Digital Speech Processing hw3

Environment

```
CSIE workstation
Python 3.6
```

How to compile

MakeFile is provided, so easily execute following commands

```
cd dsp_hw3_r06922095/
make all
```

How to execute

MakeFile is provided, so easily execute following commands

```
cd dsp_hw3_r06922095/
make all
make map
make run
```

Or typing commands by yourself as follow

```
cd dsp_hw3_r06922095/
python3 mapping.py Big5-ZhuYin.map ZhuYin-Big5.map
./mydisambig -text testdata/$$i.txt -map ZhuYin-Big5.map
 -lm bigram.lm -order 2 > result2/$$i.txt
```

Implementation

參考SRILM的manpages(http://www.speech.sri.com/projects/srilm/manpages/)，裡面提供許多有用的data structure、function、class。我利用Ngram class引入language model，利用<VocabMap.h>下的VocabMap class引入ZhuYin-Big5.map，利用<File.h>下的File class引入測試資料。另外，最好用的function莫過於Vocab::parseWords，因為

Big5編碼的中文字在C++佔2 Bytes，所以不論用string還是char處理都很麻煩，要維護array內的中文字數。這個function能斷開string並把斷開的words放在傳入的VocabString array內，不得不說這個manpages寫得沒有很詳細，這個function在manpages蠻難找到的。

SRILM的library也提供很多好用的資料型別，例如LogP、VocabIndex、VocabString等等，雖然只是在library內用原生資料型別typedef，但可以清楚地知道變數用來處理什麼。

演算法的部分則跟hw1的test很像，都是實作Viterbi algorithm，只不過這次的hidden state改成word candidate。

比較特別的地方是，我實作完後發現Viterbi會把原本不是注音的字改成<unk>，例如1.txt中的這句"還 挺 登 ㄉ的"，經過Viterbi後會輸出"還 <unk> 登 島 的"。後來看了bigram.lm發現原來P(登|挺)不在language model，所以可能P(登|<unk>)比P(登|挺)高，結果就是"<unk>"把"挺"取代掉。所以我在輸出前加了一條Rule-based，如果原本的word非"<unk>"但對應輸出的word卻是"<unk>"，那就保留原本輸入的word。加了這條rule後，mydisambig在1.txt到10.txt的輸出就跟SRILM的disambig都相同了。