Problem 1: Suppose that at time t the hidden state is $c^{(t)}$. Then we can compute the probability that the next hidden state is $c^{(t+1)}$ and the next observed word is $w^{(t+1)}$. This probability is equal to:

$$\begin{aligned}&\Pr(W^{(t+1)} = w^{(t+1)}, C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) \\ =&\Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)})\end{aligned}$$

Write a procedure `score-next-state-word` which computes the log of this probability. It should take five arguments: the current hidden state, the next hidden state, and next observed word, the hidden state transition distributions, and the observation distributions. A call to (`score-next-state-word current-hidden next-hidden next-observed theta-transition-dists theta-observation-dists`) should return the log probability of the next hidden state and next observed word, given the current hidden state.

Hint: use the functions `dist-lookup` and `logscore-categorical` defined above.

```
(defn score-next-state-word [current-hidden next-hidden next-observed
 theta-transition-dists theta-observation-dists]
 (+
 (logscore-categorical next-observed vocabulary
  (dist-lookup next-hidden hidden-states theta-observation-dists))
 (logscore-categorical next-hidden hidden-states
  (dist-lookup current-hidden hidden-states theta-transition-dists))))
```

Problem 2: Suppose that at time t the hidden state is $c^{(t)}$. Then we can compute the *marginal* probability of the word $w^{(t+1)}$ at time $t+1$ given the hidden state at time t. This probability is equal to:

$$\begin{aligned}&\Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t)} = c^{(t)}) \\ =&\sum_{c^{(t+1)}} \Pr(W^{(t+1)} = w^{(t+1)}, C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) \\ =&\sum_{c^{(t+1)}} \Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)})\end{aligned}$$

That is, we sum over all of the possible hidden states that could appear at time $t+1$.

Using the procedure `score-next-state-word` that you wrote in Problem 1, write a function `compute-next-observation-marginal`, which takes four arguments: `current-state`, `next-`

observation, theta-transition-dists, and theta-observation-dists. When (compute-next-observation-marginal current-state next-observation theta-transition-dists theta-observation-dists) is called, it should return the marginal probability of the next observed word given the current hidden state.

Hint: use logsumexp.

Answer 2:

```
(defn compute-next-observation-marginal [current-state next-observation
  theta-transition-dists theta-observation-dists]
  (logsumexp (map (fn [x] (score-next-state-word current-state x next-observation
    theta-transition-dists theta-observation-dists))
      (rest hidden-states))))
```

Problem 3: Suppose that at time t the hidden state is $c^{(t)}$. Then we can compute the probability of the next k hidden states and next k observed words given the hidden state at time t. This probability is equal to:

$$
\begin{aligned}
&\Pr(W^{(t+1)}, \ldots, W^{(t+k)}, C^{(t+1)}, \cdots, C^{(t+k)} \mid C^{(t)} = c^{(t)}) \\
&= \prod_{i=t}^{t+k-1} \Pr(W^{(i+1)} = w^{(i+1)} \mid C^{(i+1)} = c^{(i+1)}) \Pr(C^{(i+1)} = c^{(i+1)} \mid C^{(i)} = c^{(i)}) \\
&= \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) \Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \\
&\quad \cdot \Pr(W^{(t+2)}, \ldots, W^{(t+k)}, C^{(t+2)}, \cdots, C^{(t+k)} \mid C^{(t+1)} = c^{(t+1)})
\end{aligned}
$$

Note that the equation shows that there is a recursive formula for this probability.

Write a procedure score-next-states-words which computes this probability. The function should take five arguments: the current hidden state, a list of k hidden states, a list of k observed words, the transition distributions, and the observation distributions. When (score-next-states-words current-hidden next-hidden-states next-words theta-transition-dists theta-observation-dists) is called, it should return the log probability that the k hidden states and k observed words will appear immediately following the current hidden state.

You should use recursion to write the function.

```
(defn score-next-states-words [current-hidden next-hidden-states next-words
  theta-transition-dists theta-observation-dists]
  (if (empty? next-words)
```

```
0
(+
(score-next-state-word current-hidden (first next-hidden-states) (first next-words)
  theta-transition-dists theta-observation-dists)
(score-next-states-words (first next-hidden-states) (rest next-hidden-states)
  (rest next-words) theta-transition-dists theta-observation-dists))))
```

Problem 4: Suppose the at time t the hidden state is $c^{(t)}$. Then we can compute the marginal probability of the next k observed words given this hidden state, i.e. summing out all of the possible settings of the hidden states. This probability is equal to:

$$\begin{aligned}
&\Pr(W^{(t+1)}, \ldots, W^{(t+k)} \mid C^{(t)} = c^{(t)}) \\
&= \sum_{c^{(t+1)}, \cdots c^{(t+k)}} \Pr(W^{(t+1)}, \ldots, W^{(t+k)}, C^{(t+1)}, \cdots, C^{(t+k)} \mid C^{(t)} = c^{(t)}) \\
&= \sum_{c^{(t+1)}, \cdots c^{(t+k)}} \prod_{i=t}^{t+k} \Pr(W^{(i+1)} = w^{(i+1)} \mid C^{(i+1)} = c^{(i+1)}) \Pr(C^{(i+1)} = c^{(i+1)} \mid C^{(i)} = c^{(i)}) \\
&= \sum_{c^{(t+1)}} \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) \Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \\
&\quad \cdot \Pr(W^{(t+2)}, \ldots, W^{(t+k)}) \mid C^{(t+1)} = c^{(t+1)})
\end{aligned}$$

