Transform HMMs into FSTs

Assume you are given a Hidden Markov Model (HMM) described by a transition matrix A and an emission matrix B, with n states and m symbols. Assume state 0 is the start state.

Write a function that converts the Hidden Markov Model into an equivalent FST and demonstrate that the two models give equivalent results.

In [ ]:
```
def hmm_to_fst(A,B):
    """Convert an HMM to an equivalent FST. State 0
    is always the start state, and state n-1 is always
    the accept state."""
    n,n1 = A.shape
    assert n==n1
    m,n2 = B.shape
    assert n==n2
    if accept is None: accept_state = n-1
    raise Exception("IMPLEMENT ME")
    return fst
```

For the following, just call the OpenFST functions.

In [3]:
```
def fst_log_probability(fst,s):
    """Find the lowest cost path corresponding to the string `s`
    through `fst`. The string `s` is given as a 1D numpy array."""
    raise Exception("IMPLEMENT ME")
    return log_p
```

For the following, you can use the Viterbi algorithm from lecture as a starting point.

In [4]:
```
def hmm_log_probability(A,B,s):
    """Find the lowest cost path through the HMM corresponding
    to the given string `s`."""
    raise Exception("IMPLEMENT ME")
    return log_p
```

Now write some unit tests.

In [ ]:
```
A = ones((1,1))
B = ones((1,1))
assert abs(hmm_log_probability(A,B,zeros(10,'i'))) < 1e-4

fst = hmm_to_fst(A,B)
assert abs(fst_log_probability(fst,zeros(10,'i'))) < 1e-4

# more meaningful unit tests
```

And write some general purpose random tests.

In [ ]:
```
for trial in range(10):
    A = rand((10,10))
    # normalize appropriately
    B = rand((17,10))
    # normalize appropriately
    fst = hmm_to_fst(A,B)
    for trial2 in range(10):
        s = array(10*rand(7),'i')
        p1 = fst_log_probability(fst,s)
        p2 = hmm_log_probability(A,B,s)
        assert abs(p1-p2)/min(abs(p1),abs(p2)) < 1e-4
```