//
/// 인공지능 기말고사 대체과제 ///
/// ///
/// Viterbi algorithm을 이용한 음성인식기 ///
/// ///
/// 컴퓨터학과 2018320241 정소희 ///
//

A. 구현 환경

- ubuntu 18.04 (linux 하위 시스템)에서
- c언어로 작성하고
- gcc를 사용해 compile하고 실행하였다.

B. Source Code Explanation

주어진 파일 중 수정한 것은 hmm.h이고 새로 작성한 것은 Viterbi.c 파일이다.

1. hmm.h에서 hmmType의 tp배열의 행의 크기와 state배열의 크기를 다음과 같이 변경하였다.

```
typedef struct {
  char *name;
  float tp[N_STATE*8+21][N_STATE*8+21];
  stateType state[N_STATE*8]; //
} hmmType;  //hmm consist of name, trans_matrix, states
```

N_STATE -> N_STATE * 8 로 변경한 이유는, dictionary.txt를 참고했을 때 한 단어에 존재하는 최대 음소의 개수는 (sp를 포함하여) 6개이므로, 단어 hmm을 표현하기 위해 matrix와 배열의 크기를 늘려 주었다.

2. 음성 인식기를 구현하기 위해 Viterbi.c를 작성하였다. Viterbi.c에 다음과 같은 함수들과 구조체를 생성하였다.

```
#include "hmm.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define max(a,b) a > b ? a : b;

typedef struct {
    hmmType hmm; //hmm of each words(or unigrams)
    char* word; //word
    int phnum;  //number of phones
} word_hmms_type;

void make_unigram(int,word_hmms_type*,word_hmms_type*); //make unigram for each words
float* make_bigram(int,word_hmms_type*,float*); //create bigrams transition matrix Tb, size = (wordcount X wordcount)
hmmType* ADD_HMM(hmmType*,int,int); //add two hmms (state#=3)
hmmType* ADD_SP(hmmType*,int,int);  //add sp hmm (state#=1)
float calc_gausian(int,int,word_hmms_type*,int,float*); //calculate gausian probability
void Viterbi(char*,int,word_hmms_type*,word_hmms_type*,float*,float*); //Viterbi algorithm
```

- word_hmms_type은 phone hmm을 합친 단어 hmm을 생성할 때 hmm, 구성하는 phone

의 개수, 그리고 단어 이름을 함께 저장하기 위해 만든 구조체이다.

- max(a,b)를 정의하였다.
- hmmType* ADD_HMM(hmmType*,int,int) 함수는 어떤 hmm 뒤에(phone hmm이 1개 이상 결합된 hmm) state 개수가 N_STATE인 phone hmm을 결합할 때 사용되는 함수이다. 마찬가지 원리로 state가 1개인 sp phone hmm을 붙일 때 사용하는 ADD_SP 함수도 생성하였다. 여기서 k는 이 hmm이 최초의 단일 phone hmm으로부터 몇 번째로 더하는 hmm인지를 나타낸다. k가 증가하면 전체 state의 개수인 n_state도 증가한다. (150째줄)

```
147 hmmType* ADD_HMM(hmmType* f,int b,int k){ //kth addition (f+b)
148     hmmType* front = f;
149     hmmType back = phones[b];
150     int n_state=N_STATE*(k+1); //new number of states
151 
152     //new name concatenation
153     char* front_name = strdup(front->name);
154     char* back_name = strdup(back.name);
155     strcat(front_name, back_name);
156     front->name=NULL;
157     (front->name) = strdup(front_name);
158     free(front_name);   free(back_name);
159 
160     //new state concatenation
161     for(int i=0;i<N_STATE;i++){
162         (front->state)[i+(n_state-2)-1] = back.state[i];
163     }
164     //new transition matrix concatenation
165     for(int i=0;i<4;i++){
166         for(int j=1;j<5;j++){
167             if(i==0 && j==1)
168                 (front->tp)[i+N_STATE*k][j+N_STATE*k] *= back.tp[i][j];
169             else{
170                 (front->tp)[i+N_STATE*k][j+N_STATE*k] = back.tp[i][j];
171             }
172         }
173     }
174     return front;
175 }
```

(164~173줄) 두 개의 hmm을 합치기 위해서는 앞쪽 hmm의 output이 뒤쪽 hmm의 input으로 이어질 수 있는지 확인한 후, 이어지는 모든 부분에 대해 확률을 곱해서 연결해준다. 167~168줄에서 두 확률을 곱해서 hmm들을 이어주었고 170줄에서 뒤쪽 hmm의 tp를 앞쪽 hmm에 이어주었다. (161~162줄) 에서는 뒤쪽 hmm의 state들을 앞쪽에 추가해 주었다. 또한 string duplicate시켜 hmm의 이름도 바꿔주었다.