The final equation is very important. It shows that the marginal probability of the words following hidden state $c^{(t)}$ can be computed in a recursive manner. Write a function `compute-next-words-marginal`, which computes the log marginal probability of a sequence of observed words given the current hidden state. The function should take four arguments: the current hidden state, a list of words, the transition distributions, and the observation distributions. When `(compute-next-words-marginal current-hidden next-words theta-transition-dists theta-observation-dists)` is called, it should return the log of the marginal probability of the list of words given the hidden state, as defined in the equation above.

You should use recursion to write your function.

```
(defn compute-next-words-marginal [current-hidden next-words theta-transition-dists
  theta-observation-dists]
  (if (empty? next-words)
    0
    (logsumexp
    (list
    (+ (score-next-state-word current-hidden (first (rest hidden-states))
```

```
    (first next-words) theta-transition-dists theta-observation-dists)
    (compute-next-words-marginal (first (rest hidden-states)) (rest next-words)
      theta-transition-dists theta-observation-dists))
  (+ (score-next-state-word current-hidden (first (rest (rest hidden-states)))
    (first next-words) theta-transition-dists theta-observation-dists)
    (compute-next-words-marginal (first (rest (rest hidden-states)))
      (rest next-words) theta-transition-dists theta-observation-dists))))))
```

Problem 5: Find a sequence of two words $w^{(1)}, w^{(2)}$ such that the marginal probability of this sequence is different than the marginal probability of its reverse. That is, find $w^{(1)}, w^{(2)}$ such that:

$$\Pr(W^{(1)} = w^{(1)}, W^{(2)} = w^{(2)} \mid C^{(0)} = \mathit{Start}) \neq \Pr(W^{(1)} = w^{(2)}, W^{(2)} = w^{(1)} \mid C^{(0)} = \mathit{Start})$$

Use your solution to Problem 4 to confirm that your sequences have different probabilities, and explain why this is true.

```
(compute-next-words-marginal
  'Start '(Call me) theta-transition-dists-1 theta-observation-dists-1)
;; -4.227016447861896
(compute-next-words-marginal
  'Start '(me Call) theta-transition-dists-1 theta-observation-dists-1)
;; -1.9002335137070705
(compute-next-words-marginal
  'Start '(me Ishmael) theta-transition-dists-1 theta-observation-dists-1)
;; -3.4916727707196795
(compute-next-words-marginal
  'Start '(Ishmael me) theta-transition-dists-1 theta-observation-dists-1)
;; -3.583808806104786
```

The left side of the equation marginalises over the possible states. The right side of the equation marginalises over the possible states and the possible previous states. That is why these two are not equal.

Problem 6: In the next several problems, we will be using your solution to Problem 4 to perform Bayesian inference in the HMM. Our goal will be to compute the posterior distribution over hidden states given a sequence of words. Recall that Bayes' Rule tells us how to calculate this posterior distribution:

$$\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(t)} = c^{(t)} \mid W^{(1)} = w^{(1)}, \ldots, W^{(t)} = w^{(t)}) =$$
$$\frac{\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(t)} = c^{(t)}) \Pr(W^{(1)} = w^{(1)}, \ldots, W^{(t)} = w^{(t)} \mid C^{(1)} = c^{(1)}, \cdots, C^{(t)} = c^{(t)})}{\Pr(W^{(1)} = w^{(1)}, \ldots, W^{(t)} = w^{(t)})}$$

Write a procedure `compute-hidden-prior`, which takes two arguments: a list of k hidden states and the hidden state transition distributions. When `(compute-hidden-prior hidden-states theta-transition-dists)` is called, it should return the log prior probability of the hidden state sequence, i.e. the log value of

$$\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(k)} = c^{(k)})$$

```
(defn compute-hidden-prior [k thetas]
  (if (= (count k) 1)
    (logscore-categorical (first k) hidden-states (first thetas))
    (+ (logscore-categorical (first (rest k)) hidden-states
      (dist-lookup (first k) hidden-states thetas))
       (compute-hidden-prior (rest k) thetas))))
```

