Problems

1. The Shortest Path Home (Without Wandering Too Much).

Dijkstra’s algorithm can find a shortest $s$-$t$ path this in time $O(m \log n)$—and even $O(m + n \log n)$, if we use data-structures called Fibonacci heaps. Still, it explores a lot of the graph that may not relevant to the shortest $s$-$t$ path. Here’s an idea that does better on many real-world instances.

Let’s fix some notation: recall that Dijkstra’s algorithm maintains an (over)estimate $d(v)$ of the shortest $s$-$v$ path length for each vertex $v$. (This is initially $d(s) = 0$ and $d(v) = \infty$ for all $v \neq s$). It then repeatedly picks an unmarked vertex $u$ with the lowest $d(u)$ value, marks it, and “relaxes” all its outgoing edges (i.e., sets $d(v) \leftarrow \min\{d(v), d(u) + \ell_{uv}\}$ for all other unmarked vertices $v$).

(a) Suppose we have target-estimates $h_t(v) \geq 0$, which are estimates of the shortest-path distances from each vertex $v$ to the target $t$. Define the targeted-Dijkstra algorithm where the algorithm repeatedly picks an unmarked vertex with lowest $d(v) + h_t(v)$.

i. (Do not submit.) Observe that if $h_t(v)$ equals the shortest-path distance from $v$ to $t$, then the algorithm will explore a shortest-path from $s$ to $t$. Also that $h_t(v) = 0$ gives back Dijkstra’s algorithm.

ii. Construct examples which show that poor settings of target-estimates may result in this algorithm not returning the shortest path distances.

(b) Call a target-estimate function $h : V \to \mathbb{R}$ sensible if $h(v) \leq \ell_{vw} + h(w)$ for all edges $(v, w) \in E$. Prove that the targeted-Dijkstra algorithm always returns the shortest-path distance. Hint: inductively show that the values $d(v)$ assigned to marked vertices are the correct shortest-path distances from $s$.

(c) Clearly we cannot maintain target-estimates $h_t(v)$ for each vertex $v$ and each target $t$! But we can do the following:

i. Fix a dozen or so landmarks $L \subseteq V$ in the graph.

ii. For each vertex $u$, precompute and store its distance $D(u, a)$ to each landmark $a \in L$. Now when we need a target-estimate $h_t(v)$ for the distance from $v$ to $t$, return $h_t(v) := \max_{a \in L}(D(v, a) - D(t, a))$. Show that these target-estimates are sensible.

(Hint: show that for any $a$, $D(v, a) - D(t, a)$ is a sensible estimate. Then show that the maximum of sensible estimates is sensible.) Feel free to assume part $(1b)$ even if you haven’t solved it.

Hence by storing a dozen or so pieces of data for each vertex (i.e., the distance of each location to all the landmarks), we can reduce the search space and still correctly compute the shortest $s$-$t$ path. Here are examples showing how this reduces the amount of the graph searched; the red diamonds are the landmarks.
2. TLS/SSL Certificate Revocations.

A common problem that web browsers face is checking if a certificate has been revoked. Let $U$ be the set of all TLS Certificates and let $R \subset U$ be the set of certificates that have been revoked. The set of revoked certificates changes over time, so your goal is to periodically broadcast the current set to all web browsers. Ideally, you would like to represent $R$ using a data structure $D$ that is both compact and fast at answering membership queries. Compactness is important for minimizing network traffic and speed is important for efficiency on the browser side.

Bloom filters are a natural choice for storing $R$, but we need a way to handle false positives (i.e., certificates that are identified as revoked even though they are not). Assuming we have access to both $R$ and $U$, one idea is to store all the false positives from the first Bloom filter $B_1$ in a second Bloom filter $B_2$, and then storing all the false positives from $B_2$ in a sorted linked list $L$. More precisely, $B_2$ stores the set of certificates in $U \setminus R$ for which $B_1$ returns true and $L$ stores the set of certificates in $R$ for which $B_2$ returns true.

(a) For any $u \in U$, describe an algorithm for checking if $u \in R$, with no false positives or false negatives, using the multi-level Bloom filter described above.

(b) Let $r = |R|$, $n = |U|$ and $s = n - r$. For the remaining parts of this problem, suppose the Bloom filter $B_1$ uses $k = \log_2 (s/r)$ hash functions and an array of $m = \frac{r k}{\ln 2}$ bits (you can assume that $k$ and $m$ work out to be integers). What’s the probability of a false positive in $B_1$? In other words, given any $u \in U \setminus R$, what’s the probability (over the random choice of hash functions) that $B_1$ will return true on $u$? State the probability in terms of $r$, $n$ and $s$. (You can assume the hash functions are perfectly random and you can use the approximation in equation (5.2) of notes for Lecture 3 (Page 33).)

