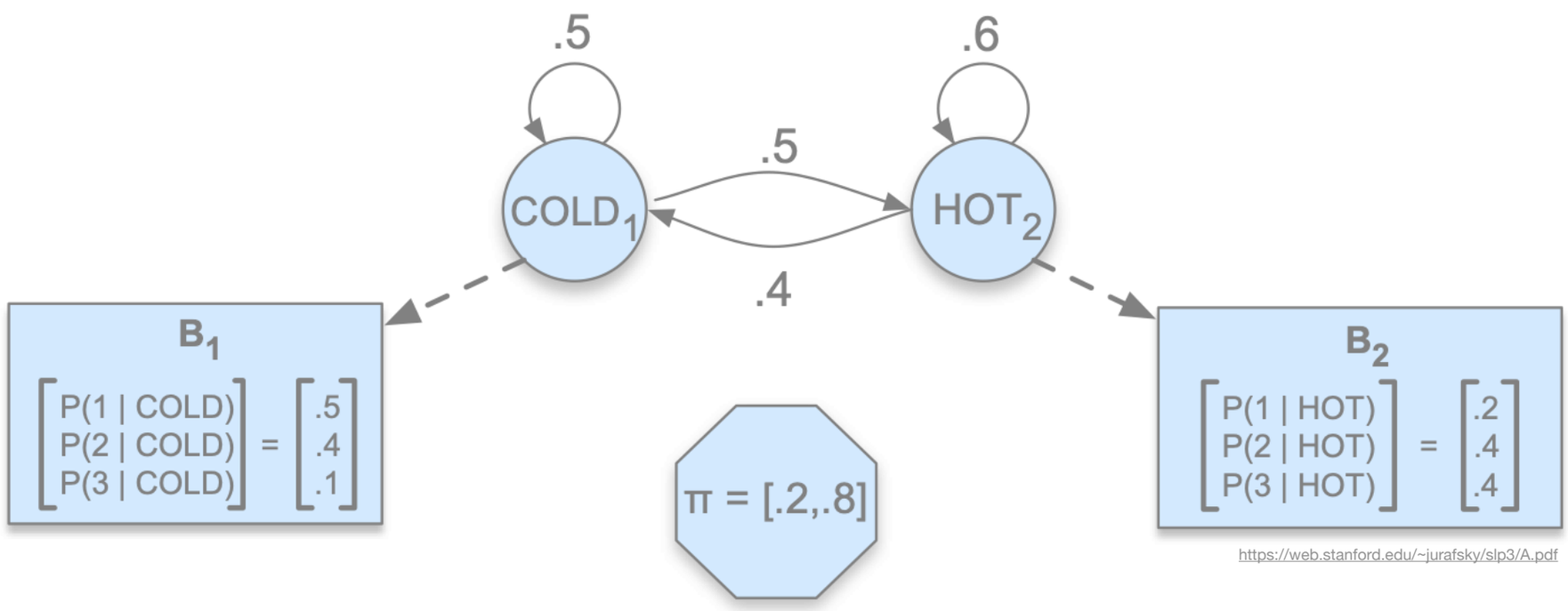
Hidden Markov Models (HMMs)



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

Markov chains

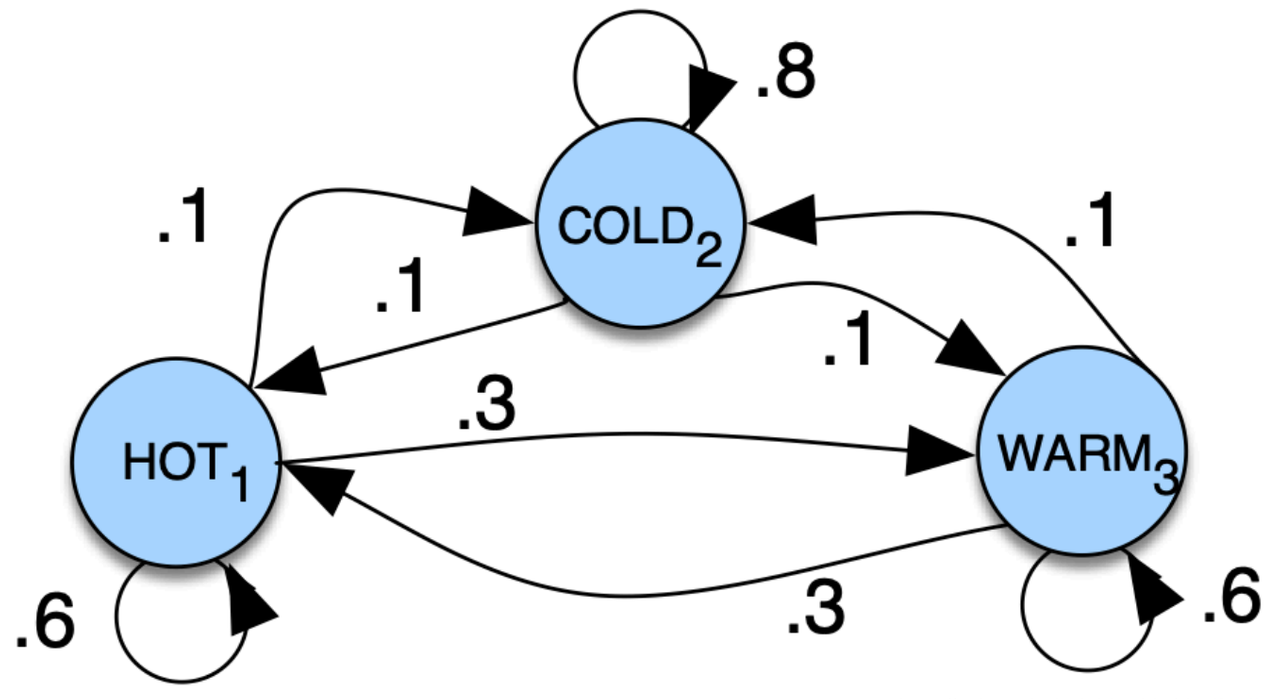
State space: A

Random variable: $X_n \in A,\ n \in \mathbb{N}$

Sequence: $(X_1, X_2, X_3, \ldots)$

Markov property:

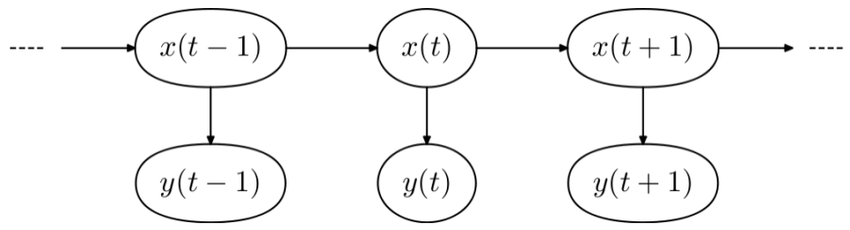
$$P(X_{n+1} = x | X_1 = x_1, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$