- void make_unigram(int,word_hmms_type*,word_hmms_type*) 함수

```
206 void make_unigram(int wordcount,word_hmms_type* word_hmms,word_hmms_type* uni_hmms){
207     FILE *uni;
208     //word_hmms_type uni_hmms[15];
209     for(int i=0;i<wordcount;i++){
210         uni_hmms[i] = word_hmms[i];
211     }
212 
213     if(!(uni = fopen("unigram.txt","r"))){
214         printf("Error opening file\n");
215         exit(1);
216     }
217 
218     char line[35]; //dict의 각 line
219     while(fgets(line, sizeof(line), uni)){
220         if(line[strlen(line)-1]=='\n'){
221             line[strlen(line)-1]='\0';
222         }
223 
224         char* word1;
225         float prob;
226         char * token;
227 
228         token = strtok (line, "\t"); //word1
229         word1=token;
230 
231         token = strtok(NULL,"\t");  //prob
232         if(token[strlen(token)-1]==13) //carriage return
233             token[strlen(token)-1]='\0';
234         prob = atof(token);
235 
236         for(int i=0;i<wordcount;i++){
237             if(strcmp(uni_hmms[i].word, word1)==0){
238                 ((uni_hmms[i].hmm).tp[0][1]) *= prob;
239                 break;
240             }
241         }//for
242     }//while
243     fclose(uni);
244 }
```

make_unigram 함수는 unigram.txt에서 확률 값을 읽어와서 (234줄) 앞서 만들었던 word_hmm들에 추가로 input probability를 곱해주어(238줄) 새로운 uni_hmm들을 만든다. word_hmm과 uni_hmm의 차이는 transition probability matrix의 값이고 나머지는 모두 동일하다. 이 함수의 input으로는 생성된 word hmm들과 unigram hmm이다.

- float* make_bigram(int,word_hmms_type*,float*) 함수

  make_bigram 함수는 먼저 bigram.txt에서 확률 값을 읽어온다. (271줄)

```
246  //create bigrams transition matrix Tb sized wordcount * wordcount
247 float* make_bigram(int wordcount,word_hmms_type* word_hmms,float* Tb){
248     FILE *bi;
249     if(!(bi = fopen("bigram.txt","r"))){
250         printf("Error opening file\n");
251         exit(1);
252     }
253     char line[35]; //dict의 각 line
254     while(fgets(line, sizeof(line), bi)){
255         if(line[strlen(line)-1]=='\n'){
256             line[strlen(line)-1]='\0';
257         }
258         char* token; char* word1; char* word2;
259         float prob; int x,y;
260
261         token = strtok (line, " \t");
262         word1=token;    //word1
263         token = strtok(NULL," \t");
264         if(token[strlen(token)-1]==13) //carriage return
265             token[strlen(token)-1]='\0';
266         word2=token;    //word2
267
268         token = strtok(NULL," \t");
269         if(token[strlen(token)-1]==13) //carriage return
270             token[strlen(token)-1]='\0';
271         prob = atof(token); //prob
272
```

그리고 ith word hmm -> jth word hmm으로 transition 할 확률을 모아놓은 matrix Tb를 생성한다. (287줄)

```
273         //find word1 index
274         for(int i=0;i<wordcount;i++){
275             if(strcmp(word_hmms[i].word, word1)==0){
276                 x=i;
277                 break;
278             }
279         }//for
280         //find word2 index
281         for(int i=0;i<wordcount;i++){
282             if(strcmp(word_hmms[i].word, word2)==0){
283                 y=i;
284                 break;
285             }
286         }//for
287         Tb[x*wordcount+y] = prob;   //push prob into table Tb
288     }//while
289     fclose(bi);
290     return Tb;
291 }
```

Viterbi algorithm에서 한 word hmm에서 다른 word hmm으로 갈 확률을 구할 때 Tb값이 쓰인다. 이 함수의 input은 생성된 word_hmm의 개수, word_hmm, 그리고 Tb matrix이고 output은 생성된 Tb matrix이다.

