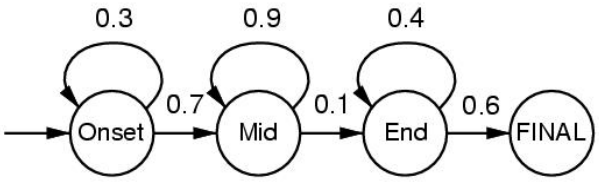
A C++ Implementation of Hidden Markov Model

The package contains three executables:

Executable	Purpose
`vit`	Given an HMM and an observation sequence, compute the sequence of the hidden states that has the highest probability using the Viterbi algorithm.
`genseq`	Generate an observation sequence using a HMM model.
`trainhmm`	Using a collection of observation sequences to train a HMM model using the Baum-Welch algorithm.

The three executables share the same implementation of HMM in `hmm.h` and `hmm.cpp`. The executables can be computed by typing the `make` command in the `hmm` directory.

A HMM is specified in two files: `NAME.trans` and `NAME.emit` where `NAME` is the name of the HMM. The file `NAME.trans` contains the transition probabilities between the states. The file `NAME.emit` contains the emission probabilities. Normally, HMM also needs a set of initial probabilities of the states. We treat them as part of the transition probabilities by adding a special initial state. The transition probabilities from the special initial state to other states correspond to the initial probabilities of states.



Output probabilities for the phone HMM:

Onset:	Mid:	End:
C1: 0.5	C3: 0.2	C4: 0.1
C2: 0.2	C4: 0.7	C6: 0.5
C3: 0.3	C5: 0.1	C7: 0.4

Consider the above HMM (from Russell and Norvig's AI textbook). Its transition and emission probabilities are specified in the files `phone.trans` and `phone.emit` in the `phone` directory as follows:

`phone.trans:`

```
INIT
INIT    Onset   1
Onset   Onset   0.3
Onset   Mid     0.7
```

```
Mid     Mid     0.9
Mid     End     0.1
End     End     0.4
End     FINAL   0.6
```

`phone.emit:`

```
Onset   C1      0.5
Onset   C2      0.2
Onset   C3      0.3
Mid     C3      0.2
Mid     C4      0.7
Mid     C5      0.1
End     C4      0.1
End     C6      0.5
End     C7      0.4
```

The first line of the `phone.trans` file is the name of the initial state. Each of the subsequent lines is a transition probability. For example, `P(Mid|Onset)=0.7`, `P(C4|End)=0.1`. The transition and emission probabilities not listed in these two files are treated as zeros.

The `vit` program takes the name of a HMM as the command-line argument. It then reads sequences of observations from the standard input and prints the most probable sequences of states as well as their probability on the standard output. For example, the file `phone.input` contains the observation sequence:

```
C1 C2 C3 C4 C4 C6 C7
C2 C2 C5 C4 C4 C6 C6
```

The results of issuing the following command in the `phone` directory:

```
../src/vit phone < phone.input
```

are the following:

```
P(path)=0.625286
path:
C1      Onset
C2      Onset
C3      Mid
C4      Mid
C4      Mid
C6      End
C7      End
P(path)=0.936748
path:
C2      Onset
C2      Onset
C5      Mid
C4      Mid
C4      Mid
C6      End
C6      End
```

The `genseq` program takes two parameters. The first is the name of a HMM (i.e., `NAME.trans` and `NAME.emit` specifies the transition and emission probabilities of the HMM). The second is the number of sequences to generate. The program generates a collection of observation sequences with each sequence on a line. For example, the outputs of the command

```
../src/genseq phone 10
```

are:

```
C1 C4 C6 C7 C6
C1 C5 C7
C2 C1 C3 C1 C1 C4 C4 C4 C4 C5 C4 C4 C4 C4 C5 C5 C3 C4 C4 C3 C3 C5 C7 C6 C7
C2 C1 C5 C4 C4 C4 C4 C4 C4 C4 C5 C4 C4 C6
C3 C4 C7 C7
C2 C3 C4 C3 C4 C3 C6 C6
C3 C4 C4 C4 C4 C4 C4 C4 C5 C4 C4 C4 C5 C4 C3 C3 C4 C4 C4 C3 C6
C3 C3 C1 C4 C3 C5 C4 C4 C4 C4 C4 C6
C2 C3 C5 C4 C7
C2 C3 C4 C4 C4 C4 C4 C3 C4 C4 C6
```