(c) Let $r'$ be the number of certificates stored in $B_2$, what’s the expected value of $r'$?

(d) The number of revoked certificates is usually much smaller than the total number of certificates. Suppose $s = \Theta(r^2)$. What’s the expected size of $L$? Also, what’s the expected running time of the query algorithm described in part (a)? Give asymptotic bounds for both questions in terms of $r$, $n$ and $s$, assuming $B_2$ uses $k' = k = \log_2 (s/r)$ hash functions with an array of $m' = \frac{r k}{\ln 2}$ bits.

Hint: Jensen’s inequality might be helpful for showing the expected time bounds.

3. A Very Likely Story.

A classic method to encrypt information is to use a one-time pad, which takes a message $m \in \Sigma^T$ and a key $k \in \Sigma^T$ and produces a ciphertext $c := m \oplus k$; here $\Sigma$ is some alphabet. As
the name implies, the key should be used only once. If the key is used again to get \( c' = m' \oplus k \),
an attacker could take the XOR of \( c \) and \( c' \) to obtain \( d = m \oplus m' \). In this problem, we will
derive a dynamic programming algorithm that could be used to recover the most likely values of \( m \) and \( m' \)
from \( d \), assuming the messages \( m, m' \) correspond to English sentences.

A *Hidden Markov Model* (HMM) has a set of variables \( x_{1:T} = (x_1, \ldots, x_T) \) where \( x_t \) is a
“hidden” state at time \( t \) (and takes on some value in \( \{1, \ldots, N\} \)), and another a set of
variables \( y_{1:T} = (y_1, \ldots, y_T) \) where \( y_t \) is the observation made at time \( t \) (and takes on values
in \( \{1, \ldots, M\} \)). An HMM is characterized by two assumptions:

- \( y_t \) depends only on \( x_t \): i.e. \( \Pr(y_t \mid y_{1:t-1}, y_{t+1:T}, x_{1:T}) = \Pr(y_t \mid x_t) \).
- \( x_t \) depends only on \( x_{t-1} \): i.e. \( \Pr(x_t \mid x_{1:t-1}) = \Pr(x_t \mid x_{t-1}) \).

Hence a HMM is often depicted like this:

As input, you are given:

- \( \pi = (\pi_1, \ldots, \pi_N) \): initial state probability distribution such that \( \pi_i = \Pr(x_1 = i) \).
- \( A \in [0, 1]^{N \times N} \): state-transition matrix such that \( a_{i,j} = \Pr(x_t = j \mid x_{t-1} = i) \).
- \( B \in [0, 1]^{N \times M} \): observation matrix such that \( b_{i,j} = \Pr(y_t = j \mid x_t = i) \).

In our example, \( x_t \) would correspond to the pair of characters \( (m_t, m'_t) \) (which we want to find out)
and \( y_t \) would correspond to \( m_t \oplus m'_t \) (which we know). The parameters \( \pi \) and \( A \) could be
obtained by studying the relative frequencies of sequences of two characters in English text, and \( B \) would be such that \( b_{(m_t, m'_t), y_t} = 1 \) if \( m_t \oplus m'_t = y_t \) and 0 otherwise.

Given a sequence of \( T \) observations \( y_{1:T} \), we want to find the most likely sequence of states \( x_{1:T} \).
That is, we want to find:

\[
\arg\max_{x_{1:T}} \Pr(x_{1:T} \mid y_{1:T}) = \arg\max_{x_{1:T}} \frac{\Pr(x_{1:T}, y_{1:T})}{\Pr(y_{1:T})} = \arg\max_{x_{1:T}} \Pr(x_{1:T} \mid y_{1:T}).
\]

where the first equality uses the definition of conditional probability, and the second equality
uses that \( \Pr(y_{1:T}) \) is constant with respect to \( x_{1:T} \).

a) Express \( \Pr(x_{1:t}, y_{1:t}) \) in terms of the elements of \( A, B, \pi \) and \( \Pr(x_{1:t-1}, y_{1:t-1}) \) for \( t = 1, \ldots, T \).
Don’t forget the base case \( t = 1 \).

b) Let \( V_t(i) \) be the maximum possible probability when \( x_t = i \), i.e. \( V_t(i) := \max_{x_{1:t-1}} \Pr(x_t = i, x_{1:t-1}, y_{1:t}) \) for \( t \geq 2 \) and \( V_1(i) := \Pr(x_1 = i, y_1) \).
Give a formula for calculating the value of \( V_t(i) \) for a given \( i \in \{1, \ldots, N\} \) using \( \{V_{t-1}(j)\}_{j=1}^N, A, B, \) and \( \pi \).

c) Give an algorithm (just the pseudocode) that uses dynamic programming to find and
output the most likely sequence of states \( x_{1:T} \) for given \( y_{1:T} \). The running time of your
algorithm should be \( O(TN^2) \).