- float calc_gausian(int,int,word_hmms_type*,int,float*) 함수

calc_gausian 함수는 말 그대로 gausian 확률분포 식과 주어진 정보들을 가지고 확률을 구하는 함수이다. 이 때 gausian 확률분포 값이 매우 작아질 수 있으므로 log를 취해서

계산해준다. input은 진입할 state의 index #, word_hmm 배열의 index, feature vector의 dimension, 그리고 feature vector의 원소를 저장한 배열이다. pi_val 변수는 $(2\pi)^{D/2}$ 의 값이다. log_p_gaus은 pdf의 gause확률변수 값에 log를 취한 것이다. gause확률변수 값을 구하는 식을 그대로 적용하고 weight를 도입하여 각각 log_p_gaus1과 log_p_gaus2를 구하고, result를 도출하였다. (456줄)

```
423 float calc_gausian(int s_index,int j,word_hmms_type* word_hmms,int dimension,float* features){
424     //각 gausian은 word_hmms[j].state[1부터 (word_hmms[j].phnum-1)*3+1까지].pdf[0 아니면 1]에 저장돼있다.
425     // j = word index
426     float pi_val = pow(2*M_PI,dimension/2);
427 
428     //gausian model 1의 값 구하기 (p_gaus1)
429         pdfType pdf1;
430         pdf1= (word_hmms[j].hmm).state[s_index].pdf[0];
431         float weight1 = pdf1.weight; //float
432         //with log
433         float log_sum_var1=0;
434         float p_gaus1=0;
435         for(int i=0;i<dimension;i++){
436             log_sum_var1+=1/2*logf(pdf1.var[i]);
437             p_gaus1+= pow((features[i] - pdf1.mean[i]),2)/(pdf1.var[i]);
438         }
439         float log_p_gaus1 = logf(weight1)-(dimension/2)*logf(2*M_PI)-log_sum_var1- (1/2)*p_gaus1;
440 
441     //gausian model 2의 값 구하기 (p_gaus2)
442         pdfType pdf2;
443         pdf2= (word_hmms[j].hmm).state[s_index].pdf[1];
444         float weight2 = pdf2.weight; //float
445 
446         //with log
447         float log_sum_var2=0;
448         float p_gaus2=0;
449         for(int i=0;i<dimension;i++){
450             log_sum_var2+=1/2*logf(pdf2.var[i]);
451             p_gaus2+= pow((features[i] - pdf2.mean[i])/pdf2.var[i] , 2);
452         }
453         float log_p_gaus2 = logf(weight2)-(dimension/2)*logf(2*M_PI)-log_sum_var2- (1/2)*p_gaus2;
454 
455     //결과 합치기
456         float result = log_p_gaus1 + logf(1+exp(log_p_gaus2-log_p_gaus1));
457 
458     return result;
459 }
```

- void Viterbi(char*,int,word_hmms_type*,word_hmms_type*,float*,float*) 함수

: Viterbi 알고리즘을 구현한 함수로, 단계별로 나누어 설명하면 다음과 같다.

1. 다음과 같이 'filename'(tst file) 이름을 가지는 txt file을 불러오고 첫 번째 line을 읽어들여 총 행의 개수와 feature vector의 dimension을 알아낸다. 그리고 m1, m2배열을 생성한다. (뒤에서 설명)

```
307 void Viterbi(char* filename,int wordcount,word_hmms_type* word_hmms,word_hmms_type* uni_hmms,float* Tb,float* features){
308     FILE* feat;
309 
310     if(!(feat = fopen(filename,"r"))){
311         printf("Error opening file\n");
312         exit(1);
313     }
314     char line[2000]; //each line
315     int countline=0;
316     int dimension,nrows;
317 
318     fgets(line, sizeof(line), feat);
319     //initial line read
320     if(line[strlen(line)-1]=='\n'){
321             line[strlen(line)-1]='\0';
322         }
323     char * token;
324     token = strtok (line, " \t");
325     nrows=atoi(token);
326 
327     token = strtok(NULL," \t");
328     dimension = atoi(token);
329 
330     float m1[nrows][wordcount];
331     int m2[nrows][wordcount];
```

2. tst 폴더 안의 각 .txt 파일은 2번째 line ~ nrows번째 line까지 feature vector 값을 읽어들인다. 한 행에는 39개의 feature vector 값이 있다. 그 값들을 features 배열에 저장한다. 이 features 배열은 calc_gausian 함수로 넘어가서 확률 계산에 쓰이게 된다.

```
333     //next line read
334     int k=1;
335     while(fgets(line, sizeof(line), feat)){ //each 39 dimension feature vector
336         if(line[strlen(line)-1]=='\n'){
337             line[strlen(line)-1]='\0';
338         }
339 
340         token = strtok (line, " ");
341         int countfeature=0;
342 
343         while(token != NULL){
344             features[countfeature++]=atof(token);   //feature vector -> put into array
345             token = strtok(NULL,"  ");
346         }
347 
```

