Nathan Tung
Com Sci 32 Dis 2C
March 13, 2012

CS32 Project 4 Report

1. **Describe the algorithm and data structures you used for your:**
   a. **AdHunterImpl class: Describe the data structures and algorithms you used to efficiently construct, order, and return your ad target match list to the caller.**
   →My AdHunterImpl class has a private dynamic Matcher object, along with a vector for storing seed site strings. On construction of the class, the Matcher is initialized with the ruleStream parameter. A site is pushed back into the string vector each time the addSeedSite function is called.

   The getBestAdTargets clears the passed in Match vector. Then, using a for-loop, for each seed site in the string vector, a Crawler is dynamically created with that seed to continuously crawl and return Document pointers until the maximum count of pages per site is reached or a returned pointer is NULL. In each cycle, Crawler is deleted to prevent memory leak. Each Document pointer returned by the Crawler is passed into the private Matcher's process method, which adds accordingly to the vector of Matches. Finally, AdHunterImpl calls the <algorithm> STL sort using a simple sort-Comparer method to determine which of two Matches comes first. Matches size is then returned, and AdHunterImpl completes one such lifecycle.

   b. **MatcherImpl class: Describe the data structures and algorithms you used to efficiently find the matches in a document.**
   →MatcherImpl has a private vector of pointers to Rules, which consist of all rules taken in via getline and ruleStream in the constructor. In the main process function, another vector of pointers to Rules is created, but this one is purely for those that are "applicable" to the Document on hand. We cycle through all the rules via a for-loop, and if any elements in that rule are found in the Document, we push it back into the private Rule vector. This concludes determining which Rules need to be checked for.

   The process function then iterates using a for-loop through all applicable rules. For each rule, if it returns a true for the match function and the rule's value is at least the minimum price specified in the parameter, we will construct a temporary Match with all corresponding values. The Match object is then pushed into the vector within the parameter, completing MatcherImpl's delegated task.

   c. **MyHashMap class: Describe the data structures and algorithms you used to implement your hash table mapper class.**
   →My version of MyHashMap has a pointer to a dynamic array of node pointers; its size is specified in the MyHashMap constructor and saved as the number of buckets. This class also contains three node pointers: head, tail, and curr. The head and tail pointers initialize to NULL, while curr initializes to head only when getFirst is run. The array consists of buckets of linked-lists (sorted by hash codes), while the head, tail, and curr pointers make up a second linked-list of pointers linking back to the SAME nodes in the array. This second linked-list makes it easier to call the destructor or fetch a random key.

   My hashCode method is calculated by adding the ASCII values of all characters in the parameter string, multiplied by the index of each character plus one. The final returned hash code undergoes a modulus operation over the number of buckets in the class.

The associate function can run into three possibilities. If the bucket being looked at is empty, then that pointer points to the new node, which is also appended to the second linked-list. If the bucket has something, we go through all the pointers until we either find a matching key or until the pointers run out. If no match is found, we append the node pointer to the end of the bucket as well as the end of the second-linked list. If a match is found, we simply update its value without adding any objects.

2. Provide a thorough list of test cases that you used to test each of your classes and note what each test tested.

MyHashMap:

```
MyHashMap<bool> map;
assert(map.find("stupid")==NULL); //assert(*map.find("stupid")==NULL); would dereference NULL
map.associate("blah", true); //try inserting an arbitrary key
map.associate("asldjfkla", true); //try inserting a second arbitrary key
map.associate("asdf", true); //try inserting a third arbitrary key

assert(*map.find("asdf")==true); //fetch and verify the value of a key
map.associate("asdf", false); //try changing the value of a key
assert(*map.find("asdf")==false); //fetch and verify the value changed

map.associate("zzzz", true); //insert key with same code as below, 1220
map.associate("dPPPP", false); //insert key with same code as above, 1220
assert(*map.find("zzzz")==true);
assert(*map.find("dPPPP")==false); // verify their values

map.associate("zzzz", false);
map.associate("dPPPP", true); //change the values of the two keys
assert(*map.find("zzzz")==false);
assert(*map.find("dPPPP")==true); //see if their corresponding values changed

map.associate("PZPPP", true); //insert key with same code as zzzz and dPPPP (1220)
assert(map.numItems()==6); //verify number of items

MyHashMap<int> map2;
map2.associate("one", 1);
string tKey;
assert(*map2.getFirst(tKey)==1 && tKey=="one"); //check if getFirst and key correspond
assert(map2.getNext(tKey)==NULL); //make sure a NULL is returned if no more elements
assert(map2.numItems()==1); //verify number of items
```

ExtractLinks:

```
string text;
int count=0;
if(HTTP().get("www.google.com", text))
{
	ExtractLinks* e = new ExtractLinks(text);
	string newLink;
	while(e->getNextLink(newLink)) //extract all the links from text
	{
		cerr << newLink << endl; //manually check if the links match
		count++; //verify number of links found
	}
	delete e;
	cerr << count << " links were extracted!" << endl;
}
```

```
if(HTTP().get("www.yahoo.com", text)) //yahoo has several URLs that are literally #
{
	ExtractLinks* e = new ExtractLinks(text);
	string newLink;
	while(e->getNextLink(newLink)) //extract all the links from text
	{
		cerr << newLink << endl; //manually check if the links match
		count++; //verify number of links found
	}
	delete e;
	cerr << count << " links were extracted!" << endl;
}
```

