function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix $viterbi[N+2,T]$
for each state s **from** 1 **to** N **do** ; initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

Figure 6.11 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence. Note that states 0 and q_F are non-emitting.

1. **Initialization:**

$$v_1(j) = a_{0j}b_j(o_1) \quad 1 \le j \le N \qquad (6.20)$$
$$bt_1(j) = 0 \qquad (6.21)$$

2. **Recursion** (recall that states 0 and q_F are non-emitting):

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\,a_{ij}\,b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \qquad (6.22)$$
$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\,a_{ij}\,b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \qquad (6.23)$$

3. **Termination:**

$$\text{The best score: } P* = v_t(q_F) = \max_{i=1}^{N} v_T(i) * a_{i,F} \qquad (6.24)$$
$$\text{The start of backtrace: } q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{i,F} \qquad (6.25)$$

6.5 HMM Training: The Forward-Backward Algorithm

We turn to the third problem for HMMs: learning the parameters of an HMM, that is, the A and B matrices. Formally,

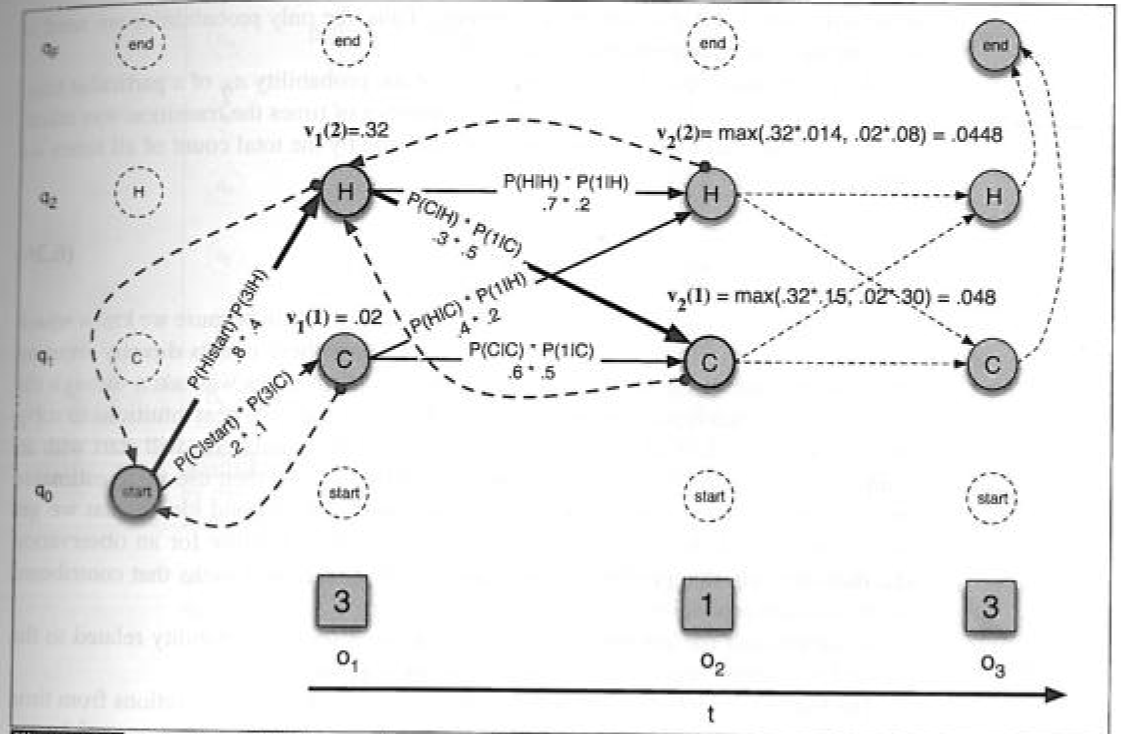

Figure 6.12 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

> **Learning:** Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B.

The input to such a learning algorithm would be an unlabeled sequence of observations O and a vocabulary of potential hidden states Q. Thus, for the ice cream task, we would start with a sequence of observations $O = \{1,3,2,...,\}$ and the set of hidden states H and C. For the part-of-speech tagging task, we would start with a sequence of observations $O = \{w_1, w_2, w_3 \ldots\}$ and a set of hidden states *NN, NNS, VBD, IN,...* and so on.

Forward-backward *Baum-Welch* *EM*

The standard algorithm for HMM training is the **forward-backward**, or **Baum-Welch** algorithm (Baum, 1972), a special case of the **Expectation-Maximization** or **EM** algorithm (Dempster et al., 1977). The algorithm will let us train both the transition probabilities A and the emission probabilities B of the HMM.

Let us begin by considering the much simpler case of training a Markov chain rather than a hidden Markov model. Since the states in a Markov chain are observed, we can run the model on the observation sequence and directly see which path we took through the model and which state generated each observation symbol. A Markov chain of course has no emission probabilities B (alternatively, we could view a Markov chain as a degenerate hidden Markov model where all the b probabilities are 1.0 for the