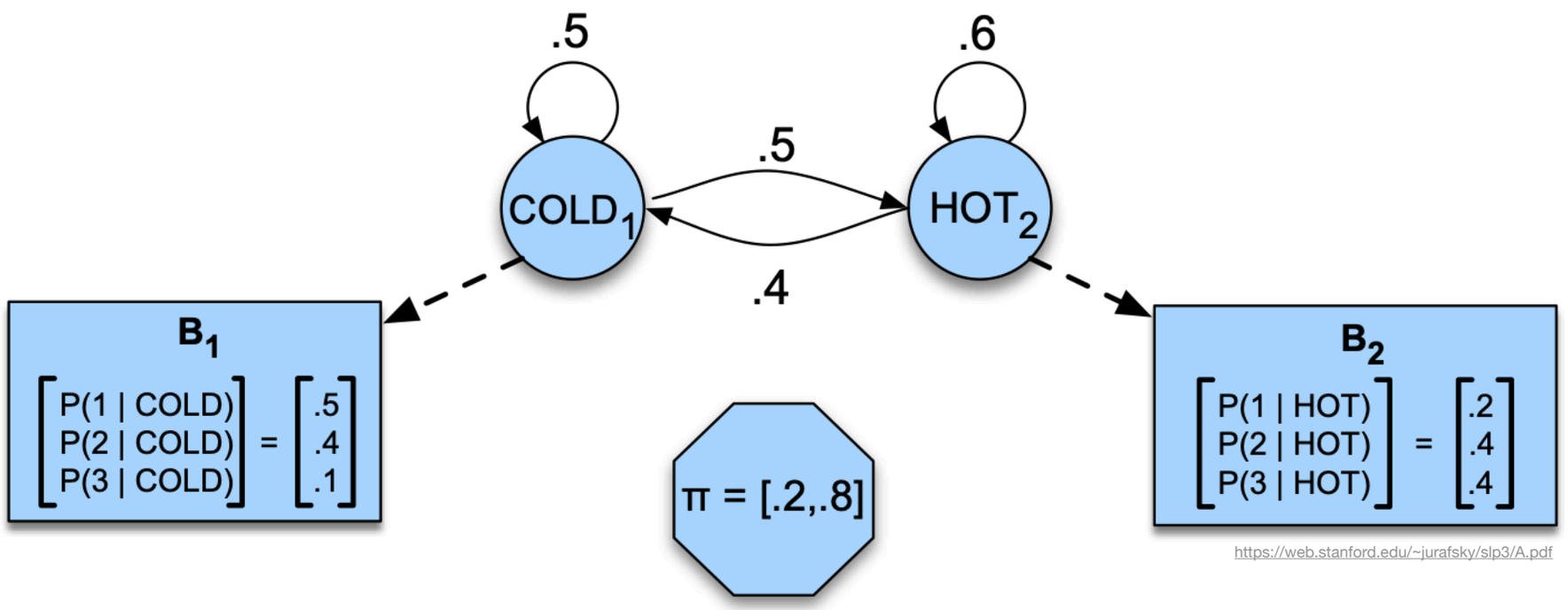
https://web.stanford.edu/~jurafsky/slp3/A.pdf

Definition

The pair (X_n, Y_n) is an HMM if:

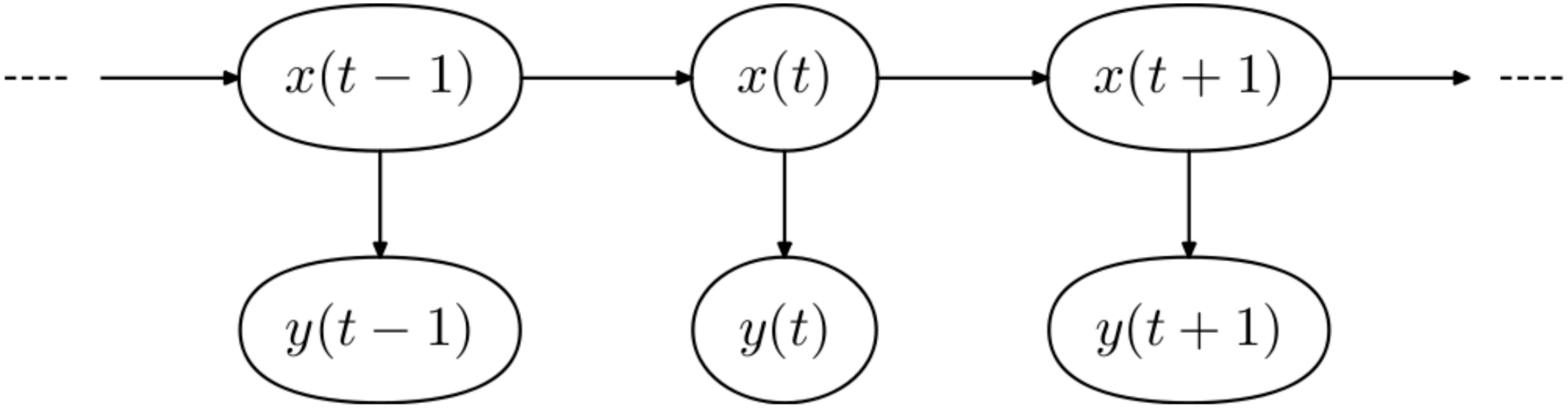


1. X_n is not directly observable;
2. $P(Y_n \in A \,|\, X_1 = x_1, \ldots, X_n = x_n) = P(Y_n \in A \,|\, X_n = x_n)$



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

HMM



https://en.wikipedia.org/wiki/Hidden_Markov_model#Inference

Example HMM

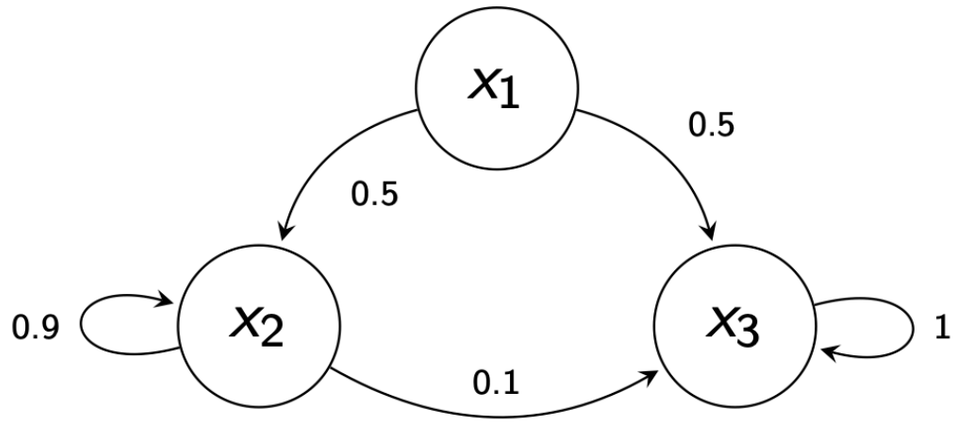
Three states:
$\mathcal{X} = \{x_1, x_2, x_3\}$.

