1. I was unable to implement the IntelWeb::purge() method because there were bugs in the DiskMultiMap::remove method. For some reason, I was unable to read data in the remove method and the read function in the BinaryFile class would return a false flag. Other than those two methods, everything else is working correctly.
2. My disk-based data structure is quite simple. It begins with a 4 bytes containing an integer of the number of buckets. The next 4 bytes contain a pointer to a deleted node. If there more than one deleted node, the deleted nodes will be in a linked list. Then, the next bytes contain the buckets in order. Data nodes come after the buckets, and contain the three char arrays capable of holding 120 + 1 characters and a “next” pointer to create a linked list, in case there is a collision. Each data node takes 368 bytes. My insert function, first, finds the hash for the given key and gets a pointer to where the pointer to the new node should be stored. A new node is created (or replaced if there was a previously deleted node) and the values are filled. If there is a collision, the pointer to the new node is stored in the “next” variable of the last node in the linked list starting from the pointer in the hash table. My search function, first, finds the hash corresponding to the key and obtains the pointer to the node. If the node’s key doesn’t match, then it searches through the linked list if there is one. If there is no node with a matching key, then the iterator is returned in an invalid state. The “ingest” function in the IntelWeb class is one of the slower ones. It creates two files using the DiskMultiMap class. One file contains the actual order of key and value. The other file has values and keys flipped. This way the crawl function can easily calculate the prevalence of each item and check for malicious properties easily. Basically, the function opens the file and using the file stream to read data and put it into the on-disk data structure. The crawl function in IntelWeb class uses a queue which contains the known malicious entities. Firstly, the queue is filled with the known malicious files/websites. Then the search function in the nested Iterator class finds the keys and “values” which are malicious and identifies potential malicious entries. Towards the end of the loop, the potential malicious entires are counted in the TWO on-disk data structures to identify the prevalence. If the prevalence is below the threshold, the potential malicious entries are pushed onto the queue where they will be scanned again through the loop to check for more potential malicious entries. The while loop continues until he queue is empty.
3. I believe each method satisfies the big-O requirements; although, there may be a slightly more efficient way to write the crawl method. I don’t believe it will make significant speed improvements.