DS_final_report

姓名：鄧弘利
學號：111062107

How you implement your code

我事先在每一次執行 query 搜尋項目前，先對傳進來資料夾中的所有 txt 檔案，先建立 trie，並且，我額外建立了一個每一個字的輸入都是顛倒的 trie_reversed，這個的原因是因為我想要在做 suffix 查詢的時候，用 trie_reversed 查比較快速。並且我在全域設了一個儲存 trie 還有 trie_reversed 的 vector。我有先對傳進來的資料檔名先進行排序處理，所以陣列儲存順序就是從 0 開始，一次加一(0.txt 存在 tries[0]還有 tries_reversed[0]，1.txt 則是 tries[1]還有 tries_reversed[1])。在建立 Trie 的時候，也有一些額外的處理，會先讓輸入的字串，去掉不是英文字母，還有都轉成小寫。針對每篇文章的標題，我是用 vector<vector<string>> Titles 去存。至於 query 的部分，我會每一次讀取 query 的一行搜尋項目，將遇到的+ - /先儲存在一個 op 的 queue 中。並且將被運算符分開的字串儲存在 search_words 這個 queue 中。接下來要決定輸出符合條件的檔案名稱的部分，我會跑一個 for 迴圈，去一個一個跑剛剛建立的 trie。每次先取 search_words 的 top 字串並 pop，接著丟進找尋字串是否在 trie 的 function，該 function 回傳的是 bool，如果是 true 代表有找到，false 則是沒有。接下來跑一個 while loop，直到 search_words 變為 empty 才結束。每次從 search_words 的 top 取出字串並 pop 後就丟進找尋的 function，接著從 op pop 出指令，進行邏輯運算，並將結果儲存在 valid。最後如果 valid 為 1 代表該篇 txt 有符合搜尋條件，否則沒有。當 valid 為 1，就把該檔案的標題存在 titles 中。然後接著跑下一個 trie，直到跑完。跑完之後把 titles 裡面符合搜尋條件的資料輸出。至於是怎麼搜尋的，我分別建立了三種的搜尋 function。
第一個是找 prefix 的方式。Ex . graph。這個就比較基本，就是從 root 開始，針對我們丟給它要找的字串，去找，如果都找到就返回 true。
第二個是找 exact word。Ex . “graph”。這跟第一種方式差不多，但是找完之後還要去確認是不是一個字的結尾。
第三個是 wildcard 的搜尋。Ex . <com*on>。這個主要是透過遞迴的方式去找，一個情況是跑忽略*，另一個是先保留*。

Other implementations for optimization

在找 suffix 的時候，因為我已經建立 trie_reversed(建立資料時每一個輸入的字串都是翻轉的)，所以我只要將 query 要找的字串先翻轉，再丟入找 prefix 的 search function，並且是找 trie_reversed，這樣就可以達到效果。另外在建立 Trie 的時候，我會先將文章中的文字轉為小寫，還有忽略不是英文字母。

```cpp
// string parser: output vector of strings (words) after parsing
vector<string> word_parse(vector<string> tmp_string, Trie& trie) {
    vector<string> parse_string;
    for (auto& word : tmp_string) {
        //transform(word.begin(), word.end(), word.begin(), ::tolower);
        trie.insert(word); // Insert the original word into Trie
        parse_string.emplace_back(word);
    }
    return parse_string;
}
vector<string> word_parse2(vector<string> tmp_string, Trie& trie) {
    vector<string> parse_string;
    for (auto& word : tmp_string) {
        // 保留單詞中的英文字母
        word.erase(std::remove_if(word.begin(), word.end(), [](char c) { return !std::isalpha(c); }), word.end());

        // 轉換為小寫
        transform(word.begin(), word.end(), word.begin(), ::tolower);
        //cout << word << endl;
        trie.insert(word); // 將原始單詞插入 Trie
        parse_string.emplace_back(word);
    }
    return parse_string;
}
```

Challenges you encounter in this project ,or Conclusion

遇到困難的地方，主要在於一開始不熟悉 Trie 的運作方式，還有整體資料結構運作的方式。另外在做找 wildcard 的 function 時，也讓我困惱了很久，不知該怎麼做，是去網路上查詢別人是怎麼實作之後，才有了初步的概念。

References that give you the idea (github/paper…)

Trie (字典樹)

https://ithelp.ithome.com.tw/articles/10324306

geeksforgeeks

https://www.geeksforgeeks.org/trie-insert-and-search/

stackoverflow

https://stackoverflow.com/questions/8145552/simple-wildcard-search-algorithm-in-c

github

https://github.com/jamesonwilliams/wildcardtrie/blob/master/src/main/java/org/nosemaj/wildcardtrie/WildcardTrie.java

https://github.com/jamesonwilliams/wildcardtrie