Three possible observations:
$\mathcal{Z} = \{2, 3\}$.

Initial distribution:
$\pi = (1, 0, 0)$.

Transition probabilities:

$$T = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0 & 0.9 & 0.1 \\ 0 & 0 & 1 \end{bmatrix}$$



Observation probabilities:

$$M = \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$$

Observation sequence:

$Z = (2, 3, 3, 2, 2, 2, 3, 2, 3)$.

https://ocw.mit.edu/courses/16-410-principles-of-autonomy-and-decision-making-fall-2010/2ebbc8cc4bc9adc3418a572a17331f63_MIT16_410F10_lec21.pdf

Inference

Assume we know the *emission*, *transition* and *initial* probabilities and are given an observed sequence.

Forward algorithm: *probability* of observed sequence **up to** a certain time point.

Backward algorithm: *probability* of observed sequence **from** a certain time point.

Forward-backward algorithm: *posterior marginals* of **all** hidden state variables.

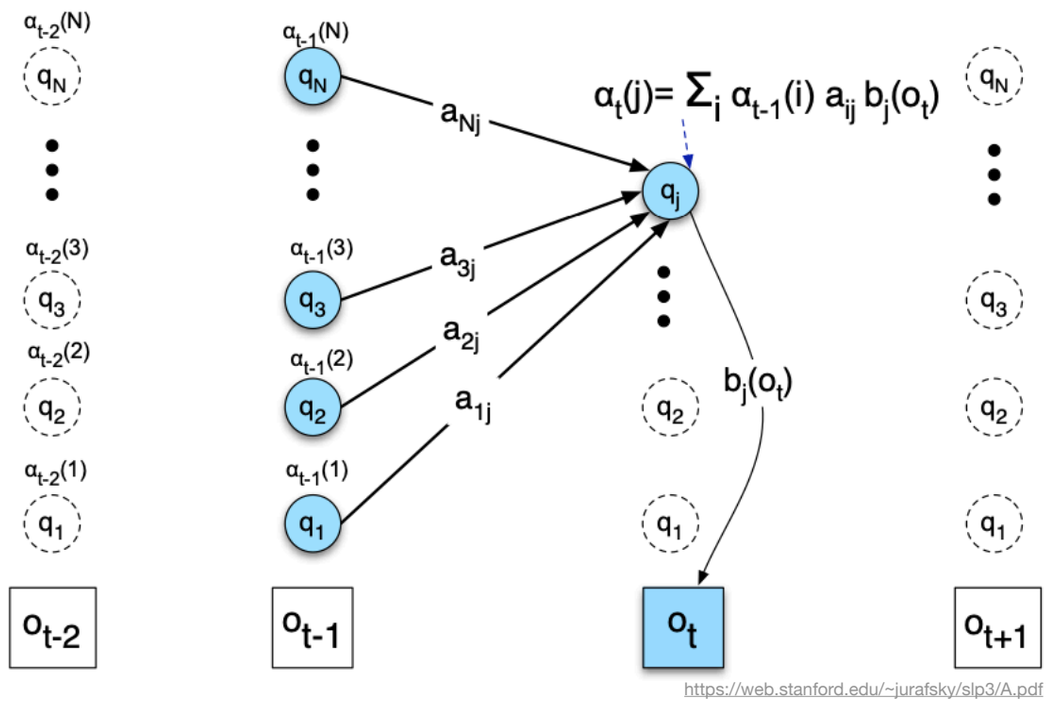
Posterior decoding: obtain *most likely* hidden states at **each** individual position

Viterbi algorithm: obtain *most likely* hidden state **sequence** (Viterbi path)

Forward algorithm

$$\text{Let } \alpha_t(x_t) = p(x_t, y_{1:t}), \quad \alpha_0(x_0) = \pi$$

$$\alpha_t(x_t) = p(x_t, y_{1:t}) = \sum_{x_{t-1}} p(x_t, x_{t-1}, y_{1:t})$$



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

Forward algorithm

$$\text{Let } \alpha_t(x_t) = p(x_t, y_{1:t}), \quad \alpha_0(x_0) = \pi$$

$$\alpha_t(x_t) = p(x_t, y_{1:t}) = \sum_{x_{t-1}} p(x_t, x_{t-1}, y_{1:t})$$

$$\alpha_t(x_t) = \sum_{x_{t-1}} p(y_t|x_t, x_{t-1}, y_{1:t-1}) p(x_t, x_{t-1}, y_{1:t-1})$$

Forward algorithm

$$\text{Let } \alpha_t(x_t) = p(x_t, y_{1:t}), \quad \alpha_0(x_0) = \pi$$

$$\alpha_t(x_t) = p(x_t, y_{1:t}) = \sum_{x_{t-1}} p(x_t, x_{t-1}, y_{1:t})$$

$$\alpha_t(x_t) = \sum_{x_{t-1}} p(y_t|x_t, x_{t-1}, y_{1:t-1}) p(x_t, x_{t-1}, y_{1:t-1})$$

$$\alpha_t(x_t) = p(y_t|x_t) \sum_{x_{t-1}} p(x_t|x_{t-1}, y_{1:t-1}) p(x_{t-1}, y_{1:t-1})$$

Forward algorithm

$$\text{Let } \alpha_t(x_t) = p(x_t, y_{1:t}), \quad \alpha_0(x_0) = \pi$$

$$\alpha_t(x_t) = p(x_t, y_{1:t}) = \sum_{x_{t-1}} p(x_t, x_{t-1}, y_{1:t})$$

$$\alpha_t(x_t) = \sum_{x_{t-1}} p(y_t|x_t, x_{t-1}, y_{1:t-1}) p(x_t, x_{t-1}, y_{1:t-1})$$

$$\alpha_t(x_t) = p(y_t|x_t) \sum_{x_{t-1}} p(x_t|x_{t-1}, y_{1:t-1}) p(x_{t-1}, y_{1:t-1})$$

$$\alpha_t(x_t) = p(y_t|x_t) \sum_{x_{t-1}} p(x_t|x_{t-1}) \alpha_{t-1}(x_{t-1})$$

Forward algorithm

$$\alpha_t(x_t) = \underbrace{p(y_t|x_t)}_{\text{emission probability}} \sum_{x_{t-1}} \underbrace{p(x_t|x_{t-1})}_{\text{transition probability}} \alpha_{t-1}(x_{t-1})$$

Forward algorithm

$$\alpha_t(x_t) = \underbrace{p(y_t|x_t)}_{\text{emission probability}} \sum_{x_{t-1}} \underbrace{p(x_t|x_{t-1})}_{\text{transition probability}} \alpha_{t-1}(x_{t-1})$$

$$\alpha_t = p(y_{1:t}) = \sum_{x_t} p(x_t, y_{1:t}) = \sum_{x_t} \alpha_t(x_t)$$

$$p(x_t|y_{1:t}) = \frac{p(x_t, y_{1:t})}{p(y_{1:t})} = \frac{\alpha_t(x_t)}{\alpha_t}$$

