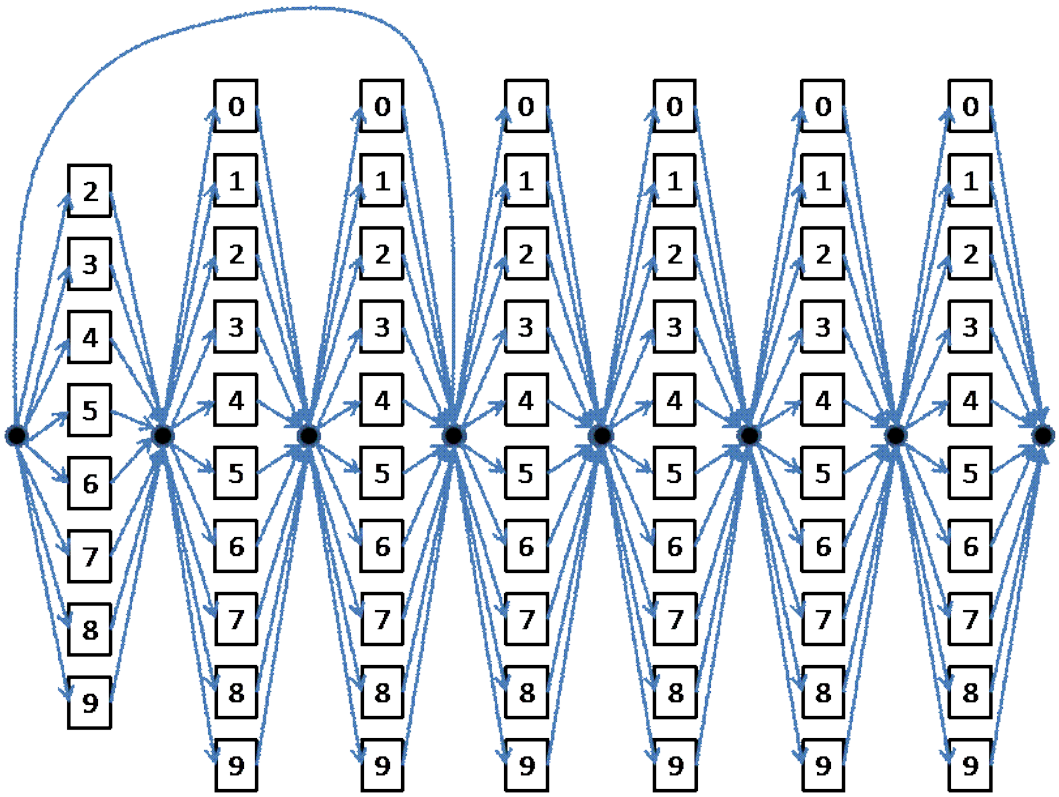
J1799D Project 6 and 7, due date Dec 8th

Project 6:

In this assignment we will write a continuous speech recognition system! Amazing as it may seem, we are ready to do this.

Problem 1: Build a continuous-speech recognition system that can recognize telephone numbers. To do so, connect the digit models you have learned via segmental K-means into the following larger HMM. In the figure each rectangular box enclosing a digit represents the HMM for the digit. Each black dot represents a non-emitting state.



Note that the above figure allows the speaker to say telephone digits numbers either as a 7-digit number or a 4-digit number. Furthermore, we are taking advantage of the fact that the first digit in a telephone number is never 0 or 1.

The grammar for the above problem, expressed in Augmented Backus-Naur format would be as follows:

```
telephone-number = [area-code] number

area-code = first-digit digit digit
```

```
number = digit digit digit digit

first-digit = "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
digit = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
```

The above specification states that a telphone number is a combination of an optional area code and a main number, where the area code is a 3 digit number, where the first digit can be one or 2,3,4,5,6,7,8 or 9, where as the main number is a 4-digit number.

