March 10, 2016 **Project 4 Report** Karen Li

**Known Bugs/Problems**

No known bugs or problems associated with any of the classes.

**Description of Data Structures & Algorithms**

*Iterator Class*

MultiMapTuple DiskMultiMap::Iterator::operator++()

Pseudocode:

If the iterator is valid

Read in the data of the current node it’s pointing to

While there’s a valid node after the current node

Read in the data of the next node

If the key in the current and next node are the same

Move the iterator over

Check the node after the next node

Make the iterator invalid if gone through all nodes and haven’t found same key

*DiskMultiMap Class*

DiskMultiMap implements a disk-based open hash table. The beginning of the disk file contains a header struct that contains the number of buckets in the table, and offsets pointing to the end of the file, the part of the file where the node data starts, and the first previously allocated memory spot that can be overwritten (list of deleted nodes). Then, the disk file contains a series of bucket structs that contain offsets pointing to a linked list of nodes. Then, the file contains the actual node data.

bool DiskMultiMap::insert(const std::string& key, const std::string& value, const std::string& context)

Pseudocode:

Hash the key to find what bucket you should insert the data in.

Get the data inside the bucket and link the new node to the first element of the list in the bucket (pushing front)

Check the header offset that points to the list of deleted nodes.

If there is a previously deleted node, write the new node there

Set the header offset to the next previously deleted node

If there aren’t any previously deleted nodes

Write the new node to the end of the file

Reinitialize the header offset that points to the end of the file

DiskMultiMap::Iterator DiskMultiMap::search(const std::string& key)

Pseuodocode:

Hash the key to find the right bucket and get its data

Loop through the elements in the linked list at that bucket

If the key of the current node matches the parameter, return an iterator pointing to the current node

Else, move to the next element in the linked list

If couldn’t find a node with the same key, return an invalid iterator

int DiskMultiMap::erase(const std::string& key, const std::string& value, const std::string& context)

Pseudocode:

Hash the key to find the right bucket and get the bucket’s data

Loop through the linked list in that bucket

Get the data from the current node

Get the data from the previous node if there is one

If you find a match

Set the node to deleted and link it to the front of the list of deleted nodes

Rewrite the header to the disk file so it contains the correct offsets

Else

If you haven’t reinitialized the bucket’s head

Initialize the bucket head to the current node

If there’s a valid previous node, link it to the current node

Move the offset pointing to the previous node over

Move the offset pointing to the current node over

If you’ve deleted everything in the list, set the bucket’s offset to invalid

*IntelWeb Class*

IntelWeb contains two DiskMultiMap objects, a forward map (which maps key, value, context) and a reverse map (which maps value, key, context), which allows for an interaction to be easily searchable using either the key or the value.

bool IntelWeb::ingest(const std::string& telemetryFile)

Pseudocode:

Open the file for input using ifstream

Read each line

Extract the info from the line into key, value, and context string variables

Insert the data into the forward map and the reverse map

unsigned int IntelWeb::crawl(const std::vector<std::string>& indicators, unsigned int minPrevalenceToBeGood, std::vector<std::string>& badEntitiesFound, std::vector<InteractionTuple>& interactions)

Pseudocode:

Clear the badEntitiesFound and interactions vector

Put all of the indicators in a checkerVector vector

Loop through the vector

For both the forward map and the reverse map

Remove the last item from the vector and find possible bad items related to it

For each possible bad item, count the number of times it occurs and if it occurs less than the minPrevalenceToBeGood, insert it into the set of possibleBadThings

If you can insert it into the set, insert it into the vector as well

Loop through all things inside of possibleBadThings to find all bad interactions and insert them into another set called setOfBadInteractions

Copy all entries from possibleBadThings and setOfBadInteractions sets to badEntitiesFound and interactions vectors

bool IntelWeb::purge(const std::string& entity)

Pseudocode:

Search for entity inside both the forward and reverse map

For each node that contains entity, save its value into a tuple

Increment the iterator before erasing the node from the map

Erase the node

Return true if you deleted at least one thing from the maps

**Big-O Requirements**

*DiskMultiMap Functions*

DiskMultiMap(); Yes, it is O(1)

~DiskMultiMap(); Yes, it is O(1)

bool createNew(const std::string& filename, unsigned int numBuckets); Yes, it’s O(B)

bool openExisting(const std::string& filename); Yes, it’s O(1)

void close(); Yes, it’s O(1)

bool insert(const std::string& key, const std::string& value, const std::string& context); Yes, it’s O(N/B)

Iterator search(const std::string& key); Yes, it’s O(N/B)

int erase(const std::string& key, const std::string& value, const std::string& context); Yes, it’s O(N/B)

*DiskMultiMap::Iterator Functions*

Iterator(); Yes, it’s O(1)

Iterator(bool valid, BinaryFile::Offset current, BinaryFile\* bfile); Yes, it runs in O(N/B)

bool isValid() const; Yes, it’s O(1)

Iterator& operator++(); Yes, it’s O(N/B)

MultiMapTuple operator\*(); Yes, O(1)

*IntelWeb Functions*

IntelWeb(); Yes, it’s O(1)

~IntelWeb(); Yes, it’s O(1)

bool createNew(const std::string& filePrefix, unsigned int maxDataItems); Yes, it’s O(maxDataItems) time

bool openExisting(const std::string& filePrefix); Yes, it’s O(1)

void close(); Yes, it’s O(1)

bool ingest(const std::string& telemetryFile); Yes, it’s O(N)

unsigned int crawl(const std::vector<std::string>& indicators, unsigned int minPrevalenceToBeGood, std::vector<std::string>& badEntitiesFound, std::vector<InteractionTuple>& interactions); Yes, it’s O(TlogT)

bool purge(const std::string& entity); Yes, it’s O(M)