Forward algorithm

```python
def forward(observations, states, start_prob, trans_prob, emm_prob) -> (float, np.ndarray[float]):
    """
    Forward algorithm.
    :return: probability of the observation sequence and the forward probabilities matrix
    """
    # convert state names to indices for easier access
    state_indices = {state: i for i, state in enumerate(states)}

    # initialize the forward probabilities matrix
    fwd = np.zeros((len(observations), len(states)))

    # initialize the first time step with start probabilities
    for state in states:
        fwd[0][state_indices[state]] = start_prob[state] * emm_prob[state][observations[0]]

    # iterate over each observation (after the first)
    for i in range(1, len(observations)):
        for current in states:
            sum_prob = 0
            for previous in states:
                prob = fwd[i - 1][state_indices[previous]] * trans_prob[previous][current]
                sum_prob += prob
            fwd[i][state_indices[current]] = sum_prob * emm_prob[current][observations[i]]

    # probability of the observation sequence
    return np.sum(fwd[-1]), fwd
```

Backward algorithm

$$\text{Let } \beta_t(x_t) = p(y_{t+1:T}|x_t), \quad \beta_T(x_T) = 1$$

$$\beta_t(x_t) = p(y_{t+1:T}|x_t) = \sum_{x_{t+1}} p(y_{t+1:T}, x_{t+1}|x_t)$$

$$\beta_t(x_t) = \sum_{x_{t+1}} p(y_{t+1}|x_{t+1}, y_{t+2:T}, x_t) p(y_{t+2:T}, x_{t+1}|x_t)$$

$$\beta_t(x_t) = \sum_{x_{t+1}} p(y_{t+1}|x_{t+1}) p(y_{t+2:T}|x_{t+1}, x_t) p(x_{t+1}|x_t)$$

$$\beta_t(x_t) = \sum_{x_{t+1}} \underbrace{p(y_{t+1}|x_{t+1})}_{\text{emission probability}} \underbrace{p(x_{t+1}|x_t)}_{\text{transition probability}} \beta_{t+1}(x_{t+1})$$

Backward algorithm

```python
def backward(observations, states, start_prob, trans_prob, emm_prob) -> (float, np.ndarray[float]):
    """
    Backward algorithm.
    :return: probability of the observation sequence and the backward probabilities matrix
    """
    # convert state names to indices for easier access
    state_indices = {state: i for i, state in enumerate(states)}

    # initialize the backward probabilities matrix
    bwd = np.zeros((len(observations), len(states)))

    # initialize the last time step with probabilities of 1
    bwd[-1][:] = 1

    # iterate over each observation backwards (excluding the last)
    for i in range(len(observations) - 2, -1, -1):
        for curr_state in states:
            sum_prob = 0
            for next_state in states:
                prob = bwd[i + 1][state_indices[next_state]] * trans_prob[curr_state][next_state] * \
                       emm_prob[next_state][observations[i + 1]]
                sum_prob += prob
            bwd[i][state_indices[curr_state]] = sum_prob

    # initial probability of the observation sequence
    p = sum(start_prob[state] * bwd[0][state_indices[state]] * emm_prob[state][observations[0]]
            for state in states)

    return p, bwd
```

Forward-backward algorithm

$$\alpha_n(x_n) = p(y_{1:n}, x_n)$$
$$\beta_n(x_n) = p(y_{n+1:T} \mid x_n)$$

$$P(x_n|y_{1:T}) = \frac{P(x_n, y_{1:T})}{P(y_{1:T})} =$$
$$= \frac{\alpha_n(x_n)\beta_n(x_n)}{\sum_{i=1}^{T} \alpha_i(x_i)\beta_i(x_i)}$$

Forward-backward algorithm

```python
def forward_backward(observations, states, start_prob, trans_prob, emm_prob) -> List[Dict[str, float]]:
    """
    Forward-backward algorithm.
    :return: probability of the observation sequence
    """
    p, fwd = forward(observations, states, start_prob, trans_prob, emm_prob)
    _, bwd = backward(observations, states, start_prob, trans_prob, emm_prob)

    # Merging the two parts
    posterior = []
    for i in range(len(observations)):
        posterior.append({state: fwd[i][j] * bwd[i][j] / p for j, state in enumerate(states)})

    return posterior
```

Posterior decoding

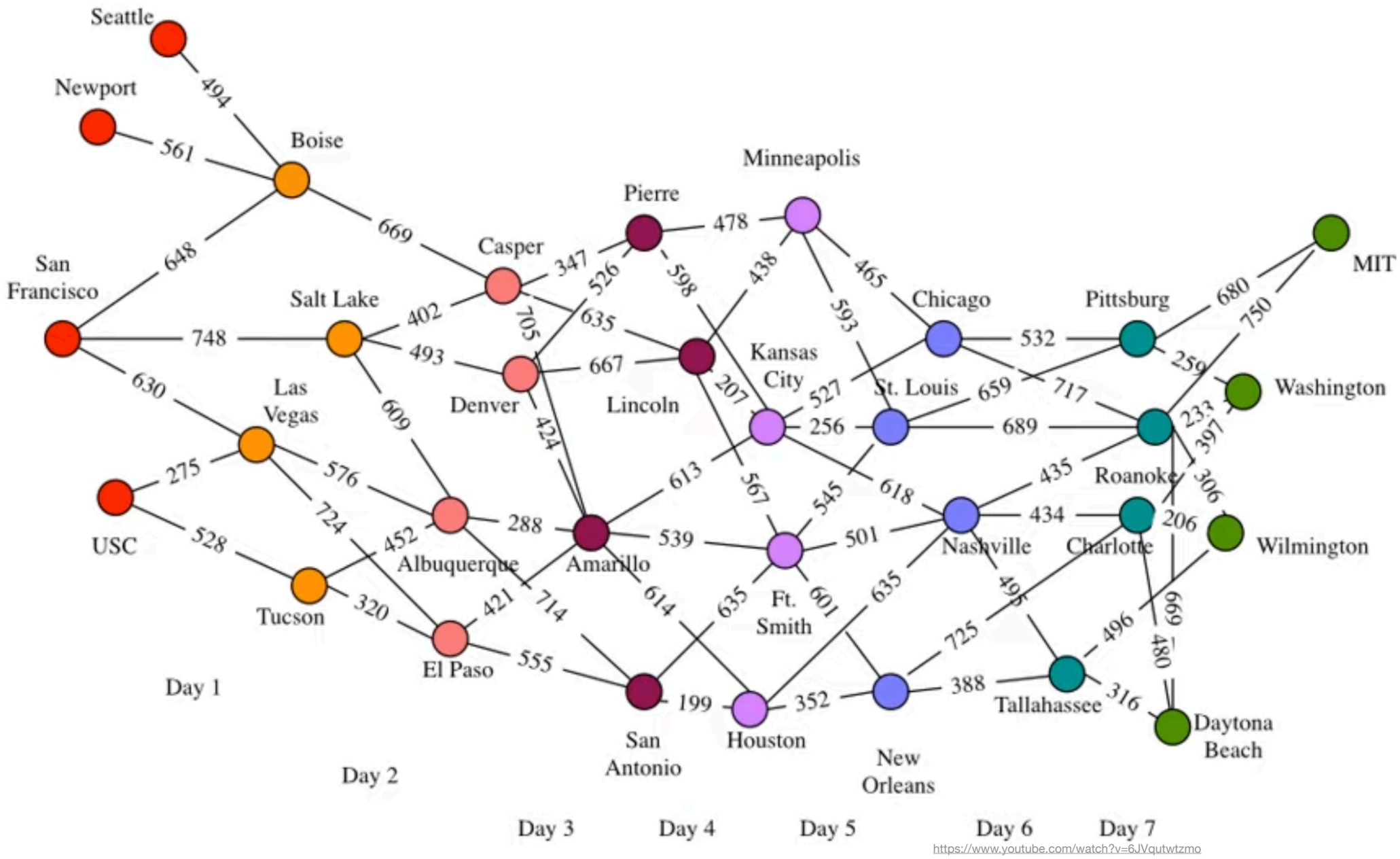
$$x_n^* = \arg\max_{x_n} P(x_n \mid y_{1:T})$$