You may find it useful to read in the specification of the word graph (in the figure above) externally. External specifications are often in BNF/ABNF forms such as the one above. However, a simpler format is a *finite state graph* which explicitly characterizes the graph above, e.g. as

```
N_States: 8
Start_State: 0
Terminal_States: 7
Edge 0 1 "2"
Edge 0 1 "3"
Edge 0 1 "4"
...
Edge 1 2 "0"
Edge 1 2 "1"
Edge 1 2 "2"
...
Edge 0 3
Edge 2 3 "0"
Edge 2 3 "1"
Edge 2 3 "2"
..
```

Note that the digits are associated with edges in the graph. Also note the "null" edge from state 0 to state 3 that permits the speaker to skip the area code. In translating this format to a graph, each node becomes a non-emitting HMM state. Each edge in the graph becomes the HMM for a digit.

You can allow for optional pauses at specific locations (e.g. after the 3rd digit) by having simple silence loops such as

```
Edge 3 3 "silence"
```

You will require a specially trained model for "silence" (trained from recordings of silence, possibly) for this model.

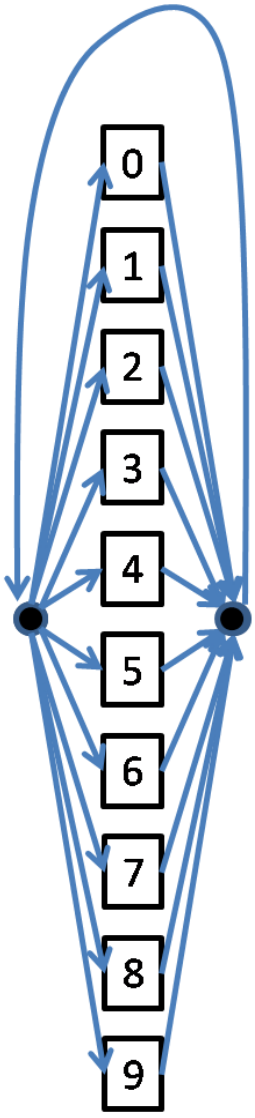
Record 25 valid telephone numbers randomly and report recognition accuracy. Accuracy is reported in terms of:

- Sentence accuracy: How many telephone numbers were recognized correctly.
- Word accuracy: How many digits were recognized correctly.

Do the above with and without pruning and report speed.

Problem 2: Do the above using a backpointer table. Report accuracies as described above.

Problem 3: Build a system to recognize unrestricted digit strings. For this we will use the following graph



In this case it is useful to assign an "insertion penalty" to the loopback to ensure that large numbers of digits are not hypothesized randomly. Determine the

optimal insertion penalty empirically (by evaluating recognition error as given below) and report both the optimal insertion penalty and the recognition performance as required below.

Record the following 10 digit strings in your voice and evaluate accuracy on them. Accuracy is to be measured in terms of:

- Sentence accuracy as before.
- Word error rate. For this perform DTW alignment between the recognition output and the recognized string. The number of errors is the edit distance between the true text and the recognized output. Divide the total number of word errors for all test data and divide by the number of words in the actual (true) strings to obtain error rate.

Test data to record:

- 9 1 1 3 8 5
- 8 2 6 4 1 4 0 5 2 0 0 2
- 8 2 1 2 1 7 6 3 4 2
- 7 3 4 3 3 3 2 1 9 0 3 7 7
- 2 2 1 2
- 1 2 3 4 5 6
- 6 8 9 0 3 7 2 3 4 4
- 7 2 1 8 4 3 4 7 9 2 4
- 5 5 5 5 5
- 3 7 2 7 4 9 2 1

A toy example of how to form a graph. Consider a simple recognizer which recognizes arbitrary sequences of the digits "0" and "1" only. The finite state graph for it can be represented in text form as:

```
N_States: 2
Start_State: 0
Terminal_States: 1
Edge 0 1 "0"
Edge 0 1 "1"
Edge 1 0
```

As mentioned earlier, it is typically convenient to represent the grammar as a graph specified in the above format.

