Group 14

Data structure and Algorithms for the Final project

+ Data structure: We choose Trie as our main data structure to implement this project because it is an efficient information retrieval data structure, as search complexities can be brought to optimal limit, which is O(key length). We can also search the key in O(M) time (M is maximum string length)

We also need to use a dynamic allocated 2D array to store the Trie of each file (a[group][file])

+ Algorithms:

A node will contain these things:

- an array of children (0 to 9 is numbers, 10 to 35 is letters, 36 is space, 37 is dot(.), 38 is dollar($), 39 is percentage(%), 40 is number sign(#), 41 is dash (-)) as the links, a bool-type isLeaf (to mark a word), vector <int> order (to mark the order in the text file for exact match and wildcard searching operators), bool isTitle (to mark the word in the title)

Parsing documents: a line in newspaper sometimes contain non-ascii and unwanted special letters, for example (in 700 files):

- *…and vowed to 'drive out this evil' after she met…*

- *2.20pm*

- *PUBLISHED: 08:14 BST, 20 June 2017 | UPDATED: 10:59 BST, 20 June 2017*

…

To make things simple, we implement bool isNumber(char key) function to return whether a character is a number or not.

The function SentenceFilter is to “filt” the sentence, lowercase all the characters using ASCII transformation, delete all non-ascii and unwanted special characters, return the modified sentence to insert into the Trie. The pseudocode will be:

```
string SentenceFilter(string sen)
for (i = 0 => i = length of sentence)
        if it is not an accepted character do
                if it is a dot (.), check whether it is a decimal numbers case(left and right is
an integer); if it is true, append it into the resulted string.
                else, discard it
        else append the considered character into the resulted string.
Return the resulted (filted) string.
```

We also need a function to check if a character is an accepted one or not, convert it if possible. The cases will be:

```
bool accepted (char &key)
```

```
if it is an uppercase character, lowercase it (by plus 32), then return true.
if it is a lowercase character or an integer, return true;
if it is an enter (\n), single quote (‘), replace by a blank then return true; (1)
if it is a long-dash (--), replace by a dark (-) then return true;
if it is a blank, dollar ($), percentage (%), sharp (#), dash (-) return true; (2)
```

(1) (for example, we would encounter the word like: *Trump’s resident…, smog-eating forest city,* also found that in some sentences, the writers did not put the end dot…)

(2) (for the search engine operators)

Inserting data:

We made the function NumberToString to convert a number to string type using sstream.

We implement the InputFile function in order to open each file and for simple, read the file sentence by sentence.

To read 750 files, we use two “for” loops to obtain:

```
for (i =1 => i=15)
        for (j=1 => j=50)
        convert i, j to char
        if (i<10 && j<10) open(Group0+i+_0+j);
        else if (i<10) open(Group0+i+_j);
        else if (i>=10 && j<10) open(Group+i+_0+j);
        else open(Group+i+_+j);
        string get; getline(file,get); //reading a sentence
        Split it by word (filt the sentence and use sstream) then put it into the Trie for title
        Split it by word (filt the sentence and use sstream) then put it into the Trie for
searching.
        while (not end of file) read a sentence, filt, split, then put it into the Trie for
searching;
        close file;
```

But it has the four cases that need to be considered, for example, Group01_01, Group01_12, Group14_01, Group14_12. We would also run other two loops for the extensive 100 files of group 10 and group 14, also extra data files.

If the order of the group/file is smaller than 10, we will need a zero in front of it.

We also need a function to convert number to string to open the file properly (since i and j are integers). A simple way is to use sstream. (ss << number then return ss.str())

In each file case, we read each sentence. First pass the sentence through the “filter”, then we use stringstream to get each single word from the resulted string, put that word into the Trie of that file. The pseudocode for splitting will be:

```
Split(root, sentence, start)//start: mark the position of single word
        Filt the sentence
        if (root == NULL) return;
        put the sentence into the sstream ss;
        while (ss >> sen)

                if true; continue;
```

```
        insert the word into the Trie
        start = start + 1;
```