$$= \arg\max_{x_n} \frac{\alpha_n(x_n)\beta_n(x_n)}{\sum_{i=1}^{T} \alpha_i(x_i)\beta_i(x_i)}$$

Posterior decoding

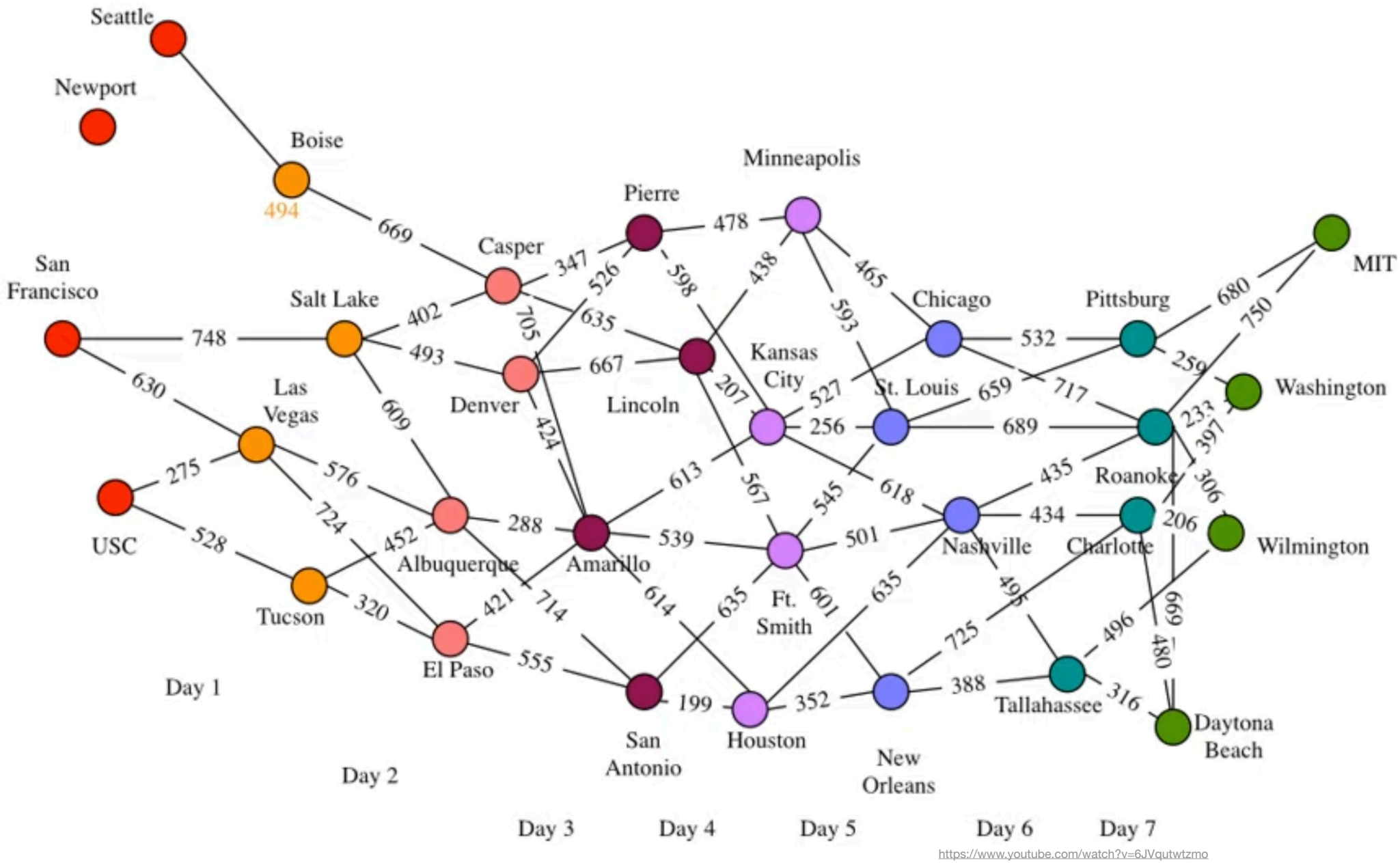
```python
def posterior_decoding(observations, states, start_prob, trans_prob, emm_prob) -> List[str]:
    """
    Posterior decoding.
    :return: the most probable state sequence
    """
    posterior = forward_backward(observations, states, start_prob, trans_prob, emm_prob)

    return [max(x, key=x.get) for x in posterior]
```