Assume we have HMMs for "0" and "1". The combined HMM can be formed as

```
ngrammarhmmstate = READ(N_states);
startstate = READ(Start_state);
finalstate = READ(Terminal_state);

while ([edge_source_state, edge_end_state, edge_id] = READ(Edge)) do
    if (edge_id) then
        [n_edgestate, edge_tmat, edge_state_meanvar] =
read_hmm(edge_id);
        for i = 1:n_edgestate
            transition_prob [edge_source_state, ngrammhmmstate+i] =
PI_edge_model(i)
        end
        for i = 1:n_edgestate
            transition_prob [ngrammhmmstate+i,edge_end_state] =
tmat_edge_model(i,edge_absobing_state)
        end
        for i = 1:n_edgestate
            grammar_state_meanvar[ngrammarhmmstate+i] =
edge_state_meanvar[i]
        end
        ngrammarhmmstate += n_edgestate - 1;
    else
        transition_prob [edge_source_state,edge_end_state] =
insertion_penalty
    endif
done
```

Note that all we are doing here is literally "inserting" the HMM for the digit in the place of the edge it is associated with. We introduce transitions from the source state of the edge to the entry state(s) of the HMM. We "merge" the final absorbing state with the destination state of the HMM.

This is no longer a true HMM -- the probabilities at some states no longer sum to 1.0. If the finite state grammar had probabilities associated with them (e.g. "Edge 0 1 AND 0.4" states that the word AND occurs on the edge between nodes 0 and 1, with a probability of 0.4) these can be included by associating the probability with entry transitions into the HMM (transition_prob [edge_source_state, ngrammhmmstate+i] = PI_edge_model(i)*PROB)

Project 7:

In this problem we will use recordings from assignment 6 as test data.In this assignment we will train HMMs for digits from continuous speech recordings.

Problem 1: Record each of the following digit sequences five times each. Record the digit "0" as "zero":

```
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
1 2 3 4 5 6 7 8 9 0
0 9 8 7 6 5 4 3 2 1
1 3 5 7 9 0 2 4 6 8
8 6 4 2 0 9 7 5 3 1
```

Record each digit sequence as a continuous recording (without pauses between words). You will now have 30 recordings of each of the digits.

Train models for all ten digits from these continuous recordings. To do so, compose the model for each digit string by concatenating the models for the individual digits. Include silence models on either side, e.g. to model "0 1 2 3 4 5 6 7 8 9", compose the HMM as "sil * 0 * 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * sil" (The "*" here indicates conctenation, and the digits represent their HMMs).

Initialize the HMMs for all digits by the models you learned for them from isolated recordings in previous assignments. For "silence" HMMs, record 5 separate 1-second segments of silence, train an HMM from them and use that to initialize the models trained from the continuous recordings.

Recognize all the continuous digit sequences you recorded for assignment 6 using these models. Report accuracies as reported in assignment 6.

Problem 2: We will now go for a "real" task -- training models from a medium sized corpus of recordings of digit sequences. “http://asr.cs.cmu.edu/spring2014/assignments/assignment6/assign6.data.tar” points to a tar file that includes a portion of the "Aurora2" corpus. In it you will find a "train" directory with 8400 training recordings. The list of filenames for training is in the file TRAIN.list. Their corresponding recordings are in TRAIN.transcripts. The recordings are continuous digit strings, including the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. "0" is recorded both as "zero" and as "oh" (as you will find in the transcriptions). Use these data to train models for all digits. The transcriptions also have the "silence" marked, so you need not explicity add silences at the end of the digit strings. Initialize all models with the models from problem 1. Record a few instances of "oh" and initialize your models for "oh" with those.

The "test" directory contains 1000 test recordings. Each recording is a digit sequence. Use the setup from the second problem of assignment 6 (loopy digits) to recognize these sequences. Report the recognition accuracy as you did for problem 2 of assignment 6.