3. 여기서 m1 배열과 m2 배열은 각각 float과 int type을 저장하는 크기가 nrows * wordcount 인 matrix이다. k는 몇 번째 feature vector (총 nrows개) 중 몇 번째 feature vector인지를 나타내고 (k는 흐른 시간의 개념이므로 앞으로 k번째 시간이라고 하자), j는 word hmm의 index를 나타낸다고 할 때, m1[k][j]는 k 번째 시간에서 state j에 도달할 확률 중 가장 큰 값을 나타낸다. m1[k][j] 값을 최대로 만드는 transition이 i->j였다면 m2[k][j]는 i이다.

```
348     ////////////////viterbi algorithm start/////////////////////////
349     //for all unigrams (initialize at time k=1)
350     if(k==1){
351         for(int j=0;(j<wordcount)&&(k==1);j++){
352             m1[1][j]=(uni_hmms[j].hmm).tp[0][1] * calc_gausian(0,j,uni_hmms,dimension,features);
353             m2[1][j]=0;
354             k++;
355         }
356         continue;
357     }
```

tst file의 첫 번째 행을 읽으면, 첫 번째 행을 observation value로 하고 unigram을 사용하여 확률값을 계산해 m1, m2배열을 initialize한다(k=1에서). unigram j에서 m1[1][j]는

k=1 일 때 진입함이 보장되므로 unigram에서 1번째 state(=state[0])으로 진입할 확률이다. 따라서 m1[k=1][j]=(unigram[j]의 tp[0][1]값) * (unigram[j]의 state [0]에서의 gausian값)으로 나타낼 수 있다. m2[1][j]는 0으로 초기화한다.

4. 이제 2번째 시간부터 nrow 시간까지의 path를 구해야 한다. 시간 k= 1->2, 시간 2->3, ... 시간 nrow-1 -> nrow의 path를 recursive하게 구해 나간다. 이 때 각 시간의 흐름은 fgets로 파일을 한줄씩 읽는 것과 같다. k가 고정된 상태에서 아래 2번의 for문이 실행된다.

이 2개의 for문의 목표는 모든 가능한 word hmm에 대해, k 시간에 jth word hmm일 확률 중 최대값을 갖는 i->j transition을 찾아내는 것이다.

각 word hmm은 내부에 phone 개수와 비례하는 state 개수가 존재한다. ((phone개수-1)*3+1개) word hmm 내부의 state에 대한 최대의 확률값을 구하여 그것이 word hmm의 확률값(=여기서는 float gausian)을 대변하도록 하는 것이 목표이다.

우리는 내부 state사이의 이동에는 관심이 없고 그 전체의 gausian확률값만 max값으로 keep tracking하면 되므로 확률값을 저장하는 배열 p[t][s]를 둔다. 여기서 t는 word hmm 내부의 시간 개념으로, 최대 state 개수만큼의 횟수로 진행할 수 있다고 하였다.

바로 아래 for문은 p[1][s]를(t=1) word 내부의 첫 번째 state로 진입할 확률로 초기화하는 것이다. 그 확률값은 해당 word의 tp[0][1]에서 얻을 수 있다. (state1로만 진입할 수 있기 때문에)

```
358
359     //k는 2~nrow //k 하나는 fgets의 각 line 하나하나에 해당 //39차원 벡터(행) 하나에 대해 아래를 검사
360         for(int j=0;j<wordcount;j++){
361             //bigram transition matrix Tb (i->j)
362             float p[(word_hmms[j].phnum-1)*3+1+1][(word_hmms[j].phnum-1)*3+1+1];
363             for(int s=1;s<=(word_hmms[j].phnum-1)*3+1;s++){ //s-1은 state index, state#=s
364                 p[1][s]=(word_hmms[j].hmm).tp[0][s];
365             }
```

5. 다음 for문에서는 t가 진행하고 state transfer이 될 때 어떤 경로로 가는 것이 가장 큰 확률값을 갖는지 찾아낸 후 p[t][s]에 저장한다. (374~375째줄) 이전 시간의 state까지 갈 최대 확률 * state transfer 확률 * 해당 state에서 gausian확률(calc_gausian으로 구한다)을 계산하면 v->s 로 s에 도달했을 때의 확률을 구할 수 있다. 모든 v에 대해 최대 확률값을 찾기 위해 p[t][s]=max(p[t][s],calc_temp); 로 최대값을 계속 유지한다.

