Final Project Report

學號: 109062129　　姓名: 林奕廷

1. Implementation

```
// trie node
struct TrieNode {
    struct TrieNode *children[ALPHABET_SIZE];
    // isEndOfWord is true if the node represents
    // end of a word
    bool isEndOfWord;
};

// vector of TreeNode roots for each txt file
vector<TrieNode *> tries;
vector<TrieNode *> tries_reversed;
```

我使用 Trie 的方式來實作，每個 txt 會建兩個 tries，每個 trie 的 root 另外用 vector 記錄下來。

Trie 的定義和相關 Function 我使用了 GeeksForGeeks 的 code 範本。

```
// Search function for "exact" key in trie
bool search(struct TrieNode *root, string key) {
    struct TrieNode *pCrawl = root;

    for (int i = 0; i < key.length(); i++) {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            return false;

        pCrawl = pCrawl->children[index];
    }

    return (pCrawl->isEndOfWord);
}
```

搜尋 exact key 的方式就是從 root 不斷往 child 找，如果在 key 的長度內都符合，而且最後一個 child 是 leaf 的話就代表找到。找 prefix 的話只要不確認是否是 leaf 就可以了。

找 suffix 的部分，我在建立一般的 trie 同時，也把每個單字反轉後再建另外一個 reversed trie。尋找的時候，只要也把 key 反轉，就能用找 prefix 的方式來找。

對於每行 query，我使用學期初學到的方法，將 prefix 轉成 postfix，然後使用 stack 的 push 跟 pop 來運算。這次的規則是 left associative，應該可以不用這個方式，不過這個方式只要修改一下，就能應付有括號的情況。

最後，在每次讀入 query 後，我會檢查 query 是否有重複。如果重複的話，就直接用之前儲存下來的符合結果的 titles 作為輸出，就不用再運算一次，減少時間浪費。

```cpp
// infixToPostfix conversion
vector<string> infixToPostfix(vector<string> key_string) {
    vector<string> key_postfix;
    stack<string> s;
    for (auto i : key_string) {
        if (i == "+" || i == "/") {
            while (!s.empty()) {
                key_postfix.push_back(s.top());
                s.pop();
            }
            s.push(i);
        } else {
            key_postfix.push_back(i);
        }
    }
    while (!s.empty()) {
        key_postfix.push_back(s.top());
        s.pop();
    }

    return key_postfix;
}
```

```cpp
// Evaluate function for postfix query
bool eval(TrieNode *root, vector<string> key_postfix, int num) {
    stack<bool> s;
    for (auto i : key_postfix) {
        if (i == "+" || i == "/") {
            bool right = s.top();
            s.pop();
            bool left = s.top();
            s.pop();
            if (i == "+") {
                s.push(left & right);
            } else {
                s.push(left | right);
            }
        } else {
            if (i[0] == '"') {
                string sub_string = i.substr(1, i.size() - 2);
                s.push(search(root, sub_string));
            } else if (i[0] == '*') {
                string sub_string = i.substr(1, i.size() - 2);
                reverse(sub_string.begin(), sub_string.end());
                s.push(search_prefix(tries_reversed[num], sub_string));
            } else {
                s.push(search_prefix(root, i));
            }
        }
    }
    return s.top();
}
```

2. Challenges

- 一開始不知道要如何下手，不過上網搜尋後就有找到方法。
- 不知道怎麼 parse 所有的 query，後來想到學期初教過的方法。
- 不知道如何 iterate 資料夾內的所有檔案，後來發現可以用 directory_iterator。
- 找不到如何安裝 gcc 7.5.0 的版本，最後裝了 7.3.0。

3. References

- Trie 的 definition 和 search funciton
  https://www.geeksforgeeks.org/trie-insert-and-search/
- LeetCode 上類似的題目
  https://leetcode.com/problems/prefix-and-suffix-search/
- CSDN 上對於這題的詳解，提到如何搜尋 suffix
  https://blog.csdn.net/qq_32523711/article/details/107814617
- 另一個詳解
  https://blog.csdn.net/magicbean2/article/details/79525726?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link&utm_relevant_index=2