NewsAggregatorImpl Class

class NewsAggregatorImpl

{

public:

NewsAggregatorImpl();

~NewsAggregatorImpl();

void addSourceRSSFeed(string feed);

int getTopStoriesAndKeywords(double thresholdPercentage,

vector<Cluster>& topStories, vector<Keyword>& topKeywords);

private:

void kernelCluster();

bool submitCluster(NewsCluster\* nc1, NewsCluster\* nc2);

void repeatSubmitCluster( bool& redo);

bool findDuplicates();

void topStoriesCluster(vector<Cluster>& topStories, double p);

void findTopKeywords(vector<Keyword>& topKeywords, double p);

vector<NewsCluster\*> v;

vector<RSSProcessor\* > m\_rssVector

};

1). The constructor for NewsAggregatorImpl class clears vector <NewsCluster\*> and vector <RSSProcessor\*>

2). The destructor is responsible for deleting dynamic memory allocated to the vector <NewsCluster\*> and vector <RSSProcessor\*>

3). addSourceRSSFeed function creates a dynamic RSSProcessor variable and if the call RSSProcessor.getData() return true, push the RSSProcessor variable to the vector of RSSProcessor\* container. If returned false, delete the allotted dynamic memory.

The below are helper function to assist getTopStoriesAndKeywords function:

4). KernelCluster function is responsible for creating a kernel NewsCluster for each distinct headline story. With each successful kernel cluster created, push this kernel cluster to the vector of <NewsCluster\*>. After we created a kernel cluster for each distinct headline, delete the dynamic memory allocated to the vector of RSSProcessor\* since we no longer need this data.

5). findTopKeywords function accepts two argument : the topKeywords vector and the thresholdPercentage. This function uses three STLs containers to find the top keywords.

i). The first STL container is the StringMapper<Keyword>, this containers maps a single “word” to a Keyword, coincidentally the Keyword.keyword is the same word as the first parameter mapped in the StringMapper. This allows for a quick search of the word when used for sorting. This container will effectively contain a BST of all the words and the number of counts it appear in each distinct headline.

ii). The second STL container is a set container of Keywords, this STL will automatically sort the Keyword by the Keyword.numUses, from least to greatest. An item is inserted to the this container container only if it satisfies the thresholdPercentage limitation.

iii). The third STL container is a queue container of Keywords, this is use because of the functionality of FIFO. The last element in the set container of Keywords will be push to the queue container and continue until after the first element in the vector of Keywords is pushed to the queue container.

Then we can pop each item in the queue and inserted to the topKeywords vector, this will ensure that the word with highest count is at the top of the topKeywords vector.

6). repeatSubmitCluster function accept a boolean argument by reference. This function utilizes the clustering algorithm describe in the specification. It calls another helper function submitCluster to submit one NewsCluster to another NewsCluster variable. The submitCluster will return true if the call to submitStory() of the second NewsCluster variable returns true. This will goto a loop until the beginning while loop is set to false (ie. The Boolean argument passed to repeatSubmitCluster).

7). findDuplicates function will erase the extra cluster and return true if a identical cluster is found in the vector of NewsCluster. The function will return false if all cluster’s getIdentifier() function returns a distinct string.

8). topStoriesCluster function uses 3 STL container to find the cluster with the url count.

i). The first STL container used is a set of Keyword to hold the headline title and the number of related URLs the NewsCluster contains. This will automatically sort the number of URL count from least to greatest.

ii). The second STL container used is a vector of Cluster; this is used to hold a cluster of headline along with its related URLs. This is a temporary vector used for sorting. Since Cluster constructor accepts a argument, we cannot use StringMapper to map a string with a cluster, nor can we use a set of Cluster since there is not public variables to compare one Cluster with another.

iii). The third STL container used is a queue of Cluster; this is mainly use for sorting. Using the first STL (set of Keywords which is sorted in order of most URL counts), we can find the associated Cluster in the second STL container and push it to the queue. At the very end, we just need to pop each queue item and inserted to the topStories vector which will be in order of most URL count .