각 state 진행 (t 증가)에 대해 word hmm의 gausian으로 내보낼 float gausian 변수를 최대값으로 갱신한다. (379째줄)

```
367             float gausian=-100; //gausian 확률값 비교를 위해
368             float calc_temp;
369             for(int t=1;t<=(word_hmms[j].phnum-1)*3+1;t++){ //진행 횟수 t에서
370                 float temp=-100;
371                 for(int s=1;s<=(word_hmms[j].phnum-1)*3+1;s++){ //각 s에 대해
372                     for(int v=1;v<=(word_hmms[j].phnum-1)*3+1;v++){ //모든 가능한 transition v->s에 대해 max값 저장
373                         if(((word_hmms[j].hmm).tp[v][s])!=0){
374                             calc_temp=p[t-1][v]*((word_hmms[j].hmm).tp[v][s])*calc_gausian(s-1,j,word_hmms,dimension,features);
375                             p[t][s]=max(p[t][s],calc_temp);
376                         }
377                     }//v
378                 }//s
379                 gausian = max(gausian,p[t][(word_hmms[j].phnum-1)*3+1]); //모든 t에 대해 가장 큰 output 값r구함
380             }//t
381
```

6. 이제 위에서 구해진 word hmm의 gausian값을 이용해 시간 k에서 각각의 출발 state (ith word)에서 jth word에 도달할 확률을 구하고 최대값으로 갱신한다(386째줄). 동시에 m1[k][j]를 최대값으로 만드는 i값을 m2[k][j]에 집어넣어 m2를 갱신한다(388째줄). 모든 갱신 작업이 끝나면 다음 feature vector를 불러오고 (=k 증가) 똑 같은 연산을 수행한다.

```
382             //update m1[k][j], m2[k][j]
383             float temp1;
384             for(int i=0;i<wordcount;i++){
385                 temp1 = m1[k][j];
386                 m1[k][j]=max(temp1,m1[k-1][j] * Tb[i*wordcount+j] * gausian);
387                 if(temp1 != m1[k][j])
388                     m2[k][j]=i;
389             }
390         }//j
391         k++;
392     ////////////////viterbi algorithm end//////////////////////////////
393     }
394     fclose(feat);
```

7. 이 과정을 반복하면 각 시간(feature vector) k에서 jth word는 어떤 ith word에서 와야 (i->j) 최대 확률값을 갖는지 알 수 있다. 이것을 모두 이어 붙이면 word의 sequence가 만들어진다. word index의 sequence를 저장하는 배열을 int result[nrows]로 선언하자. result 배열의 가장 마지막 원소는 시간 nrow에서 ith word가 될 확률이 가장 큰 i, 즉 m1[nrow][i] 값이 가장 크도록 하는 i가 된다. (405째줄)

그러면 그 앞의 result원소는 result[nrow]를 가지고 m2배열을 사용해서 반복적으로 구할 수 있다. (413, 414째줄)

최종 결과는 result[nrow] 배열을 index로 가지는 word들을 차례로 print하는 것이다. (411, 415째줄)

```
//////print result into file
    //FILE* res;
    //res = fopen("recognized.txt","w");

    int result[nrows];
    int temp1=-10; int temp2=-10;
    for(int i=0;i<wordcount;i++){
        temp2=max(temp1,m1[nrows][i]);
        if(temp1 != temp2) //index update condition
            result[nrows]=i;
        temp1=temp2;
    }
    //fprintf(res,"#!MLF!#\n");
    //fprintf(res,"\"%s.rec\"\n",filename);
    printf("Final Result:\n");
    printf("%s\n",word_hmms[result[nrows]].word);

    for(int c=nrows-1;c>nrows-8;c--){
        result[c]= m2[c][result[c+1]];
        printf("%s\n",word_hmms[result[c]].word);
        //fprintf(res,"%s\n",word_hmms[result[c]].word);
    }
    //fprintf(res,".\n");
    //fclose(res);
}
```

- main() 함수

  이제 main함수의 동작 과정과 위에서 만든 함수들이 쓰이는 방법을 살펴보자.

