1. **No discovered bugs.**

DiskMultiMap::Iterator::operator++()

Check if iterator is valid or tuple is readable, return current value if not.

Iterate through tuples, starting from next adjacent tuple.

Return current value when strcmp == 0 for t.key and currentTuple.key.

DiskMultiMap::createNew(const std::string &filename, unsigned int numBuckets)

Close file if it’s open, create and overwrite new file of filename

Create a header and initialize the array of buckets.

Return true. Return false for any failures in the above actions

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

Perform length checks on key, value and context, check if file is open/header readable

Determine where to write item to (over previously removed node or in new bucket)

Create TupleItem and input key, value and context. Update offset and header. Return true.

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

Perform length check on key, check if file is open/header readable.

Read Bucket information.

Loop through everything to read tuples. Return iterator to the TupleItem if found, otherwise return an invalid iterator.

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

Perform length check on key, check if file is open/header readable.

Read Bucket information.

Loop through everything in search of matching tuples. Keep track of no. removed.

If found, update hasbucket head, prev node, erasing node and header. Return no. removed at the end.

IntelWeb::createNew(const std::string &filePrefix, unsigned int maxDataItems)

Create new DiskMultiMaps using parameters. If success return true, else return false.

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

If files not ready or can’t read telemetry files, return false.

Get line and check if format is valid.

Using DiskMultiMap insert, insert origin, destination and context.

Return true if successfully ingested, false otherwise.

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

unsigned int minPrevalenceToBeGood,

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

std::vector<InteractionTuple> &interactions)

If file not ready, return false.

Create interactions set which creates a list of unique interactions.

Create hashmap to act as marker to indicate if the vertex has been visited.

Iterate thorugh every known bad element, compare with minPrevalenceToBeGood

Continue checking queue of bad items from the graph, compare with min prevalence.

If found new interaction, insert it and label it as a good item.

Mark as visited and push onto queue. Return size of it.

IntelWeb::prevalence(const std::string& key, int min)

If key not found in map, calculate all the interactions where it is pointing.

Cache this and continue iterating thorugh all the interactions.

Return number of instances of this particular interaction.

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

If file not ready, return false.

Remove from cache the entity, then get all other related references in the different tuple arrangements. (key, value, context or value, key, context)

Remove entity context origin and destination from diskfiles without using default iterator. Return true after removal, otherwise false.

**3. Methods satisfy big O requirements.**