Next, the function to insert each single word into the Trie is needed. Every character of the word will be inserted as a node. Children is an array of pointers to next node (the node does not contain the character). If the word is not in the Trie or an extension of one existing word, we need to make new nodes of the key and mark leaf node. We need another variable to determine the depth of the Trie. Also, plus 1 to the occurrence of the word for document scoring feature and push the place where the word stands in the document into the vector. The pseudocode will be:

```
InputTrie(root, word, occurrence place, bool title)
int level, length = length of word, index;
TrieNode* cur = root;
for (level = 0 => level = length – 1) do the following
        convert the key to the index order using the above function
        if (cur -> children[index] == NULL) create cur->children[index]
        move down (cur = cur -> children[index])
mark cur as leaf
if (title is true) isTitle = true;
plus 1 to cur->occurrence;
push the occurrence place into the vector;
```

Searching:

Fundamentally, we need a function to search a single keyword. (before that, we must check whether that keyword is in the stopword Trie or not)

We build up a function to index each character ((0 to 9 is numbers, 10 to 35 is letters, 36 is space, 37 is dot(.), 38 is dollar($), 39 is percentage(%), 40 is sharp(#), 41 is dash(-), -1 for invalid character)

To search a word in the Trie, we compare the characters and move down. The seach can be stopped due to end of string or lack of key in the Trie. If the "value" field of last node is non-zero then the keys exist in the Trie, or the case, the search ends without examining all the characters in the key because the key is not in the Trie. The pseudocode will be:

```
searchTrie(root, key, title)
create a TrieNode pointer variable named cur, place cur = root;
for (int i = 0 => i = length of word – 1) do the following
        convert the key to the index order using the above function
        if the indexing function return -1, continue.
        if (cur -> children[index] == NULL), return NULL (which means the word is not
exist in the Trie)
        else move down (cur=cur->children[index]);
check if cur is not NULL
if (title is true && cur->isTitle is false) return NULL;
if (isLeaf is true) return cur;
if not, return NULL;
```

Analyse the Query:

(Our search engine supports 11/12 types of query, and 3 items in the bonus part)

Normally, if no special operator is between two keywords, we consider it as AND operator.

For example, *"please sit"* and *"please AND sit"* are the same.

We mainly use stringstream to get each individual word. In the following function, all type of queries that are supported will be considered in the following order.

while (ss>>tmp) do the following algorithms:

Check if the first letter of tmp is "

1) exact match and wildcards

Check every pair of adjacent words. We denote tmp1 and tmp2 as two adjacent words. We will check whether there are one occurrence place of tmp2 and one occurrence place of tmp1 such that tmp2.occur – tmp1.occur = 1. If there is a wildcard between them (tmp1 * tmp2 or tmp1 * * tmp2), we count the number of asterisk * (cnt) then check (tmp2.occur – tmp1.occur = cnt). Else, return false. If true, get all the satisfied occurrence place and put it into the output vector.

For example: "manchester united team"

"manchester" appears in these positions: 4, 12, 20

"united" appears in these positions: 5, 16, 21

"team" appears in these positions: 6, 17

We check "manchester united" firstly, out1 = {4,12,20}, out2 = {5,16,21} we find out these satisfied cases: 4-5, 20-21, so out1 = {4,20}, out2 = {5,21}. Then tmp1 = tmp2, also try not to change the pointer in the Trie.

We check "united team" with out1 = {5,21}, out2 = {6,17} firstly. Then we find out 5-6. So the exact match appears in that document, also add the satisfied elements into the output vector.

Check if the first letter of tmp is –

2) minus sign

If the word after the minus sign is in the Trie, return false; else continue;

Get the first 8 letters of tmp to check if it is "intitle:"

3) intitle:

Get the remaining letters after "intitle:"

Search the remaining word in the Trie, if (isTitle is false), return false; else continue;

Check if tmp is OR

Make new variable: TrieNode* searchRes;

4) OR

get another tmp from sstream

searchRes = search tmp in the Trie

if tmp is found: get union the occurrence place of it with the output vector, plus the score.

continue;

(Note that we do not return false in this case. For example, we search for the query: A OR B. If A is found, then we do not need to find B, but for the ranking, we should consider B afterward)

searchRes=search tmp in the Trie

check if the first letter of tmp is +

5) Plus sign (+): force the engine to return common words (like stopwords) that might be ordinarily discarded.

Get the remaining letters after the plus sign.

if the remaining word is not found in the Trie, return false;

get union of its occurrence place. Plus the score by its occurrence vector size.