  1. 제일 먼저 dictionary.txt를 한줄씩 읽어서 각 단어 word가 어떤 phones로 이루어져 있는지 파악한다. (36~58번째 줄) word_phones[i] = token 과 같이 각 단어를 구성하는 phone들을 word_phones 배열에 저장한다.

```
21  int main(){
22      FILE *dict;
23
24      if(!(dict = fopen("dictionary.txt","r"))){
25          printf("Error opening file\n");
26          exit(1);
27      }
28
29      char line[35]; //dict의 각 line
30      word_hmms_type word_hmms[15];
31      int wordcount=0;
32
33      while(fgets(line, sizeof(line), dict)){ //create word hmms
34
35          if(line[strlen(line)-1]=='\n'){
36              line[strlen(line)-1]='\0';
37          }
38
39          char* word_phones[10]={NULL,};  //save phone in word_phones array
40          char* word; //save word
41          int i=-1;
42
43          char * token;
44          token = strtok (line, "' ''\t'"); //seperate each line with " " or "\t"
45
46          while(token != NULL){
47              if(i<0){ //save word
48                  word=token;
49                  i++;
50              }
51              else {
52                  if(token[strlen(token)-1]==13) //carriage return
53                      token[strlen(token)-1]='\0';
54                  word_phones[i] = token; //save phone
55                  i++; //count phone #
56              }
57              token = strtok(NULL,"' ''\t'");
58          }//get word, word_phones
59          int phone_count = i; //number of phones
60
```

2. word_phones 배열에 저장해 놓은 phone들의 hmm에 접근하여 word hmm으로 합치는 작업을 수행한다. 이 때 합쳐질 대상이 되는 hmm은 new_hmm이고 합칠 대상이 되는 hmm의 index를 back_hmm_index라고 하자. 그러면 앞서 정의한 ADD_HMM으로 new_hmm과 back_hmm_index를 넘겨주면 ADD_HMM 함수는 합쳐진 결과 hmm을 return한다. (91줄 ~ 92줄) 과 같이, 합쳐진 결과를 다시 new_hmm으로 할당해준다.

합칠 대상, 즉 back_hmm을 찾는 방법은 전체 phone hmm의 name을 word_phones 배열에 있는 hmm의 name과 비교하여 일치하는 name의 index를 찾는다. (67, 89째줄)

일반적인, N_STATE 개수의 state를 갖는 hmm은 모두 ADD_HMM으로 더한다. 하지만 sp와 같이 state가 1개인 hmm은 ADD_SP 함수를 사용하여 더해준다. (85째줄)

zero word의 경우 두 가지 phone sequence가 존재하므로, z->ih->iy->r->ow->sp 순으로 합쳐지도록 flag를 사용해 순서를 잘 조정해준다. flag=0이면 iy가 합쳐진 후를 의미한다.

(99~102째줄) 하나의 word가 가진 phone들이 모두 합쳐지면 최종 결과를 나타내는 new_hmm을 word_hmms 배열에 저장한다.

```
61      ////combine hmms
62          hmmType* new_hmm=(hmmType*)malloc(sizeof(hmmType));
63          int front_hmm_index;
64          int back_hmm_index; //index of hmm to be combined
65
66          for(int j=0;j<21;j++){ //find first phone hmm as front_hmm
67              if(strcmp(word_phones[0],phones[j].name)==0){
68                  (*new_hmm) = phones[j];
69                  break;
70              }
71          }
72
73          int k=1;
74          int flag=1;
75          while(k < phone_count){ //adding phones one by one using ADD_HMM(hmm index,hmm index)
76              for(i=0;i<21;i++){
77                  if((strcmp(word,"zero")==0)&&(k==4)&&flag){ //zero에 iy state 추가
78                      hmmType* temp_hmm = ADD_HMM(new_hmm,15,k);
79                      (*new_hmm) = (*temp_hmm);
80                      flag=0; k--;
81                      break;
82                  }
83                  if(strcmp(word_phones[k],"sp")==0){//add sp
84                      if(flag==0) k=k+1; //iy가 추가된 경우
85                      hmmType* temp_hmm = ADD_SP(new_hmm,17,k);
86                      (*new_hmm) = (*temp_hmm);
87                      break;
88                  }
89                  if(strcmp(word_phones[k],phones[i].name)==0){
90                      back_hmm_index = i; //find phone to add next to front_hmm
91                      hmmType* temp_hmm = ADD_HMM(new_hmm,back_hmm_index,k);
92                      (*new_hmm) = (*temp_hmm);
93                      break;
94                  }
95              }
96              k++;
97          }
98          //save created word hmm into word_hmms array
99          (word_hmms[wordcount].hmm)=(*new_hmm);
100         (word_hmms[wordcount].word)=strdup(word);
101         word_hmms[wordcount].phnum = phone_count;
102         wordcount++;
103
104     }//finish reading file
105     fclose(dict);
```

