實作方式

使用的程式語言：c++

Utf8 解碼

由於 c++ 沒有內建 utf8 的支援，所以我先自己建立了一個 utf8 解碼器。利用 utf8 第一個 byte 的 prefix ，來判斷這即將被解碼的資料是一個長度為多少(1~4)的字。隨後，利用 shifting 的方式，將整個經過 utf8 編碼後的字解碼存在一個 int 中。重複上述步驟即可將整份 utf8 編碼文件的資料讀取完畢。

程式啟動時，會先將 keyword 文件解碼後讀入 c++的 std::map< std::vector<int>, int > 中，要比對的文字則是解碼後存入一個 std::vector<int> 中。

比對

有了解碼過後的文件後，我對於所有的開頭，將子字串從長度 7, 6, 5, 4, 3, 2 依序比較是否出現在 keyword 集中。如果出現的話，我就將 keyword count 加一，並向後移動到keyword之後。

舉例而言，如果有一句話是 “今天天氣真好，我喜歡晴天”，關鍵字是“今天”、“天天”的話，我的比對部分的程式會如下面所述進行：

設 “今天天氣真好，我喜歡晴天” 從左到右每個字的索引編好分別為 0 到 11。

比對程式會從索引 0 的字開始，取長度為 7 的子字串，意即 “今天天氣真好，”。之後，比對此字串是存在於否在 keyword 集中。以此例而言，答案為否，程式會進而縮短子字串長度而獲得“今天天氣真好”，並重複上述比對過程。

當子字串長度縮至2時，我們會發現“今天”存在於keyword集中，因此，我們會將今天的計數加一，並且下次的比對，將會從索引 2 ，子字串長度為7，開始繼續比較。

最終，我們會得到結果 有1個“今天”，有0個“天天”。

效能

測試資料

感謝陳丕祐同學提供他從維基百科索爬下的大量文字資料與keyword，文字檔案的部分大約 598 MB ，keyword的部分是約 1.1 MB 。

時間與空間

由於使用的是 map 與 vector，所以效能並不甚好。上述的測資下，單線程運行的話，讀取檔案約需花費 25 秒，並且進行比對約需花費 200 多秒。如果使用多線程的代碼跑在四核心的機器上，時間約可以減少三倍多，比對可以降低至 60秒內。(CPU: i5-5257U)

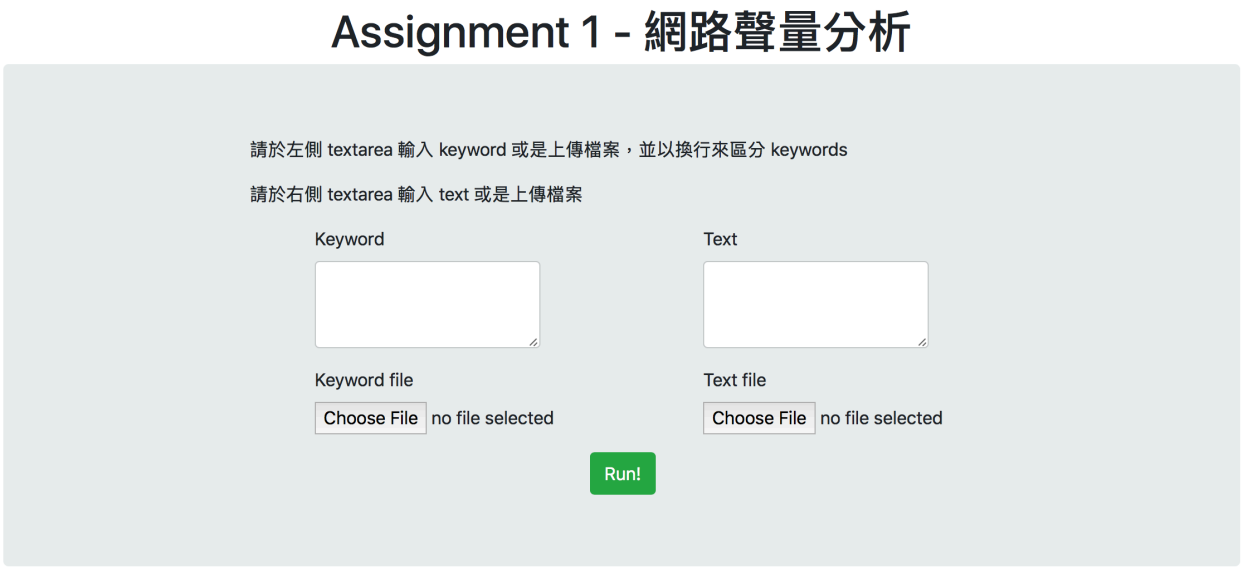
記憶體使用量也因為使用 map 與 vector 所以不太佳，最多會使用到 4 倍的文件大小的空間（在讀取檔案時，vector會以2倍的大小一直增長，直到shrink_to_fit()被執行後，才回降到使用兩倍大的記憶體空間）。

如果使用 Robin-Carp hashing 的話，效能應該可以提升非常多，只是實作未完成，所以無法進行測試。

從這次的作業，我深刻體會到 c++ 內建的函式庫的不足。比程式設計競賽時，我以為這些函式庫都已經是非常高速了，結果事實上並不是如此。

介面

使用 Bootstrap 進行介面設計。後端使用 python 的 tornado (http://www.tornadoweb.org/en/stable/) 進行 server 架設。



將 keyword 與 text 分別輸入並且送出後，可以得到頻率表。

選取 keyword 送出後，可以得到 highlighted 的結果。