This week was all about hash tables. We learned how to set them up and how to retrieve data from them. Hash tables are great because they create a very quick way to access data via direct access. This is different from a linked list where each item must be iterated over until the desired element is found. Typically, a hash table is a vector of a fixed size and elements are assigned to the vector via a key. Most of the time, keys are created using a modulus function on a data member of the element that can be uniquely identifiable, such as an ID.

A collision is when an element would be assigned to a vector bucket that already contains another element. This is standard procedure when it comes to hash tables but can cause problems if not handled properly. These collisions are handled using a chaining method, where multiple elements can be stored in one bucket, or by using a linear probing method where the next empty container is found, and the element is stored there instead. These methods can get complicated and aren’t the easiest to program and I would consider that to be a weakness of hash tables.

One interesting use of hash tables that I’ve seen recently is in the realm of cryptography and cryptocurrency. There are a few projects that I’ve seen that are using hash tables to store data, but also distributing the table across several peers. So, for example, a user is looking for some element stored in a container, so he asks his peer for it. His peer does not have it, so the peer now asks his peer for the object and so on. When it is found it is passed back to the user. Pretty neat!