Tracy Yu

Project 4

Report

1.

AdHunterImpl class: In this class, my constructor takes in the input files and passes the input files to the MatcherImpl class, which then sorts out the rules in the file as valid or not valid. From there, the Matcher places the rules in a vector. Within the AdHunterImpl class, there is a vector that holds the url of the websites, which is added by the addSeedSite() method. The algorithm I used for the getBestAdTargets() function is first clear any existing matches in the match vector then go through the vector of websites, and crawl through each one. Ever website it crawls through, it matches price and returns the matches of rules that correspond to the webpage. Then it counts up the total matches. After, it sorts the matches from smallest to largest and reverses the order. I had to create a operator overload for < because I want to compare to matches by dollarvalue, then url, and then ruleName if the previous two variables are equal.

MatcherImpl class: The data structure used to efficiently find the matches in the documents is using vectors. The vector of all the rules are sent to another member function called findTheSubsetOfApplicableRulesThatApply, which returns a vector of Rule pointers that point to rules that contain at least one key word in the document. This narrows, down the list of Rules. Therefore, when going to the process function, the for loops only loops through the applicableRules vector and compare the match in minPrice and with the postfix equation in each rule.

MyHashMap class: In this class, I created an array that contains pointers to Nodes for the singly linked list. Within each node, there is a key and value, pointer to next node in each element of the array, and another node called second\_next, which points to the previous node that has been created. This makes it simpler to find a node (key/pair) and also for the other functions getFirst and getNext. Both these function follows the second\_next and the prev pointer. The prev pointer points to the last node that has been created by the associate() method. Also, when hashing the key into the map, I used the fnv-1a hash function to equally, randomly distribute the keys throughout the map, to possibly reduce the amount of collisions.

2. List of test cases:

MyHashMap:

Test cases: MyHashMap<int> item;

Item.associate( “food”, 10); // test creating a node in an empty map

Item.associate(“chair”, 4); // testing creating a node in a map with more than one item

Item.assoicate(“frog, 5);

Item.associate(“food, 2); // this test that the MyHashMap should not create another node, but rather just change the value;

String key;

Item.getFirst(key); // should return int pointer to food/2 pair

Item.getNext(key); // should return another int pointer pointing to another key/value pair

numItems(); //should return 3;

item.associate(“cat”, 0);

getNext(key); // should return NULL, because must call getFirst before getNext when new pair is added.

ExtractLinks class:

std::string myWebPage = “<html><head><title>Introduction to HTML class.”</title> ……<a href=<http://www.wikipedia.com/wiki/spam>>spam</a>

ExtractLinks p(myWebPage); // this creates a class that holds the entire webpage

p.getNextLink(myWebPage); // this should return true, and set myWebPage to <ahref=”<http://www.wikipedia.com/wiki/spam>”

p.getNextLink(myWebPage); // should return false, because no more webpage to copy

Document class:

myWebPage = “<http://www.wikipedia.com/wiki/spam>”

body = <html><head><title>Introduction to HTML</title></head><body><b>This is a web page!</b> <br/>If you want to learn more about HTML, click <ahref="http://www.wikipedia.com/wiki/html">here</a>! <br/

Document k(myWebPage, body); // creates a document k, which all of the unique words int eh document is stored in a hash map

k.getURL(); // returns the url of the document

k.contains( “learn”); // returns true

k.contains(“John”); //return false because it is not in the document

string word;

k.getFirstWord(word); // return true, and returns a word from the document “html”

k.getNextWord9word); // returns true, and word is equal to “head”

Crawler class:

Crawler cr(“<http://www.wikipedia.org/wiki/DNA>”); // creates a crawler object with Wikipedia page

for (;;)

{

Document\* curDoc = cr.crawl();

if (curDoc == NULL)

break;

cout << "Got a valid document object for URL: "

<< curDoc->getURL() << endl; // expected to return a list of webpages starting from the first link it finds on the entire page and travels through the links to find other links

Prints:

“http://www.wikipedia.org/wiki/DNA”

“http://www.wikipedia.org/wiki/Introduction\_to\_genetics”

“http://www.wikipedia.org/wiki/DNA\_disambiguation”

“http://www.wikipedia.org/wiki/Nucleic\_Acid”

Rule class:

Ruletext =rule-00000017 0.32 APPEAL SANITORIA | MAKEUPS | > Click here to learn more about SANITORIA

Text = rule-00000268 0.89 OVERWORKED KIMONOS & > Buy OVERWORKED now and save!

Rule r(Ruletext); // forms a rule class, but since missing the ruleName, test if rule is set

is set up properly

r.getNumElements(); // should return 0;

Rule s(text);

s.getName(); // should return rule-00000268

s.getDollarValue(); // should return 0.89

s.getNumElements(); // should return 3

s.for( int k= 0; k <s.getNumElements(); k++) // should return “OVERWORKEDKIMONOS&”

(cout <<s.getElement(k);)

s.getAd(); // should return “Buy OVERWORKED now and save!”

s.match(k); //This return true if rules that matches the document, else false; it returns false

Matcher class:

Matcher p(ruleStream); // this will take in an input file from the user

Vector<Match> matches;

p.process( k, 1.20, matches); // this will process through all the rules and create a vector of matches that follow the postfix equation and >= the minPrice

if the Document included: “ The makeup works on everybody, no creatures can say no. hee”

p.process(k, 0.6, matches) // should return only rule-000000269 because it satisfy both requirements

rule-00000017 0.32 APPEAL SANITORIA | MAKEUPS | > Click here to learn more about SANITORIA

rule-0000025a 0.89 OVERWORKED KIMONOS & > Buy OVERWORKED now and save!

rule-00000268 0.58 HUMIDIFIES > Save 10% on HUMIDIFIES if you buy today!

rule-00000269 0.70 CREATURES HEE COLLAGEN & | > Curious about CREATURES? Click here to learn more!

AdHunter class:

int main( int argv, char\*argv[]){

std::string rule((LPCTSTR)\*argv[1]);

ifstream ruleinfile(rule.c\_str());

AdHunter p(ruleinfile); // this takes in the rule file and takes out any invalid rule

p.addSeedSite(<http://yahoo.com>); // this adds a url to the vector of websites in AdHunter object

vector<Match> matches);

getBestAdTargets(0.30, 10, matches); // returns number of matches that the rules coincide with the seedSite and its sub\_sites, also returns a vector of the matches

}

3. I was unable to check for every case of whether the rule is valid in the constructor, especially when checking if the dollarValue is valid. In addition, I was unable to finish implementing the main function.