NewsCluster Class

struct clusterInfo

{

string keywords;

string title;

string url;

clusterInfo(const string& keyword, const string& strTitle, const string& strUrl)

: keywords(keyword), title(strTitle), url(strUrl) {}

};

bool operator<(const clusterInfo& cur, const clusterInfo& other)

{

return (cur.title < other.title);

}

class NewsClusterImpl

{

public:

NewsClusterImpl();

~NewsClusterImpl();

bool submitKernelStory(string headline, string url);

bool submitStory(string headline, string url);

string getIdentifier() const;

bool getFirstNewsItem(string& headline, string& url);

bool getNextNewsItem(string& headline, string& url);

int size() const;

private:

string keepImportant(string headline) const;

vector <clusterInfo>::iterator ptr; //iterator pointer

vector <clusterInfo> m\_rssCluster; //vector [0] holds the kernel story

};

1). The NewsClusterImpl class uses one STLs, and a associated vector iterator pointer for retrieving a NewsClusterImpl Object. This STL is a vector of clusterInfo which contains 3 string members and a defined constructor along with a < operator for sorting purpose.

2). The constructor and destructor clears the vector of clusterInfo.

3). The keepImportant function parse a headline title to keep words of character length greater than 4 and return a string back.

4). The submitKernel function basically submits a headline title after calling the keepImportant function and a url to a NewsCluster, this will always be the zeroth element in the vector of rssInfo.

5). submitStory function will first check if the cluster has matching url within the same cluster. If found matching url, then do not submit story again. Then this will parse each word and keep a count of the number of matching words for two headlines. If more than 3 words matching is found, inserted it to our current NewsCluster,; otherwise, check other headlines within the NewsCluster.

Returns true if a headline is submitted to the current NewsCluster.

6). getIdentifier function returns a concatenated string sorted in order of all the headline titles in the current NewsCluster. The algorithm to do this is first create a temporary vector of clusterInfo and then sort it, we do this so that our kernel headline story is not modified from its zeroth element position.

7). The getFirstNewsItem and getNextNewsItem uses the vector iterator and return the headline and url.

StringMapper Class

struct bst\_key\_node

{

std::string data; bst\_key\_node \*left;

bst\_key\_node \*right;

bst\_key\_node \*nextItem;

T\* mapped\_data;

}

bst\_key\_node\* root\_ptr; //head pointer of the BST

bst\_key\_node\* prev\_add; //points to the previously insert node

bst\_key\_node\* iterator; //iterator to iterate through all nodes

int m\_map\_size; //size of the BST

1).

Insert\_To\_BST( node\* ptr, ItemType val)

{

If the treePtr is NULL

Allocate a new node and put val into it

Set new node previous\_add, left, and right node to NULL

Point the tree pointer to new node.

Create a temporary pointer pointing to where treePtr points

while the temporary pointer is not null

{

if the val equal to ptr->value, do nothing

else if the val is less than ptr->value

further if the ptr->left is null

create a new node

set ptr->left to new node

set previous\_add ->nextItem to new node

set previous\_add to new node

otherwise

set ptr to point to ptr->left

else

if ptr->right is null

create a new node

set ptr->right to new node

set previous\_add ->nextItem to new node

set previous\_add to new node

otherwise

set ptr to point to ptr->right

}

increment map size

}

2). Find function searches the binary search tree to look for the string value, and compare if the value is smaller, set pointer to left; child if string value is greater then set pointer to right child; if the string value is equal then the string is found and return the mapped variable T

3). getFirstPair and getNextPair uses the nextItem pointer to point from the first inserted node to the last inserted node.

Problem not fixed or implemented

The program runs well but does not handle the case when a top story cluster contains the same URL count as another cluster in which case we need to alphabetically sort the cluster.

