```r
################################################################################
# THE VITERBI IMPLEMENTATION
#
# function: viterbi()
#
# parameter: model, a list containing
#   States: a vector with the names of the states
#   Symbols: a vector with the names of the symbols
#   startProbs: a vector with the starting probabilities of the states
#   transProbs: a matrix containing the transition probabilities between the states
#   emissionProbs: a matrix containing the emission probabilities of the states
#
# return value: a matrix containing the logs of probabilities of each of the states
#   for each observation.  These values are only useful relative to one another.
################################################################################
viterbi <- function(model, obs)
{
  # Create a matrix of probabilities that we will return.
  prob <- matrix(0, length(model$States), length(obs))

  # Loop through each of the model states.
  # In this loop, we only process the initial observation
  for (j in 1:length(model$States))
  {
    # Find the index of the symbol from the observation.
    y <- which(model$Symbols == obs[1])
    # If we don't find it, it's because we have encountered a symbol that the model
    # was not trained on.  In that case, we cannot proceed and return NULL.
    if (length(y) == 0) { return(NULL) }

    # Record the probability of each state for the first observation.  This is the
    # product of the prior stating probability of the state and the prior emission
    # probability of the symbol.
    prob[j, 1] <- log(model$startProbs[j]) + log(model$emissionProbs[j, y])
  }

  # Loop through all subsequent observations.
  for (i in 2:length(obs))
  {
    # Find the index of the symbol from the current observation.
    y <- which(model$Symbols == obs[i])
    # As above, if we don't find it, return NULL.
    if (length(y) == 0) { return(NULL) }

    # For each model state
    for (j in 1:length(model$States))
    {
      # Calculate the sums of logs using (1) the probability of each state in the previous
      # iteration/observation (already a log), (2) the probability of transitioning from
      # that previous state to that of the current observation, and (3) the emission
      # probability of the symbol encountered in the current iteration/observation.
      sums <- prob[, i-1] + log(model$transProbs[, j]) + log(model$emissionProbs[j, y])

      # Record the probability of each state for the first observation.  This is the
      # sum of the logs of the prior stating probability of the state and the log of the
      # prior emission probability of the symbol.
      prob[j, i] <- max(sums)
    }
  }

  # Return the log probabilities calculated for each state at each observation.
  # Typically, the caller will only be interested in the maximum value in the final
  # column as that is the most probable final state.
  return(prob)
}
```