Document:

```
string text;
int count=0;

if(HTTP().get("www.google.com", text)) //check if a text is fetched
{
	Document* d = new Document("www.google.com", text); //try to construct Document
	assert(d->getURL()=="www.google.com"); //verify that the saved URLs match

	string aWord;
	if(d->getFirstWord(aWord)) //try to get Document's first word
	{
		count++;
		cerr << aWord << endl; //output that word and manually check it
	}

	while(d->getNextWord(aWord)) //get the rest of Document's words
	{
		count++;
		cerr << aWord << endl; //output all of the words to check
	}
	cerr << count << " words were looked over!" << endl; //show number of words displayed
}
```

Crawler:

```
int count=1;
Crawler* c = new Crawler("www.google.com"); //construct a Crawler on Google
Document* d = c->crawl(); //Get the first Document from Crawler

string aWord; //display all words for the FIRST DOCUMENT ONLY (make sure get word function
works)
	if(d->getFirstWord(aWord)) //try to get Document's first word
		cerr << aWord << endl; //output that word and manually check it
	while(d->getNextWord(aWord)) //get the rest of Document's words
		cerr << aWord << endl; //output all of the words to check

while(d!=NULL)
{
	count++;
	cerr << d->getURL() << endl; //output all the crawled URLs via Document order
	d=c->crawl(); //keep crawling until all links have been displayed
}
cerr << "We crawled " << count << " times!" << endl; //there's a LOT OF LINKS!
```

Nathan Tung
Com Sci 32 Dis 2C
March 13, 2012

Rule:

```
Rule* r1 = new Rule("rule-00000000 0.50 BLAH HeY tHeRE & & > Do I work?");
assert(r1->getAd()=="Do I work?"); //Check if the ad is exactly equal
assert(r1->getDollarValue()==0.50); //Check for dollar value
assert(r1->getNumElements()==5); //Check if counter is correct; includes words and operators
assert(r1->getName()=="rule-00000000"); //verify that the name is exact
assert(r1->getElement(0)=="blah"); //test for case; everything should become lowercase
assert(r1->getElement(1)=="hey");
assert(r1->getElement(2)!="thEre");
assert(r1->getElement(r1->getNumElements()-1)=="&"); //verify both functions work

Rule* r2 = new Rule("rule-00000000  0.50 BLAH HeY | tHeRE & & > Do I work?"); //improper rule!
Rule* r3 = new Rule("rule-00000000  0.50 BLAH & HeY  tHeRE & > Do I work?"); //improper rule!

assert(r2->getNumElements()==0 && r3->getNumElements()==0); //must show that they are improper!

Document* d = new Document("www.nonexistentwebsite123abc.com", "blah hey there this should be
true");
Document* d2 = new Document("www.thisonedoesntexisteither.com", "blah hey this is false");

assert(r1->match(*d)==true); //the first document matches!
assert(r1->match(*d2)==false); //the second one NEARLY matches, but doesn't
```

Matcher:

```
Rule* r1 = new Rule("rule-00000000 0.50 BLAH HeY tHeRE & & > Do I work?");
Document* d = new Document("www.nonexistentwebsite123abc.com", "blah hey there this should be
true");
Document* d2 = new Document("www.thisonedoesntexisteither.com", "blah hey this is false");
string r1Text="r1Text.txt"; //holds r1

ifstream infile(r1Text);
if (!infile)
{
	cerr << "Problem!" << endl;
	exit(1);
}

Matcher* m = new Matcher(infile);
vector<Match> matchV;
m->process(*d, 0.10, matchV); //should match
m->process(*d2, 0.10, matchV); //should NOT match%

assert(matchV.size()==1); //make sure there's only one match
cerr << matchV.at(0).url << endl; //see if the URL matches
```

AdHunter:

```
string cText="cnnText.txt";
ifstream infile(cText);
if (!infile)
{
	cerr << "Problem!" << endl;
	exit(1);
}
AdHunter* adKiller = new AdHunter(infile);
adKiller->addSeedSite("www.cnn.com");
adKiller->addSeedSite("www.msn.com");

vector<Match> matchV;
adKiller->getBestAdTargets(0.10, 1, matchV); //there should be one match!
```

```
assert(matchV.size()==1); //verify the single match
cerr << matchV.at(0).url << endl; //see if the URL matches
cerr << matchV.at(0).ad << endl; //see if the ad matches
```

The main function can also be tested by using the command prompt and entering invalid file names for the rules or sites list. In addition, extra parameters will cause it to exit prematurely and/or return an error message accordingly.

3. **Give a list of any known bugs or issues with your program.**
   →To my knowledge, my program works as intended.

   Initially, however, my program had some problems with MyHashMap (particularly with dereferencing or accessing NULL pointers), as well as the stack in Rule for determining matches. All of those problems should have been fixed.