Viterbi algorithm



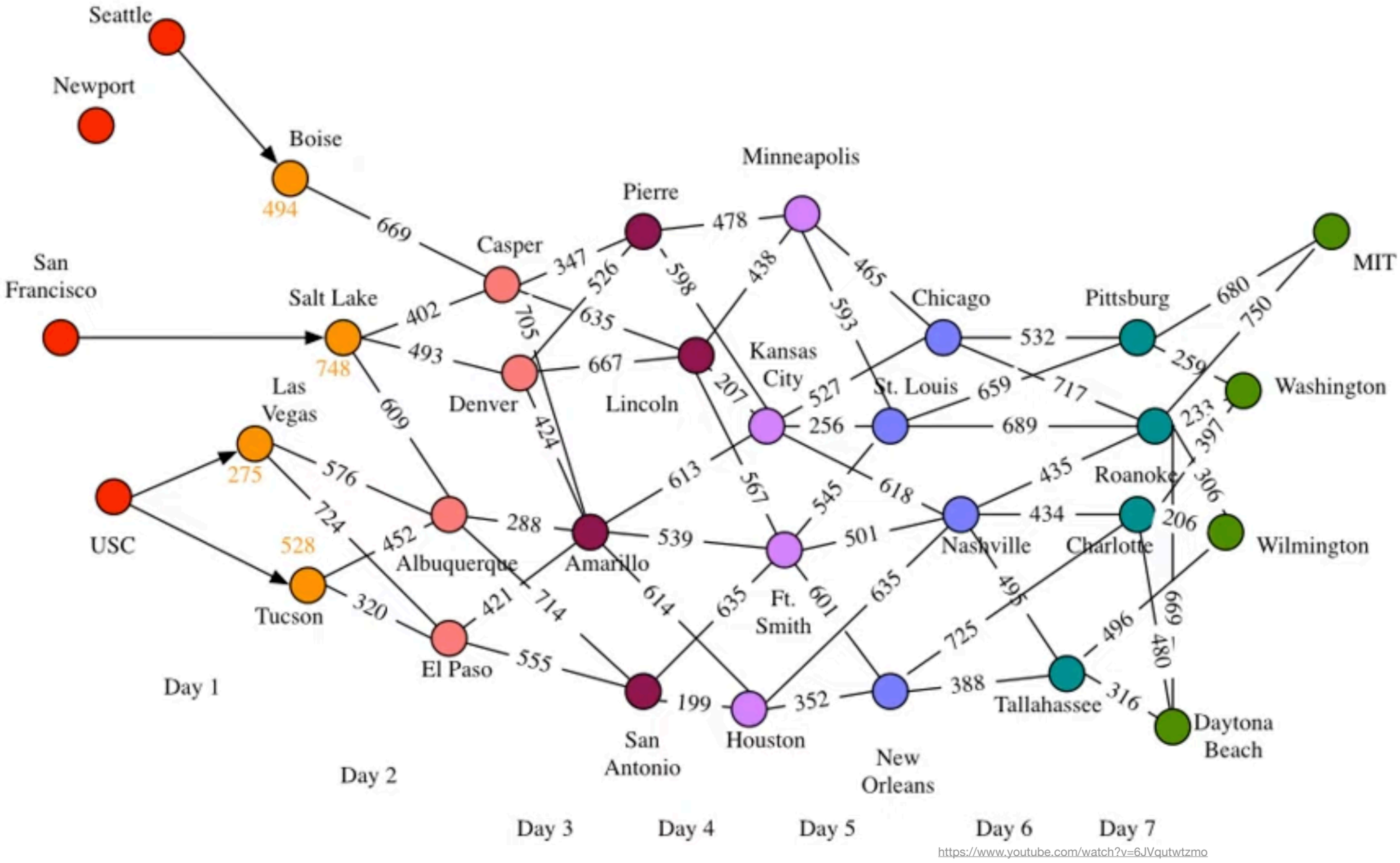
https://www.youtube.com/watch?v=6JVqutwtzmo

Viterbi algorithm



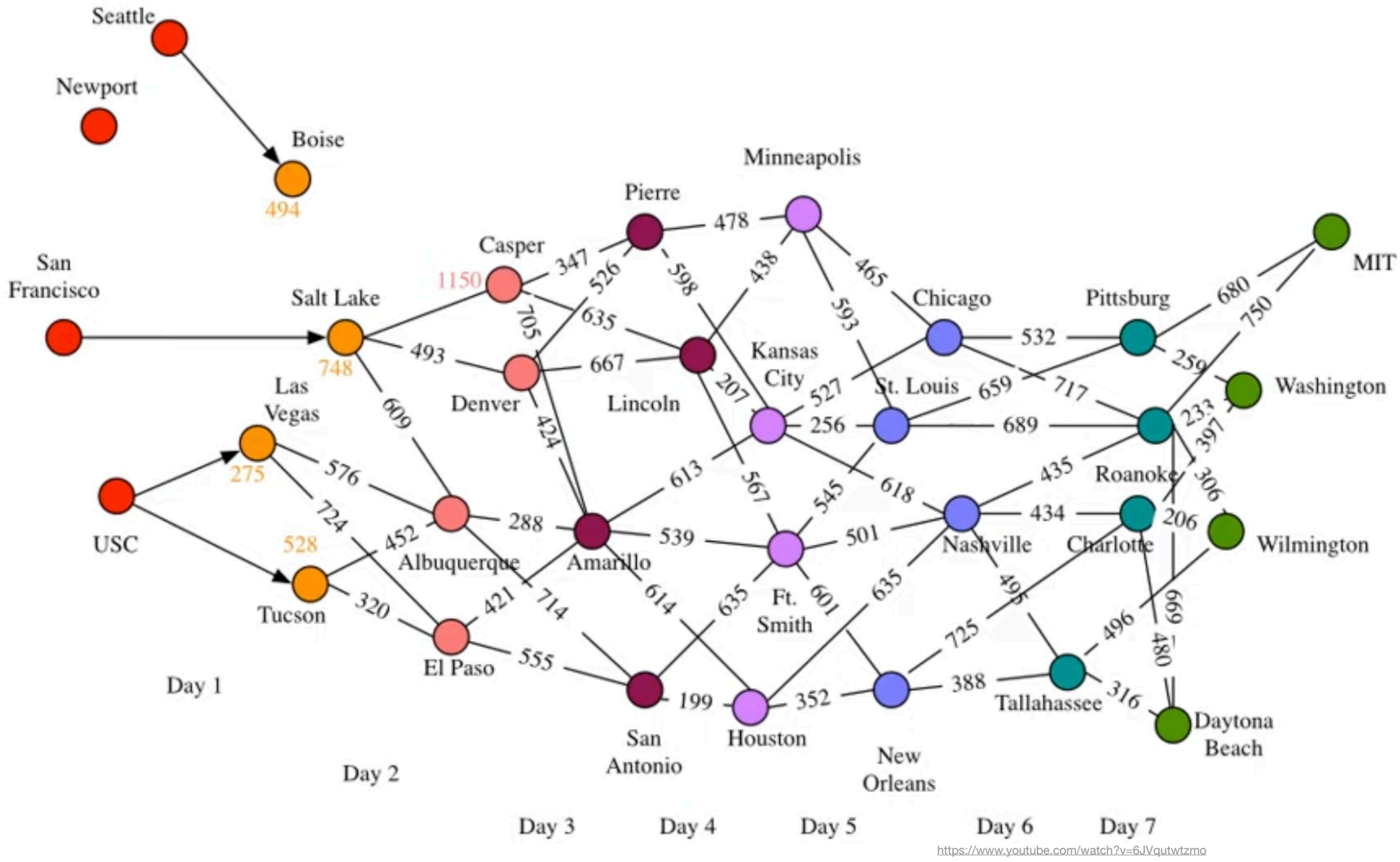
https://www.youtube.com/watch?v=6JVqutwtzmo