The program also does not handle the case when a top Keyword count contains another keyword with the same keyword count in which we must sort it alphabetically.

/\* Test for Mapper.H

==============================================

int main() {

StringMapper<int> tester;

tester.insert("Simon",12);

cout<<"Instered, size is now: "<<tester.size()<<endl;

tester.insert("Phillip", 7);

cout<<"Instered, size is now: "<<tester.size()<<endl;

tester.insert("Brian",8);

cout<<"Instered, size is now: "<<tester.size()<<endl;

tester.insert("Jimmy", 3);

cout<<"Instered, size is now: "<<tester.size()<<endl;

tester.insert("Rickey", 3);

cout<<"Instered, size is now: "<<tester.size()<<endl;

tester.insert("David", 11);

cout<<"Instered, size is now: "<<tester.size()<<endl;

int month;

string name;

bool gotPair = tester.getFirstPair(name,month);

while(gotPair){

cout<<name<<" is borned in the month: "<<month<<endl;

gotPair = tester.getNextPair(name,month);

}

cout<<"Finished Printing all pairs "<<endl;

cout<<"Now testing insertion overwriting with insert David with David, 99"<<endl;

tester.insert("David",99);

cout<<"Inserted duplicate, size is now: "<<tester.size()<<endl;

gotPair = tester.getFirstPair(name,month);

while(gotPair){

cout<<name<<" is borned in the month: "<<month<<endl;

gotPair =tester.getNextPair(name,month);

}

int A = 0;

cout<<"Finished Printing all pairs "<<endl;

cout<<"Testing find(\"Jimmy\"), A"<<endl;

tester.find("Jimmy",A);

cout<<"A is: "<<A<<endl;

cout<<"Size is "<<tester.size()<<endl;

/\* Test for RSSPROCESSOR.cpp

=======================================================

int main(){

string url;

string headline;

RSSProcessor testLink("http://feeds.latimes.com/latimes/news?format=xml");

testLink.getData();

bool gotItem = testLink.getFirstItem(url,headline);

while(gotItem){

cout<<"Headline is: "<<headline<<endl<<endl<<"Link is: "<<url<<endl<<endl;

gotItem= testLink.getNextItem(url,headline);

}

}

/\*Test for NewsClust

===========================================================\*/

int main(){

NewsCluster testClust;

testClust.submitKernelStory("green cheese found on the moon", "URL::green cheese found on the moon");

testClust.submitStory("green eggs and ham found to annoy astronauts", "URL::green eggs and ham found to anony astronauts");

testClust.submitStory("green astronauts eat cheese on the moon", "URL::green astronauts eat cheese on the moon");

testClust.submitStory("green food coloring annoy moon astronauts", "URL::green food coloring annoy moon astronauts");

testClust.submitStory("super green hulk found cheese and loves the color green", "URL::green astronauts eat cheese on the moon");

testClust.submitStory("green food coloring annoy moon astronauts", "URL::green food coloring annoy moon astronauts"); // Duplicate headlines doesn't count more than once

testClust.submitStory("green eggs and ham found to annoy astronauts", "URL::green eggs and ham found to anony astronauts"); //Order in which this headline is insterted matters;

testClust.submitStory("moon moon found on the moon","URL:: WONT SHOW UP"); // Same words don't count more than once

testClust.submitStory("Green Cheese Found On The Moon","URL:: WONT SHOW UP"); // Case must match

testClust.submitStory("eat on the ","URL:: WONT SHOW UP"); //Words lower than 3 letters won't count

cout<<"Cluster size is: "<<testClust.size()<<endl;

string headline;

string url;

bool gotClustItem = testClust.getFirstNewsItem(headline, url);

while( gotClustItem){

cout<<"Headline is: "<<headline<<endl<<endl;

// cout<<url<<endl<<endl;

gotClustItem = testClust.getNextNewsItem(headline,url);

}

cout<<"getIdentifier() test: "<<testClust.getIdentifier()<<endl;

}