6) (tmp is in the stopword Trie) or (tmp is AND) or (tmp is filetype:txt): discard it (continue);

*check if the final letter of tmp is **

Bonus: Incomplete match

Make a new function to search the prefix. When it reaches the final letter of the prefix, we traverse all the nodes below it (DFS) to get all the word with that prefix. Get all their occurrence places.

check if the first letter of tmp is a number of dollar $, then check if tmp has two adjacent dots.

7) search a range

First a function is implemented to get the lower-bound and upper-bound from tmp. (for example, $40.5..$100, so double lower-bound = $40.5, double upper-bound $100, also need to use atof to convert string to double type for comparisons). Then we traverse (DFS) all the pointers of numbers and the dot (decimal number case), convert it into double, then compare it with the lower-bound and upper-bound. If it satisfies lower-bound <= number <= upper-bound, stores (get union with the current output vector) its occurrence places.

8) Search hashtag, price: simply search like a normal word.

If a single word is not found in the Trie: check if it is OR case

```
bool check = true;
while (ss>>tmp)
    if (tmp is OR)
        ss>>tmp;
```

searchRes= search tmp in the Trie

If (tmp is found)

stores its occurrence places

check = true;

break;

else return false;//not the OR case

if (check if false) return false;//none of words in the OR sequence found

Else if a single word is found in the Trie: stores its occurrence places.

If the stream (query) passes all the cases, return true;

Some features:

- Search synonyms: We use a dictionary from Gutenberg project (which has no cost and no restrictions): http://www.gutenberg.org/files/51155/51155-h/51155-h.htm, then we store it in a text file. When "~" is encountered, the program will takes all of the word synonyms, then search for them.

- It could handle some combined queries.

Example 1: coffee AND tea OR milk

As observed, a normal search engine (like Google) will consider it as: coffee AND (tea OR milk). First it will search for coffee, pass through AND, then search for tea, which is the OR case.

Example 2: "Manchester United" +and "Paris Saint-Germain" –city intitle:soccer news

- History:

History (list of former queries) is stored in a text file. When a user asks for history suggestion, the program will display the queries of which the input query is a substring. Also an option to clear the History.

Ranking documents

The engine will display the satisfied documents in this order:

For query that does not have operators (for example: new president Trump), the engine will display the exact match first, then the AND case (the words stand in separated places).

In the main: a function is implemented to check whether the query contains operators or not.

If it is true, query = " + query + ", then put that query in the searching function.

Document scoring: the other documents will be ranked based on the occurrences of the words in the query. In the above function (in The Query part), we will also calculate the scores in each case. If a word is found, we get its occurrence places, then add to the scores of that document: score = score + tmp->occurrence.size(); (it does not support the costs, range and incomplete match)

Output results

A function is implemented: OutputResult(string filename, vector<int> pos)

We open each satisfied documents to output the sentences that contain the satisfied keywords, then colors them in order to make them easier to observe. Note that we need to consider the decimal number case (because in a decimal number, there is a dot, which is also the sign of end of sentence)

Open the file;

string title; get the first line which is the title

cout the title;

move the pointer to the beginning of the document;

int cnt = 0, cur = 0, totalLength = -1;

(cnt is used to traverse the places in the document, cur is used to get the elements in pos vector, totalLength is used to determine whether a sentence contains the elements in pos vector)

+The algorithm is: get sentences in order (tmp). Firstly, we consider the decimal number case: if the final letter of the sentence is a number, get the next sentence (next) and check if it exists and if the first letter of the next sentence is also a number or not. If it is true, tmp = tmp + "." + next, else tmp = tmp + " " + next.

Then we filt the sentence in order to avoid non-ascii or special unwanted letters. A function is implemented to count the number of word (NumWord) in a sentence (use sstream and a counting variable). totalLength = totalLength + NumWord.

If (pos[cur]>totalLength) //the sentence does not contain the desired keywords

cnt = cnt + NumWord;

continue;

cout<<"…" //for decoration

Put tmp into sstream. While (ss>>tmp), we continually cout each word in that sentence.

If it reaches the keywords

++cur;

Uppercase that word

Color that word, then cout it.

Else

Cout the word;

++cnt;

After all, cout<<"…"<<endl; for decoration.

```
If (cur>=pos.size())//which means we have color all the keywords
        Return;
```