3. zero word의 경우 z->iy, iy->r, ow->sp로의 확률이 곱셈을 통해 matrix에 추가되고 iy->sp로의 확률이 0이 되어야 한다. (아래 사진의 110~113줄)

4. 각 단어에 대한 word_hmm들이 모두 생성되고 나면, make_unigram, make_bigram 함수를 차례로 실행시킨다. (116~120) 그리고 마지막으로 Viterbi 함수를 실행시켜 음성 인식을 수행한다. (122~125줄) 이 때 feature vector 값이 들어있는 tst 폴더의 .txt 파일 이름을 char* tempfilename로 설정한 후, Viterbi 함수로 넘겨주면, Viterbi 함수가 해당 directory의 파일에 대한 음성 인식을 수행하게 된다.

```
    wordcount-=1; //zero combined

    //change trans matrix of zero a bit (connect z->iy, iy->r)
    (word_hmms[11].hmm).tp[3][13] = phones[8].tp[3][4] * phones[15].tp[0][1]; //z->iy
    (word_hmms[11].hmm).tp[6][16] = phones[15].tp[3][4] * phones[3].tp[0][1]; //iy->r
    (word_hmms[11].hmm).tp[12][16] = phones[16].tp[3][4] * phones[17].tp[0][1]; //ow->sp
    (word_hmms[11].hmm).tp[15][16] = 0.000000e+000; //disconnect iy->sp

    /////////create unigrams (read unigram.txt)
    word_hmms_type uni_hmms[15];
    make_unigram(wordcount,word_hmms,uni_hmms); //make unigram hmm

    float* Tb = (float*)malloc(sizeof(float)*(wordcount*wordcount));
    Tb = make_bigram(wordcount,word_hmms,Tb); //make trans matrix between word hmms (bigram)

    char* tempfilename="tst/f/ak/1237743.txt"; //file name
    float* features = (float*)malloc(sizeof(float)*(N_DIMENSION)); //feature vector

    Viterbi(tempfilename,wordcount,word_hmms,uni_hmms,Tb,features); //Viterbi Algorithm

    return 0;
}
```

