DSP HW3 Report R08922A04 林承德

<How to Execution>

我是在助教所提供的docker環境所撰寫，因為有做trigram，所以在執行MakeFile的時候，請執行
$ make all
這會將src裡面的mydisambig.cpp & mydisambig_trigram.cpp 編譯成 mydisambig與mydisambig_trigram，接著為了生成mapping file，請執行
$ make map
這會執行src裡面的mapping.py並將提供的Big5-Zhuyin.map轉換成 ZhuYin-Big5.map

兩個執行檔，所吃的參數相同，皆為頭影片中所要求的，依序為：

```
$1 segemented file to be decoded
$2 ZhuYin-Big5 mapping
$3 language model
$4 output file
```

1. What I observed ?

首先我比較了1~10各筆測試資料兼執行速度的差異，發現在後面的幾筆測資中有普遍執行速度較慢的趨勢，尤其在trigram中格外明顯，推測可能是因為這些資料中有出現較多的多個連續注音，使得計算量指數成長。

接著我比較了套件的disambig與我的mydisambig在bigram 與trigram在執行結果與執行速度的差異，首先在bigram上，disambig的執行時間為1分鐘，mydisambig則需要2分鐘，在執行結果上則沒有差異，而不管哪筆資料單獨拿出來比較執行速度也都大概差了兩倍，因此我想可能是有些地方沒有做好速度的優化，但苦尋良久都沒有找到改善的方法，接著在trigram的部分，除了時間的差異被放大外，執行出的結果也有些微的不同，有些測資大概看過去可能會有10來個字的不同，而普遍來看都是disambig做得比我的implement好，所以很有可能我是有些地方的機率並沒有處理好的，因而造成了這樣的差異。

2. What I have done?

1. mapping.py:

是一個python3的檔案，會讀入助教給定的Big5-ZhuYin.map，並將其轉換成 ZhuYin-Big5.map，輸入是每個國字可能有的注音，而輸出則是每個注音可能有的國字。

2. mydisambig.cpp

這裡面包含了兩個主要的部分，LoadMap的class 主要負責讀入前面先用mapping.py 處理好的ZhuYin-Big5.map，我這邊使用的是map<VocabIndex, set<VocabIndex>>的方式來記錄，每個注音所對應的的國字們的VocabIndex。
另外一個部分則是mydisambig.cpp的主體，也就是bigram版本的 viterbi algorithm。這邊基本是參考FAQ裡附的說明與推導，用兩個2維陣列來紀錄每個time state下的 delta 與 backtrack 用的Index。最後根據最高機率的路徑做backtrack，找出最後輸出的句子。

3. mydisambig_trigram.cpp

這份檔案則是對trigram的實作，和前一份相同的是，也包含了mapping檔案讀取的部分，這部分的實作與前一份相同便不再多加詳述。
值得一提的是，原本的實作方法是單純的使用三層的for loop從q_i, q_j去尋找 max q_k的方式，但經過實驗發現與套件所提供的disambig在執行效率上有極大的差距，因此在和同學討論後發現從q_j, q_k回推 P(q_i | q_j, q_k)是一個可行的辦法，因為如果delta(q_j, q_k)是logP_Zero的話呢，就表示這條路是不通的，我們也就不用往下繼續去算每個P(q_i l q_j, q_k)值。在經過這樣的改動後，執行速度變有明顯的提升，但仍達不到套件所提供的disambig的速度，不過跟原本比起來是個不錯的提升。