**Bugs or other problems**

**DiskMultiMap**

I wrote the erase function by setting the next value of the erased node to -1. However, I realized I did not fully understand the spec when I wrote this, and I should have instead created a new linked list with all the erased nodes. This makes my insert function a lot slower, because it has to go through all the spaces to find a space to insert a new node.

I also did not have time to consider collisions. To find the hash value, I used a hash function to find the key’s unique hash value and then % numBuckets.

Regardless, I think that my DiskMultiMap class works. I created a print function that would check what happens when I erase certain key/value/context sets, and it appears to be in working condition.

**IntelWeb**

I did not know how to use Terminal to test my program, and I did not have sufficient time to figure out how to do so. Hence, I was not able to test my IntelWeb implementation at all. However, it does compile – I just don’t know how well it works.

**Data structures and algorithms**

**DiskMultiMap**

Insert function

I set offset 0 to be numBuckets so that I would always have access to the number of buckets. However, I later realized this was redundant since I created a private member variable anyway m\_numBuckets that played the same role. I did not have time to rectify this.

I placed the address of the first node in the corresponding bucket of the key, and had a linked list extending from occupied buckets.

As mentioned in my problems section above, my insert function looked through the disk for an empty spot to insert a new node. If it could not find an empty spot, it would place it at the offset of the file length of the binary file.

Erase function

I implemented this very similarly to the idea of erasing a node from a linked list – I took into account if I was erasing the first node extending from a bucket, or from a bucket further down the list.

I would first find bucket that the key corresponded to, then I would check through the bucket to see if there was a key/value/context set that matched it exactly. If it matched, I would change the next value to -1 and set the previous (or the address in the bucket if it was the first linked node to the bucket) to the address of the next node.

**IntelWeb**

Crawl function

I used a set for bad entities (setBadEntitiesFound) and a set for bad interactions (setBadInteractions). This made it easier to eliminate repeat bad entities or interactions, and also made it easier to transfer all the malicious entities and interactions into the badEntitiesFound and badInteractions vectors at the end of the function.

I put all the indicators into an indicatorQueue at the beginning of the function. I used a while loop (while the queue wasn’t empty) to loop through all the indicators. I set checkingMalicious to equal to the front of the queue and popped off the front. Then I set two iterators to find the checkingMalicious in the forward and reverse files. I counted the checkingMalicious value in both files. If the count was less than the minPrevalenceToBeGood, then I would increment the maliciousNumber, insert checkingMalicious into the set of bad entities, and set two iterators to find checkingMalicious in the forward and reverse files again. This time, as I iterated through the files, I would add the corresponding value or key to the queue, and add the malicious interaction tuples to the set of bad interactions.

After the queue is empty, and all the malicious entities and interactions are in the set, I would clear the vectors of bad entities and interactions and use iterators to input the sets into the vectors.

**Big-O requirements**

Whether or not each method satisfies our big-O requirements, and if not, what you did instead and what the big-O is for your version.