主題：本地端資料內容搜尋

組員名單與分工表：
B00902003 游斯涵 利用 tfidf 設計 query，執行並紀錄實驗
B00902013 林育丞 code 簡化與效能提升，query 設計
B00902062 陳俊瑋 trie 資料結構實踐，設計演算法
B00902083 林修平 撰寫 report，檔案分析前置處理

目的：
今年要畢業了，但是看到過去四年滿滿的作業檔案，很難找出過去曾經打了什麼特定內容的檔案或者文件。Windows 內建的搜尋不但慢，而且常常會搜尋不到該找到的檔案，又不能針對文件內容作搜尋。
寫過作業一，知道可以藉由算出 query 和要搜尋的文件的相關性來決定 search 的 ranking，所以對於如何實踐這樣的問題有了初步的想像和解法。這樣的 scenario 類似作業一。
有了想要搜尋本地端大量文件的需求，又有了解決方法的構想，現階段的搜尋系統又完全不能解決這個問題，於是我們決定做這個題目。

方法：
Trie Search
雖然了解要如何 ranking 了，但是要如何有效率的 search 還是一個大問題。經過一番搜尋研究，我們發現 Trie 這個資料結構非常適合存放 word 的資訊，比起一般搜尋需要 O(所有文件內容長度和)才能找到想要的文件，Trie 是一個字母樹，每個 node 都是一個字元，從 root 開始，一直往下 traverse 整個 word，並在最後一個字元的下一個 node，存放想要儲存的資訊。這裡做為本地端的搜尋，我們存放的是出現該 word 有哪些絕對路徑，搜尋效率是 O(word 長度)，大大提升了搜尋效率。
Docx reader
又因為想解決的問題是過往打過的報告內容，所以許多需要 traverse 的檔案為 word 的.docx 檔案，因為此我們需要 python 方便的 zipfile library 和簡易解析 xml 的工具，讓我們能快速有效地提出 docx 中的內容，至於其他檔案的部分則只支援一般 open 能打開的可讀檔案，包括.txt，一些程式檔案等等，所以這個系統也能搜尋過往打過 code 的內容，可以快速找出過往打了那些特定變數的 code。同時我們將文件內容視為檔名加上其檔案內的內容。
Query
為了簡化資料結構以及符合中英文的特性，演算法設計成，如果讀到一串英文字串，我們就將這個英文字串 insert 到 Trie 裡，如果是讀到兩個以上的中文字串，那我們就將中文字串兩兩分別 insert 到 Trie 裡(bigram)，如同鄭老師上課所說，

中文大部分的(70%-80%)意義都是由 bigram 所組成，因此處理中文 bigram 是非常值得且有效率的。因此設計上，可以複數搜尋，搜尋結果不是交集而是聯集，只要 query 中任何一個 word 在某 path 的檔案出現，即會出現在搜尋結果中。分割搜尋字串的符號是”,”，搜尋限制是，中文必須兩個字，英文必須是一個單字。

Tf-idf ranking

如同作業一一樣，只是存放的資料結構變成了 Trie，必須 ranking 之前聯集出的 search result，tf 就用該 word 在 trie 中這個 result 的 path 出現了幾次（同一個文件中重複出現會重複插入），除以文件字詞的長度，idf 的部分就看這個 word 下出現了幾種不同的 path 進行 log 差運算後可得出，同樣的方法算出 query 的相似度後，兩者用 cosine similarity 算出 score 後，再按照 score 來排序。

執行參數：

python main.py [-d (搜尋起始位置)] [-q (搜尋字詞，,隔開)] [-r(有打的話結果排序)]

實驗結果：

大部分的 time 都用來 traverse file。Searching 跟 ranking 花的時間相對地非常少，因此文件內容的大小和要 traverse file 的數量決定了搜尋的速度快慢。

a. 156MB，334 個檔案，84 個資料夾含有大量 docx 檔案：
   需要 20.370 秒搜尋排序完畢
b. 4.5GB，10354 個檔案，2132 個資料夾含有大量 docx 檔案：
   需要 10 分 15 秒搜尋排序完畢
c. 13.7GB，37184 個檔案，4219 個資料夾含有大量 docx 檔案：
   跑到 12 分 23 秒 memory leak(Trie 太大，內存記憶體不夠)

Future work：

1. 可以將 python 轉換成 C 增加效能
2. 支援英文的 bigram，中文的 trigram 等等
3. 可以將 Trie 的資料結構存下來，增加重複搜尋時的效率
4. GUI 看起來會更 Fancy 一些