B05902053 資工三 陳奕均 report

一、環境與使用方式

1. 環境: Windows 10 WSL

   ⇨ result1/*.txt, Makefile 是 unix 換行

   但我用 windows 的 sublime 寫 C++

   ⇨ source code 是 windows 換行

2. 編譯:

   打 make 或 make all 編譯 mapping 和 mydisambig

3. 執行:

   (1) mapping (print to stdout, redirection required)

   command: make map

   ./[Program] [Big5-ZhuYin] > [ZhuYin-Big5]

   (2) disambig (SRILM)

   command: make disambig

   說明: 可以重新產生 result1/*.txt (lm = bigram, order = 2)

   若欲以別的參數(lm,order, keep-unk) 來做 disambig

   要按照原先 disambig 的用法

   (3) mydisambig

   command: make run

   說明: 打 ./mydisambig -help 可以查看用法

   用法與 disambig 幾乎相同, 請至 mydisambig.cpp 的註解中查看用法

   consider-rare 的意思是說, 當這個 flag 沒有 set, 程式只會找出前 240 個 candidates 做 Viterbi

   例如: 假設 ㄔ 開頭的字有 1000 個, 當這個 flag 沒有 set, 程式只會考慮最常出現的 240 個字(比較 Unigram 的機率), 若不想捨棄剩下的 760 個字, 必須有-consider-rare, 意思是 考慮罕用字

二、實作細節

1. 各類別的責任:

   (1) CNChar: 是 union, 大小 2 byte, 定義中文字, 以及相關的運算子

   (2) NgramUtils: 負責算各種 n-gram 機率

   (3) Viterbi: 負責對每一行 input 做 disambig, 用到的是 Viterbi algorithm

2. Viterbi algorithm 處理細節

   每個位置都有各個可能的選項，每個可能的選項都對應到一個 state; 跟第一次作業不同的是, 這個 state 並不是 hidden 的。確定一個 state 後就能確定是該放哪個字。於是原問題相當於找到機率最大的路徑

   演算法參考自 FAQ(test) https://hackmd.io/s/ry8uGI31X

   assume index start from 1

```
/* when order is 2 */
procedure viterbi:
    for (i = 1; i < candidates[1].size(); i++) do
        delta(1, i) = P(W1 = i)
    done
    for (L = 2; L <= length(line); L++) do
        for (i = 1; i < candidates[L].size(); i++) do
            delta(L, i) = max (P(j | i) * delta(L - 1, j))
            /* j from 1 to candidates[L - 1].size() */
        done
    done
end procedure
```

其中，在每個長度 L, candidates 個數都不相同，無法用 C 的 array 實現
所以用 vector <T> 或 vector < vector < T> > 實做長度不一的表格。
值得一提的是，在做 trigram 時，時間複雜度為 O(length *n^3), n 是所有注音中 candidates 數量的最大值經實測 n 約為 1000，假設平均長度 10，代入得 10^10，這已經超出電腦的運算速率了, 10 分鐘一定跑不完。

解決之道: 我想到字典中有很多字平常出現頻率極低，既然 1000 太大，我們刻意把 n 縮小，只取前 n 常用字出來做 viterbi。經過實測決定將 n 定為 240，代入後約為 1.3 * 10^8， 這樣就能時限內(5 分鐘) 跑完，同時保有一定正確率。這就是參數-consider-rare 的用意。
所以此參數只能在 bigram 使用，trigram 用下去會嚴重超時。

3. Parsing command line arguments: 使用 SRILM 中的 library Opt_Parse()

三、實測結果

將 disambig 的輸出當作正確答案，比對 mydisambig 的輸出，觀察正確率

bigram										
1	1	3	1	4	2	4	5	1	3	all
1	1	3	2	4	3	5	5	1	3	top240
trigram										
9	8	5	6	5	6	7	7	3	7	top240

此表格的數字由左至右是 1.txt, 2.txt, … 10.txt 所得到的錯誤行數
bigram 正確率都有超過 90%, trigram 則有 80%
錯誤應是來自 disambig 不只做了 Viterbi，可能還有 smoothing，但我們沒做
此外，只取常用字也是有問題的，例如: 「叱吒風雲」的叱，就會沒考慮到
目前的策略只是確保一定正確率以及能通過時限，但應有更好的做法。