One of the beauties of HMM is that the parameters it needs can be estimated (trained) with sequences of observations. The `trainhmm` program does exactly this. It takes three obligatory parameters and one optional parameter. The three obligatory parameters are: the name of the initial HMM, the name of the result HMM and the file containing the training sequences. The optional parameter is the maximum number iterations to run during training. If the fourth parameter is not provided, the maximum number of iterations is 10. For example, the command:

```
../src/trainhmm phone-init1 phone-result1 phone.train
```

will train a HMM with the starting parameters in the files `phone-init1.trans` and `phone-init1.emit` which contain the following contents:

`phone-init1.trans:`

```
INIT
INIT    Onset   1
Onset   Onset   0.5
Onset   Mid     0.5
Mid     Mid     0.5
Mid     End     0.5
End     End     0.5
End     FINAL   0.5
```

`phone-init1.emit:`

```
Onset   C1      0.33
Onset   C2      0.33
Onset   C3      0.33
Mid     C3      0.33
Mid     C4      0.33
Mid     C5      0.33
End     C4      0.33
End     C6      0.33
End     C7      0.33
```

By default, the `trainhmm` command runs up to 10 iterations. This can be changed by providing the program the the fourth parameter.

Although the transitions in `phone-init1.trans` have different probabilities than the model that generated the data, the transition diagram would have the same structure as the model. Now, suppose we do not know the structure of the transition diagram. We would have to assume any state (including FINAL, but excluding INIT) can follow any other state with equal probability. The transition probabilities will be as specified in `phone-init2.trans`. Suppose `phone-init2.emit` is identical to `phone-init1.emit`. The results `trainhmm` program with `phone-init2` show that the Baum-Welch algorithm can still learn the correct HMM.

To make the learning problem even harder, if you change the initial emission probability table so that any state (including FINAL, but excluding INIT) can generate any symbol with equal probability (see `phone-init3.*`), the Baum-Welch will not be able to learn the correct model.

The `pos` directory contains a part of speech tagger trained with about 41K sentences from the Wall Street Journal (the corpus is not included for copyright reasons). The initial transition probability table allow any state (POS) to follow any other state with equal probability. The initial emission probability table is based on the lexicon in Michael Collins' parser. We assume that all emissions allowed in the lexicon have equal probability. The format of the training corpus should be the same as the file `sample.txt`: each line corresponds to a sentence. The tokens are space separated.

The files `pos.trans` and `pos.emit` are transition and emission probability tables obtained with the 41K sentence corpus. One can use the Viterbi algorithm to perform POS tagging with this model. For example, the command

```
../src/vit pos <sample.txt
```

generates the following outputs:

```
P(path)=0.443872
path:
But     CC
state   NN
courts  NNS
upheld  VBD
a       DT
challenge       NN
by      RP
consumer        NN
groups  NNS
to      TO
the     DT
commission      NN
's      POS
rate    NN
increase        NN
and     CC
found   VBD
the     DT
rates   NNS
illegal JJ
.       .
P(path)=0.580858
path:
The     DT
Illinois        NNP
Supreme NNP
Court   NNP
ordered VBD
the     DT
commission      NN
to      TO
audit   VB
Commonwealth    NNP
Edison  NNP
's      POS
construction    NN
expenses        NNS
and     CC
refund  VB
any     DT
unreasonable    JJ
expenses        NNS
.       .
```