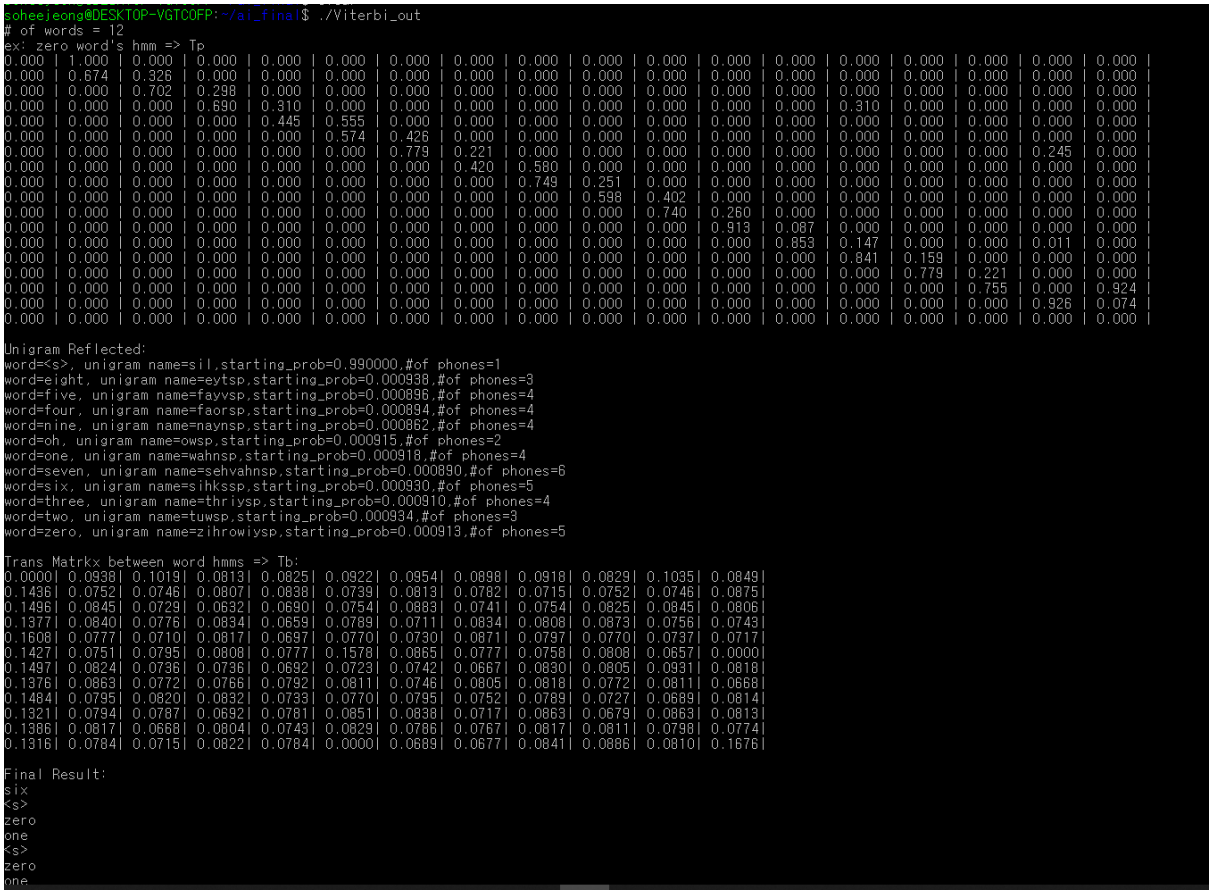
C. 프로그램 실행 방법

math.h 헤더파일을 사용하기 때문에 우분투에서 컴파일 시 아래와 같이 -lm을 붙여서 컴파일 해야 한다. 현재 Viterbi.c 파일에는 각종 hmm과 matrix 등이 잘 만들어졌는지 확인할 수 있도록 printf를 붙여놓았다.

ex)

```
$ gcc -o Viterbi_out Viterbi.c -lm
$ ./Viterbi_out
```

example output)



D. 전체 진행 상황과 소감

1. 구현한 것과 구현하지 못한 것, 개선 사항

전체 과제 진행 상황을 아래 표와 같이 나타냈다. Viterbi Algorithm을 어떻게 구현할 것인지 자세히 플랜을 먼저 설계하지 않고, 직관에 의존해 phone hmm을 word hmm으로 합친 결과, Viterbi Algorithm 구현 부분이 매우 복잡해졌고 결국 Viterbi Algorithm이 제대로 작동하지 않았다.

구현한 것	구현하지 못한 것
dictionary.txt를 읽어서 각 단어가 어떻게 발음되는지 파악한 후,	Viterbi Algorithm이 정확하게 동작하지 않는다.
phone hmm을 결합하여 word hmm을 만들었다.	Viterbi Algorithm에 존재하는 부정확성의 원인을 찾아내지 못하였다.
zero는 두 가지 발음을 가지는데, 하나의 zero word hmm으로 결합하였다.	recognized txt 파일과 reference.txt 파일을 비교하지 못하였다. (HResults.exe가 알 수 없는 오류로 인해 동작하지 않음)
word hmm과 unigram을 사용하여 input 확률이 추가된 unigram hmm을 만들었다.	
Viterbi Algorithm을 구현하고 결과 word	

sequence를 도출하였다.	

2. 고찰

- 이번 과제를 구현하기 위해 공부하고 검색해보고 여러 가지 방법을 시도해보면서, 비록 최종 구현에 성공하지는 못했지만 중요한 것들을 배웠다.
- 먼저 멀게만 느껴졌던 '인공지능 알고리즘 구현'이 조금 더 공부하고 연습하면 나도 구현할 수 있겠다는 자신감을 얻었다. 또한 여러 가지 인공지능 알고리즘의 구현 방법과 인공지능의 동작 원리에 대해 이해할 수 있었다. 수업 수강 이전에는 인공지능은 굉장히 고차원적이고 고급스러운 프로그램과 코드로 동작할 줄 알았지만, 오히려 간결하고 널리 알려진 방법으로도 충분히 구현될 수 있었다.
- 또한 과제를 수행하다가 여러 번 새로 시작하기를 반복하면서, 한 가지 기능만 구현하는 것이 아닌 여러 가지 기능이 복합적으로 맞물려서 수행되는 프로그램을 작성할 때는 먼저 정확히 구체적으로 설계하는 것이 매우 중요함을 알게 되었다.