# Disambiguation

Environment: Ubuntu Linux 12.04.2 64bit / g++ 4.6.3 / Python 2.7.3

How to compile:

1. **mydisambig** -> $ make
   # 須先編輯 Makefile 將 SRIPATH 和 MACHINE_TYPE 設好
   # 我的 SRIPATH 位於 $(HOME)/srilm-1.5.10，MACHINE_TYPE 則為 i686-m64
2. **mapping.py** -> No need to compile.

## How to Execute

### mapping.py

使用 python (2.7) 執行 mapping.py，並以舊的對照表檔名作為輸入，最後再將輸出用 > 導至一個檔案：

```
$ python mapping.py Big5-ZhuYin.map > ZhuYin-Big5.map
```

### mydisambig

我的 mydisambig 共有 4 個必要參數，照順序分別為「要處理的文字檔」、「mapping 檔案」、「language model 檔案」、「指定的 ngram order」。其中 ngram order 可為 1~4 等等，若更高效率就很差，所以雖然我的程式可改成支援更高的 order，但我限制最高只能到 4。

此外，可以在最後面加上「-prun N」，其中 N 為一個 1 以上的正整數，加上後可以有 pruning 的效果，不過 N 並不是直接對應到最好的幾條路徑，只是數字越大保留路徑越多，N = 0 的話則是沒有 pruning。

```
$ ./mydisambig input.txt ZhuYin-Big5.map unigram.lm 1 > output.txt          # Unigram
$ ./mydisambig input.txt ZhuYin-Big5.map bigram.lm 2 > output.txt           # Bigram
$ ./mydisambig input.txt ZhuYin-Big5.map trigram.lm 3 > output.txt          # Trigram
$ ./mydisambig input.txt ZhuYin-Big5.map trigram.lm 4 -prun 2 > output.txt  # Fourgram + pruning
```

## Requirement I (40%)

先將所有資料的 characters 用空白隔開：

```
$ perl separator_big5.pl corpus.txt > corpus_seg.txt
$ perl separator_big5.pl testdata/1.txt > testdata/1_sep.txt          # for 1.txt~10.txt
```

緊接著得到 language model (我將 srilm-1.5.10/bin/i686-m64 加到 PATH 中，所以可以直接執行）：

```
$ ngram-count -text corpus_seg.txt -write lm.cnt -order 2
$ ngram-count -read lm.cnt -lm bigram.lm -unk -order 2
```

撰寫 mapping.py 程式讀取 Big5-ZhuYin.map 並轉成 ZhuYin-Big5.map，程式的運作方式是按照格式讀入檔案、將字輸入到以對應的注音開頭為 key 的 dictionary，最後再將資料 dump 出來：

```
$ python mapping.py Big5-ZhuYin.map > ZhuYin-Big5.map
```

使用 disambig 來產生答案：

```
$ for i in {1..10}; do disambig -text testdata/${i}_sep.txt -map ZhuYin-Big5.map -lm bigram.lm -order 2
> result1/${i}.txt; done
```

## Requirement II (40%)

接下來，我撰寫 mydisambig.cpp 來達成第二個要求。其中，我使用了 srilm 的 library 來加快開發速度。而 viterbi 的實作方法則是使用 VNode * 產生樹狀結構來代表 search tree。除此之外，我還使用了 Front * 建立

另外一個樹狀結構來紀錄目前 leaf nodes 是從「哪些字長出來的」，這個紀錄是為了加速 ngram probability 的計算，同時因為只紀錄 leaf nodes 的資訊，所以可以節省記憶體空間。樹狀結構可以很容易的支援任意 ngram，然而效率上就比較差，實際跑起來確實也比 srilm 的 disambig 慢一些。

```
$ for i in {1..10}; do ./mydisambig testdata/${i}_sep.txt  ZhuYin-Big5.map bigram.lm 2 > result2/${i}.txt; done
```

剛寫完的時候雖然跟 disambig 產生的答案幾乎一樣，但少數情況會出現很差的結果。我研究了程式的行為後發現，這是因為有時會遇到很稀有的字，導致所有的 transition probability 都是 0。則只要經過這個字，後面的機率就完全無法比較了。我研究了 disambig 程式的行為後發現，他在這種情況下會將機率取代為另一個自己的設定的很小的機率，於是我也採用同樣做法，就得到同樣的結果了。

## Bonus: Trigram Model (10%)

如前所述，我們可以很輕易的支援 trigram。這裡我同時用了 600 作為 pruning 的參數：

```
$ ngram-count -text corpus_seg.txt -write lm3.cnt -order 3
$ ngram-count -read lm3.cnt -lm trigram.lm -unk -order 3
$ for i in {1..10}; do ./mydisambig testdata/${i}_sep.txt  ZhuYin-Big5.map trigram.lm 3 -prun 600 > result3/${i}.txt; done
```

pruning 的作法主要是保留 N 條最有可能的路徑，不過為了實作的方便，最早遇到的 N 條路徑如果後來遇到機率較高的走法也不會被刪除，所以實際上可能保留超過 N 條路徑。

跑出來的結果跟 disamig 是一致的。

## Bonus: Other Strategies (5%)

如前所述，我們可以很輕易的支援 four-gram。這裡我同時用了600 作為 pruning 的參數：

```
$ ngram-count -text corpus_seg.txt -write lm4.cnt -order 4
$ ngram-count -read lm4.cnt -lm fourgram.lm -unk -order 4
$ for i in {1..10}; do ./mydisambig testdata/${i}_sep.txt  ZhuYin-Big5.map fourgram.lm 4 -prun 600 > result4/${i}.txt; done
```

跑出來的結果和 disambig 稍有不同。估計增加 pruning 參數應可得到同樣結果，但效率就差了。

同樣的，我們也可以很輕易的支援 uni-gram：

```
$ ngram-count -text corpus_seg.txt -write lm1.cnt -order 1
$ ngram-count -read lm1.cnt -lm unigram.lm -unk -order 1
$ for i in {1..10}; do ./mydisambig testdata/${i}_sep.txt  ZhuYin-Big5.map unigram.lm 1 > result-uni/${i}.txt; done
```

我用助教提供的 3 個測資來比較，發現其實 four-gram 進步的程度非常有限，甚至有差於 trigram 的情形。再加上效能上的考量，還是使用 trigram 比較實際。