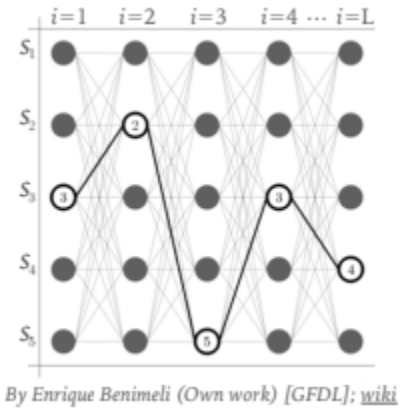
Nathan Newbury Hw5 Writeup:

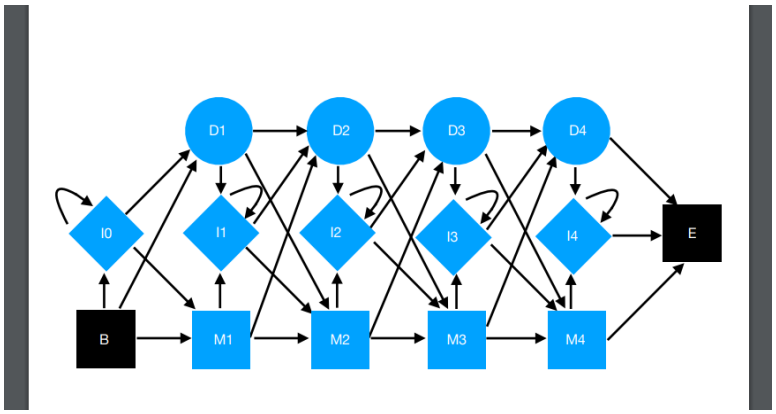
Time Complexity:

I assume that our average query is of length n and the number of states is m. As in any dynamic programing problem our time complexity is O(|E| + V*d) where E is the number of edges, V is the number of vertices and d is the time spent at each Vetrice. Again, because at each vertice we are only taking a maximum and saving an arrow, we will assume constant subproblem time. This means the time complexity can be approximated as O(|E|), or the number of edges. The normal Viterbi Algorithm has a time complexity of m^2*n. This can be seen with the graph below taken from the slides.



By Enrique Benimeli (Own work) [GFDL]; wiki

In this case, each state is connected to each other through an edge. For each iteration (n total iterations) we find m^2 connections, which makes a total of O(m^2*n) edges. In our Viterbi algorithm every state connected to 3 other states. Below is an example model:



Each state is connected to I, M, or D. In theory this yields a time complexity of number of edges |E| = **3*3*n or O(3*m*n).** However, initializing the 3 matrices to -inf at the start of the algorithm could take more time with a time complexity of **O(3*j*n)**. J is the number of layers of states, and n is the length of the query. J should not be confused with m which is still 3 in this case.

For the Forward algorithm, the time complexity is the same. Assuming constant lookup and store times, we can expect a worst case time complexity of O(m^2*n) to a best case time complexity of O(m*n) depending on the connectivity of the states in the model.

Recursive Formula for Forward:

```
   #import pdb; pdb.set_trace()
                    try:
                        FM[i][j] = self.eM[j][query[i-1]] +
log(exp(FM[i-1][j-1] + self.t[j-1]['MM']) + exp(FI[i-1][j-1] +
self.t[j-1]['IM']) +  exp(FD[i-1][j-1] + self.t[j-1]['DM']))
                    except:
                        FM[i][j] = -INF
                if i>=1:
                    try:
                        FI[i][j] = self.eI[j][query[i-1]] +
log(exp(FM[i-1][j] +self.t[j]['MI']) + exp(FI[i-1][j] + self.t[j]['II']) +
exp(FD[i-1][j] + self.t[j]['DI']))
                    except:
                        FI[i][j] = -INF

                if j>=1:
                    try:
                        FD[i][j] = log(exp(FM[i][j-1] + self.t[j-1]['MD'])
+ exp(FI[i][j-1] + self.t[j-1]['ID']) + exp(FD[i][j-1] +
self.t[j-1]['DD']))
                    except:
                        FD[i][j] = -INF
```

The forward algorithm operates similarly to the Viterbi algorithm except instead of finding the most probable path, it records the total likelihood of being in a certain state at a certain iteration. Instead of calculating the maximum, we record the log sum of all transition probabilities from previous states to this state at this iteration.

Classification task:

We use the maximum final score of either our forward algorithm or Viterbi algorithm to determine the best model. The Viterbi algorithm finds the most probable path through the hidden states. This means that Viterbi finds the most probable combination of states and leads to the most probable single alignment. A higher Viterbi score means a more probable “most” probable path. The forward algorithm determines the likelihood of being in a certain state at a certain

iteration. For example, the most probable path yields an example sequence of AcAT, however it still could be that C in isolation is more likely a match if you consider every incoming path. For evaluation purposes (finding the right model from observations), we use the forward algorithm to maximize the probability each individual observation is emitted from a certain state.