Viterbi algorithm



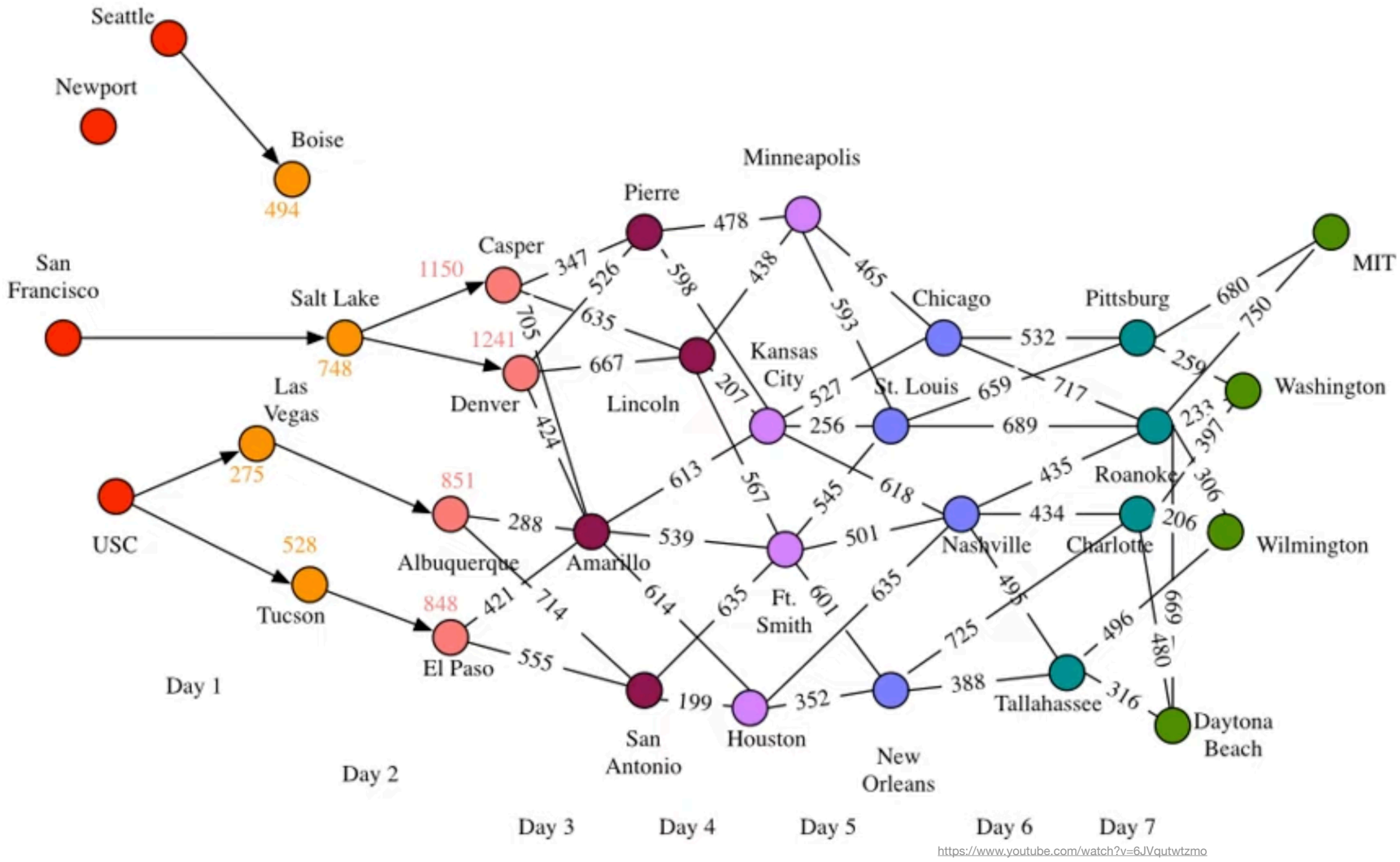
https://www.youtube.com/watch?v=6JVqutwtzmo

Viterbi algorithm



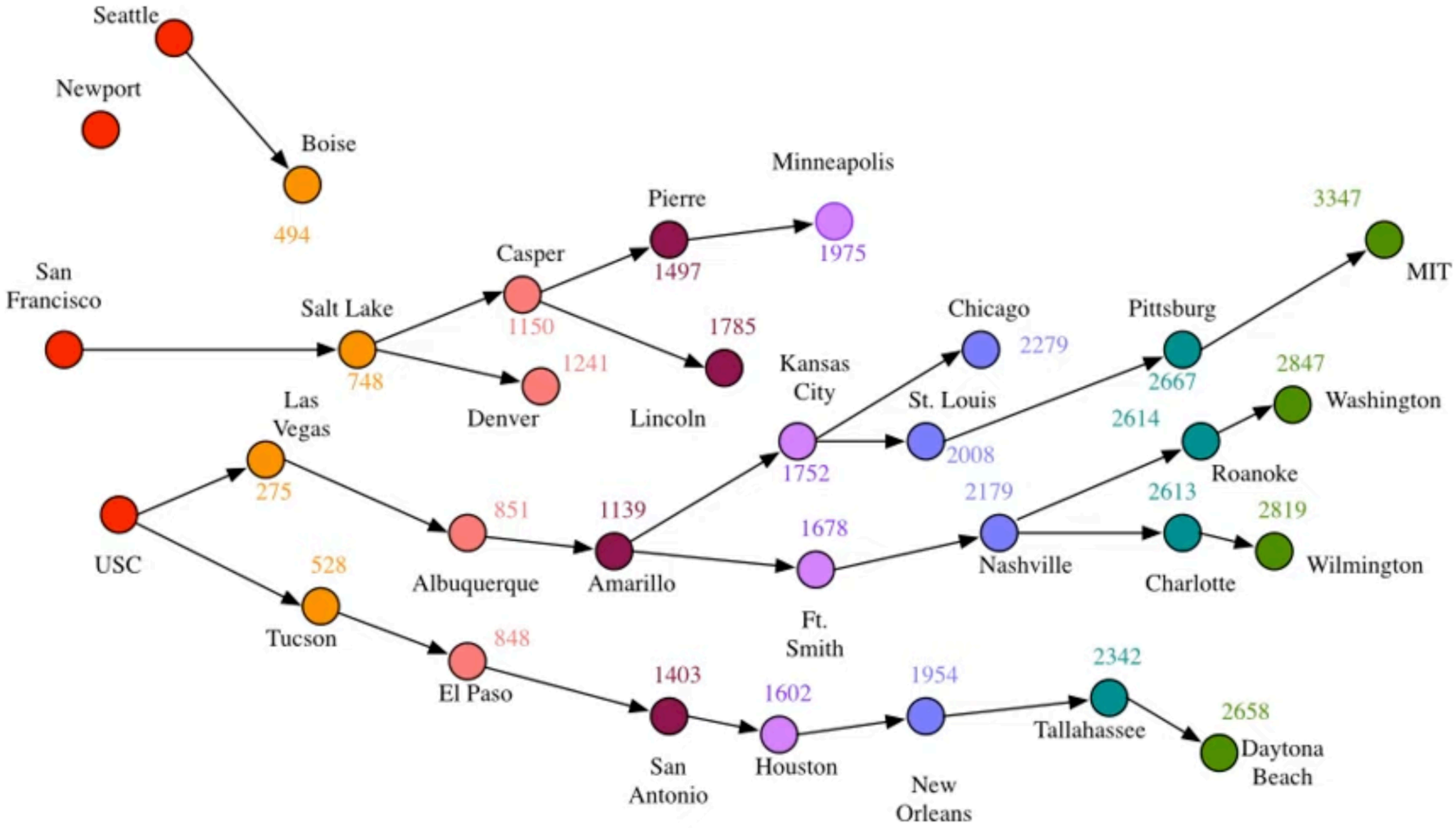
https://www.youtube.com/watch?v=6JVqutwtzmo