Problem 7: In this problem, we will continue with the Bayesian calculation from Problem 6. Write a procedure `compute-likelihood-of-words`, which takes three arguments: a list of k hidden states, a list of k words, and the observation distributions. When `(compute-likelihood-of-words hidden-states words theta-observation-dists)` is called, it should return the log probability of the k words being generated from the hidden states, i.e. the log value of $\Pr(W^{(1)} = w^{(1)}, \ldots, W^{(k)} = w^{(k)} \mid C^{(1)} = c^{(1)}, \cdots, C^{(k)} = c^{(k)})$.

```
(defn compute-likelihood-of-words [k words thetas]
  (if (empty? k)
    0
    (+ (logscore-categorical (first words) vocabulary
      (dist-lookup (first k) hidden-states thetas))
    (compute-likelihood-of-words (rest k) (rest words) thetas))))
```

Problem 8: We are finally ready to solve our Bayesian inference problem. Write a procedure `compute-hidden-posterior`, which takes four arguments: a list of k hidden states, a list of k words, the hidden state transition distributions, and the observation distributions. When `(compute-hidden-posterior hidden-states words theta-transition-dists`

`theta-observation-dists`) is called, it should return the log posterior probability of the k hidden states given the k words, i.e. it should return the log value of $\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(k)} = c^{(k)} \mid W^{(1)} = w^{(1)}, \ldots, W^{(k)} = w^{(k)})$.

Hint: Use your solutions to Problems 4, 6 and 7.

```
(defn compute-hidden-posterior [k words transition observation]
  (- (+ (compute-hidden-prior k transition)
     (compute-likelihood-of-words k words observation))
   (compute-next-words-marginal 'Start words transition observation)))
```

Problem 9: Suppose that we use the function `compute-next-words-marginal` that you wrote in Problem 4, in order to compute the marginal probability of a sequence of k observed words. How many times is `compute-next-words-marginal` called during the execution of the program? What does this mean about the run-time of the program?

Hint: your formula should involve the number of hidden states and the length k of the sequence of words.

Answer 9: `https://web.stanford.edu/~jurafsky/slp3/A.pdf`

This, in my opinion, would be considered **external research**, and I am citing my source.

As suggested in the textbook above, we can take a simple case of the number of ice creams eaten in a day (O for observation) versus the 'hidden' sequence Q of weather states (hot H or cold C days).

$$\begin{aligned}\Pr(O) &= \sum_{Q} \Pr(O, Q) \\ &= \sum_{Q} [\Pr(O|Q) \cdot \Pr(Q)]\end{aligned}$$

Let's say that we wanted to find $\Pr(3-1-3)$. We would have to sum over all eight instances of where $\Pr(3-1-3)$ is seen.

In general for an HMM with C being the set of hidden states or categories and a sequence of length k, there are $|C|^k$ possible hidden sequences. Therefore, the runtime of `compute-next-words-marginal` is $\mathcal{O}(|C|^k)$

Problem 10: In class, we introduced *memoization* as a method for sharing computation and/or reducing the run-time of programs. Memoization refers to a technique where we intercept calls to a function with some arguments, check if the function has been called with those arguments before and, if not, call the function on those arguments, get the result, store it in a table and return it. On subsequent calls to that function with those arguments, we simply look up the result in the table and avoid recomputing the value. In Clojure, functions can be memoized by a call to `memoize`.

Using `memoize`, define a memoized version of your function `compute-next-words-marginal`. Name this function `compute-next-words-marginal-mem`.

```
(def compute-next-words-marginal-mem (memoize (fn
  [current-hidden next-words theta-transition-dists
    theta-observation-dists]
  (if (empty? next-words)
    0
    (logsumexp
    (list
    (+ (score-next-state-word current-hidden (first (rest hidden-states))
      (first next-words) theta-transition-dists theta-observation-dists)
      (compute-next-words-marginal (first (rest hidden-states)) (rest next-words)
        theta-transition-dists theta-observation-dists))
    (+ (score-next-state-word current-hidden (first (rest (rest hidden-states)))
      (first next-words) theta-transition-dists theta-observation-dists)
      (compute-next-words-marginal (first (rest (rest hidden-states)))
        (rest next-words) theta-transition-dists theta-observation-dists)))))))
```

Problem 11: Suppose that we use the memoized function `compute-next-words-marginal-mem` that you wrote in the previous problem, in order to compute the marginal probability of a sequence of k observed words. How many times is `compute-next-words-marginal-mem` called during the execution of the program? How does this differ from the non-memoized version of the procedure?

Answer 11: https://web.stanford.edu/~jurafsky/slp3/A.pdf

This, in my opinion, would be considered **external research**, and I am citing my source.

From the problem above, "memoization refers to a technique where we intercept calls to a function with some arguments, check if the function has been called with those arguments before and, if not, call the function on those arguments, get the result, store it in a table and return it." As suggested in the textbook above, memoization, or storing the values in a table, is a form of **Dynamic Programming**. In this particular case, the algorithm is referred

to as the **Forward Algorithm**. The runtime of the **Forward Algorithm** is $\mathcal{O}(|C|^2 k)$. C is the set of hidden states or categories, and k is the length of the sequence.

From a completely Computer Science Point of View, the **Forward Algorithm** allows us to run a procedure in *Polynomial Time* when it would have taken *Exponential Time* using **Brute Force**.