).
Collisions. We cannot solve the conflict problem; because they are very rare we will handle them with exceptions.
We try to pick the portions more likely to be unique.

Hash Functions

An operation that maps one large set of values into another (usually smaller) set of values.
Therefore a hash function is a many-to-one mapping. It gives constant-lookups.
A hash is a mixture of stuff, jumbling.
Mead will give us the hash functions. It is mathematical work.
In addition to throwing memory to the problem, we also randomize the data which will give us better performance. (Case in point: throwing sorted data into a binary search tree is the worst, turning it into a binary search tree).
There are hash function generators which give you hash data.

Perfect hashing gives no collisions. In practice, usually not possible since you don't know what you will be throwing into the container at runtime.
It's how you deal with the collisions that determines whether the algorithm will be effective or not.
As the data grows larger and larger, data lookups will still be constant.
A good hash function will map keys uniformly and randomly into the full range of indices.

The fastest hash function known to man that Mead wrote:
'return 0;"
He will be throwing it on the stress test and see how we deal with them,

The final exam will be similar, in format too.
There will be an assignment during Spring Break. You will get two weeks after Spring Break. (We will have 3 weeks)

Collision Resolution by Linear Probing
There are two parts to hash-based algorithms that implementations must deal with:

1. Computing the hash function to produce an index from a key.
2. Dealing with inevitable collisions.

One type of collision resolution policy is called linear probing, the basic idea is that if a slot is already occupied we simply move to the next unoccupied slot.
Mod has great characteristics. It gives us the remainder. It also wraps around.
The significance of the value 7: It is a prime number.
In hash tables, it is not about worst case or best case, but average case because you are very unlikely to get to the worst case.
Probable vs possible.
A lot of students have an issue keeping track of the probe counts. When you got hundreds of probes, missing a dozen will e.

Somewhere in your class you will have a probes++;
Make sure a function called by insert/find/delete that is the only place where probes gets incremented. Have a lookup function that just supports those 3 functions.
For the homework we will be going…
There will be a struct that holds the key, as well as the information.
Load factor = number of items in the table (slots occupied) N, divided by the size of the table (M)
If the table is nearly full, we will be spending most of our time resolving collisions. We are going to keep our eye on the load factor, and when it grows past a certain percentage we will grow it to keep our performance.
The more extra space on the table, the faster it will be and the less collisions.
The table should be no more than ⅔ full = 67%, keeping it down to 2 comparisons max.
At 90%, usually 5 probes.
Probing for an open slot handles collisions, but won't help if we run out of slots.

Collisions tend to form groups of items called clusters. We want to avoid that.
Clusters have a tendency to grow quickly (snowball effect, exponential)

We are gonna have real-world hash functions with better distribution than mod7!
Implementing hash tables are much easier than AVL trees.
They are easier to implement and will give better performance.