Viterbi algorithm



https://www.youtube.com/watch?v=6JVqutwtzmo

Viterbi algorithm



https://www.youtube.com/watch?v=6JVqutwtzmo

Viterbi algorithm

```python
def viterbi(observations, states, start_prob, trans_prob, emm_prob) -> (float, List[str]):
    """
    Viterbi algorithm.
    :return: The probability of the most probable state sequence and the sequence itself
    """
    state_indices = {state: i for i, state in enumerate(states)}

    # initialize the Viterbi probability matrix and path pointers
    viterbi_prob = np.zeros((len(observations), len(states)))
    path_pointers = np.zeros((len(observations), len(states)), dtype=int)

    # initialize the first time step
    for state in states:
        viterbi_prob[0][state_indices[state]] = start_prob[state] * emm_prob[state][observations[0]]

    # iterate over each observation (after the first)
    for i in range(1, len(observations)):
        for curr_state in states:
            max_prob, max_state = max(
                (viterbi_prob[i - 1][state_indices[prev_state]] * trans_prob[prev_state][curr_state], prev_state)
                for prev_state in states
            )
            viterbi_prob[i][state_indices[curr_state]] = max_prob * emm_prob[curr_state][observations[i]]
            path_pointers[i][state_indices[curr_state]] = state_indices[max_state]

    # backtrack to find the most probable path
    last_state = np.argmax(viterbi_prob[-1])
    best_path = [states[last_state]]

    for i in range(len(observations) - 1, 0, -1):
        last_state = path_pointers[i][last_state]
        best_path.insert(0, states[last_state])

    # probability of the most probable path
    max_prob = np.max(viterbi_prob[-1])

    return max_prob, best_path
```

Inference

Assume we know the *emission*, *transition* and *initial* probabilities and are given an observed sequence.

Forward algorithm: *probability* of observed sequence **up to** a certain time point.

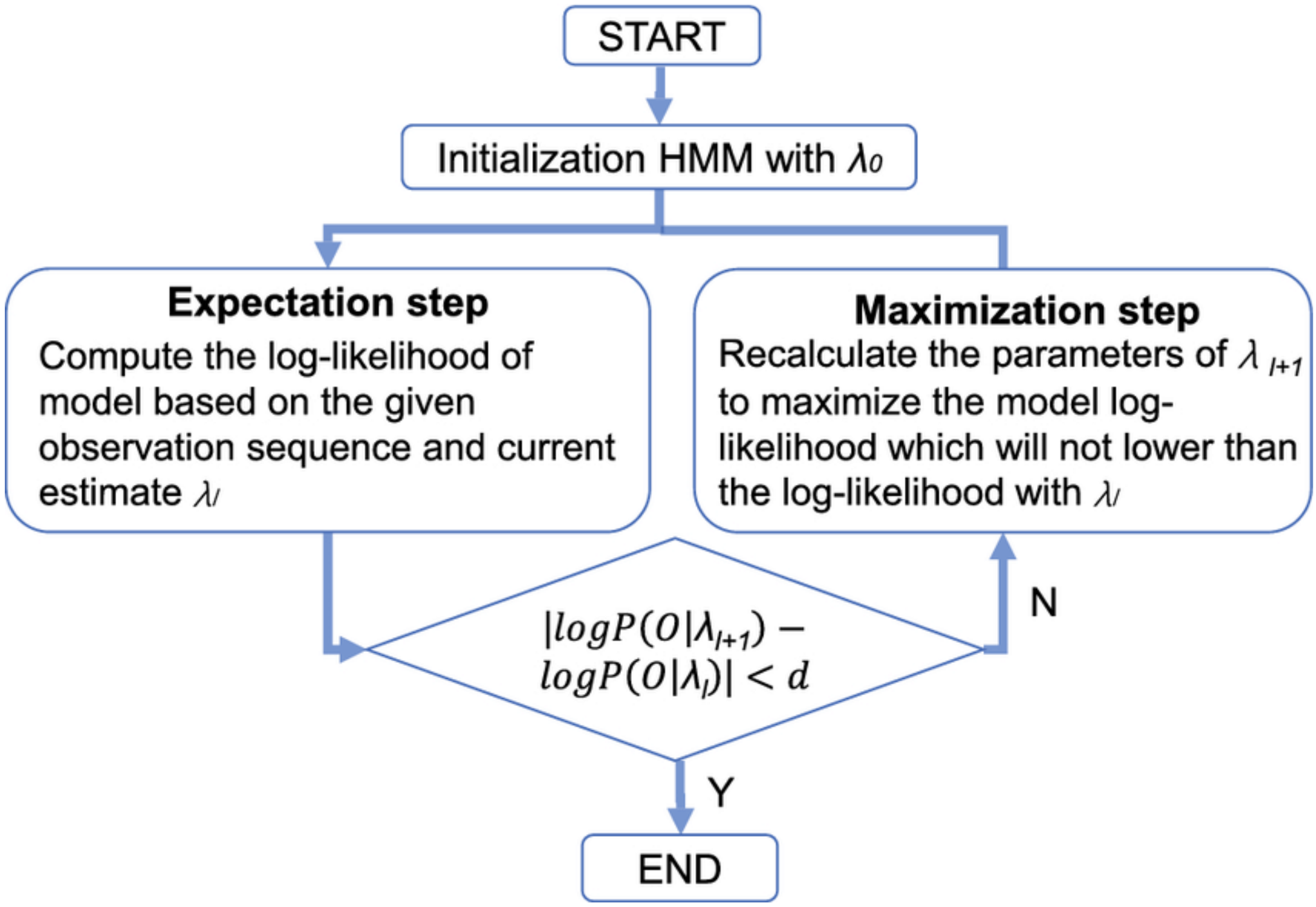
Backward algorithm: *probability* of observed sequence **from** a certain time point.

Forward-backward algorithm: *posterior marginals* of **all** hidden state variables.

Posterior decoding: obtain *most likely* hidden states at **each** individual position

Viterbi algorithm: obtain *most likely* hidden state **sequence** (Viterbi path)

Baum-Welch algorithm



https://www.researchgate.net/figure/Baum-Welch-algorithm-EM-algorithm_fig1_350610085

Exercise

SCAN ME