Report

1. Whether any of your classes have known bugs or other problems that we should know about. For example, if you didn’t finish the *IntelWeb::purge()* method or it has bugs, tell us.

I never figured out how to keep track of the offsets of my deleted nodes for later insertion. I did not notice any problems with the final implementation of my other DiskMultiMap functions, but I assume there probably are. I simply continued inserting nodes at the end of the binary file. My ingest function appears to be fine. For my crawl function, some values that should have been marked as bad entities are not. I only had time to test my purge function once. It worked perfectly the first time.

1. A high-level description of what data structures and algorithms you chose for each of your classes’ non-trivial methods, and for your disk-based data structures. Brief means <1 page of description or pseudocode per class.

Iterator(const std::string& key, DiskNode &node, BinaryFile::Offset nodesOffset,DiskMultiMap\* outer): I initialize the correlating values.

bool isValid() const: I check if my member m\_iter is set to 0, my version of null.

Iterator& operator++(): I read a node from my m\_iter. I check whether m\_next of this node is null. If not, I set m\_iter to the node’s next. Otheriwse, I set m\_iter to 0. I return a pointer to this.

MultiMapTuple operator\*(): I check if my current iterator is valid and return an empty multimaptuple if so. Otherwise, I read from my currentNode and set MultiMapTuple’s values to the currentNode’s values.

bool createNew(const std::string& filename, unsigned int numBuckets): I first check if the file is open (and close it). I then initialize my member varbiables to the input. I write my header block struct to the beginning of the binary file. For the number of buckets, I write my bucket structs tot he binary file.

bool insert(const std::string& key, const std::string& value, const std::string& context): For my “linked list”, I inserted nodes in reverse order from the hashed bucket. I first check if the size of the input is within range. I create a new DiskNode struct and copy the input values. Chose a bucket within my binary file using my hash funciton. I read that bucket’s “m\_headOfNodes”. If this is 0, my null value, I set headOfNodes to the end of the file. I then set its m\_next to 0. Otherwise, I set the newNode’s m\_next to m\_headOfNodes. I would then set my bucket’s m\_headOfNodes to the end filelenght’s offset. I would overwrite my structs.

Iterator search(const std::string& key): I would read from a hashed bucket (key). If value was zero, I’d return a null iterator. Otherwise, I Iwould read the node from the bucket’s headOfNodes and return that iterator.

int erase(const std::string& key, const std::string& value, const std::string& context): I choose a hashed bucket and read it’s offSet. If the node read at this location matches the input values, I delete it. I keep track of the previous node, setting it to null initially. If previous is still null, I know I’m at the only node in the list, and write an empty node to it. Otherwise, I read the prevoius node’s next and set it equal to the new node’s next and set that node’s next to zero. I do this until I find the value I want to delete. I return the count for how many values erased.

bool IntelWeb::createNew(const std::string& filePrefix, unsigned int maxDataItems): I create 3 DiskMultiMaps, forward, reverse, freq.

bool IntelWeb::openExisting(const std::string& filePrefix): I open these three DiskMultiMaps.

void IntelWeb::close(); I close these three DiskMultiMaps.

bool ingest(const std::string& telemetryFile): I modified the given FileInput.h. I take in my telemetry file theh same way. For each iteration of the while, I insert to my forward (value, context, and key), my reverse (context,value,key). For my freq, I iterate through forward and insert (value, string version of iteration, and empty string). I then do this again but while iterating through forward, (context, string version of iteration, and empty string).

unsigned int crawl(const std::vector<std::string>& indicators,

unsigned int minPrevalenceToBeGood,

std::vector<std::string>& badEntitiesFound,

std::vector<InteractionTuple>& badInteractions

): I prepare empty sets which will correlate to my given vectors. I clear badInteractions and push all of the indicators into a queue. While my queue is not empty, I will get the iterators form my forward and reverse maps. If this iterator is valid, I will enter a while loop. I will check if my set of bad entities already has this value, and if so, I will record the InteractionTuple and increment the iterator. I will get the frequency of the iterator’s value (second string). I will search through my frequency map to get the frequency of the this value. If this value is less than Pgood, I push the value into my queue and insert it into my set of bad entities. I will then record the Interactiontuple and increment the vector. I do the same thing for my reverse iterator while loop. After the queue is empty, I will insert my all my sets’ values back into my vectors.

bool purge(const std::string& entity): I have a purgedOnce bool set to false that will be returned as true if I ever enter the following while loops. I will create iterators for all three of my hash maps using the string value entity. While forward’s iterator is valid, I will erase from my forward map(key, value, context) and then erase from my reverse map(value, key, context). While reverse’s iterator is valid, I will erase from my reverse map(key, value, context) and then erase from my forward map(value, key, context). If purgedOnce is true, then I will erase the entity from my frequency map.

1. 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.

bool DiskMultiMap::createNew(const std::string& filename, unsigned int numBuckets): I followed the bigO requirements.

bool DiskMultiMap::openExisting(const std::string& filename): I followed the bigO requirements.

void DiskMultiMap::close(): I followed the bigO requirements.

bool DiskMultiMap::insert(const std::string& key, const std::string& value, const std::string& context): I did O(buckets) instead.

Iterator search(const std::string& key): I followed the bigO requirements.

int DiskMultiMap::erase(const std::string& key, const std::string& value, const std::string& context): I followed the bigO requirements.

Iterator(): I followed the bigO requirements.

bool isValid() const; I followed the bigO requirements.

Iterator& operator++(): I am in doubt, but I think the bigO is O(N/B).

MultiMapTuple operator\*(): I am in doubt, but I think the bigO is

bool IntelWeb::createNew(const std::string& filePrefix, unsigned int maxDataItems): I followed the bigO requirements.

bool IntelWeb::openExisting(const std::string& filePrefix): I followed the bigO requirements.

void IntelWeb::close(): I followed the bigO requirements.

bool ingest(const std::string& telemetryFile): I followed the bigO requirements.

unsigned int crawl(const std::vector<std::string>& indicators,

unsigned int minPrevalenceToBeGood,

std::vector<std::string>& badEntitiesFound,

std::vector<InteractionTuple>& badInteractions

): I am not sure about the bigO of my crawl. It might be O(L^2).

bool purge(const std::string& entity): I am not sure about he bigO of my purge. It might be O(L).