1. **MST mechanics.** Consider the following weighted, undirected graph:

(a) Assume we run Prim’s MST algorithm starting at vertex A. List the edges that get added to the tree in the order in which the algorithm adds them.

(b) Now do the same thing for Kuskel’s algorithm.

2. **Road trip.** You are going on a long trip. You start on the road at mile post \( a_0 \). Along the way there are \( n \) hotels, at mile posts \( a_1, \ldots, a_n \), where \( a_0 < a_1 < a_2 < \cdots < a_n \). The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at mile post \( a_n \)), which is your destination. You’d ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel \( x \) miles during a day, the penalty for that day is \( (200 - x)^2 \). You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties. Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

To do this, you are to use Dynamic Programming.

(a) To begin with, design a recursive algorithm that computes the minimum penalty by filling in the details of the following algorithm outline.

\[
\text{Penalty}(\langle a_0, a_1, \ldots, a_n \rangle) : \\
\text{if } n = 0 \\
\quad \text{result} \leftarrow 0 \\
\text{else} \\
\quad \text{result} \leftarrow \infty \\
\quad \text{for } i \text{ in } [0..n] \text{ do} \\
\quad \quad \text{penaltyForLastDay} \leftarrow \\
\quad \quad \text{penaltyForPreviousDays} \leftarrow \\
\quad \quad \text{result} \leftarrow \min(\text{result}, \text{penaltyForLastDay} + \text{penaltyForPreviousDays}) \\
\text{return result}
\]

(b) Next, identify the subproblems and describe the subproblem graph. In particular, estimate the total number of subproblems, as a function of \( n \).

(c) Next, show how to modify your algorithm from part (a) using “memoization” to get an efficient algorithm. Estimate the running time of your algorithm.

(d) Next, show how to turn the algorithm from part (c) into an iterative algorithm, and estimate the running time of your algorithm.

(e) Finally, show how to modify your algorithm from part (d) to actually compute an itinerary that yields the minimum penalty, and estimate the running time of your algorithm.
3. **Corrupted text.** You are given a string of \( n \) characters \( s[1 \ldots n] \), which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like “itwasthebestoftimes…”). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function \( \text{dict}(\cdot) \): for any string \( w \), \( \text{dict}(w) = \text{true} \) if \( w \) is a valid word, and \( \text{dict}(w) = \text{false} \) otherwise.

   (a) Give an algorithm that determines whether the string \( s \) can be reconstituted as a sequence of valid words. The running time should be at most \( O(n^2) \), assuming calls to \( \text{dict} \) take unit time.

   (b) In the event that the string is valid, make your algorithm output the corresponding sequence of words.

   **Hint:** Use Dynamic Programming. Approach the problem as in Problem 2: first design a recursive algorithm, identify the subproblems, and then memoize (you don’t need to give an iterative algorithm).

4. **Coping with failure.** A mission-critical production system has \( n \) stages that have to be performed sequentially; stage \( i \) is performed by machine \( M_i \). Each machine \( M_i \) has a probability \( r_i \) of functioning reliably and a probability \( 1 - r_i \) of failing (and the failures are independent). Therefore, if we implement each stage with a single machine, the probability that the whole system works is \( r_1 r_2 \ldots r_n \). To improve this probability we add redundancy, by having \( m_i \) copies of the machine \( M_i \) that performs stage \( i \). The probability that all \( m_i \) copies fail simultaneously is only \( (1 - r_i)^m_i \), so the probability that stage \( i \) is completed correctly is \( 1 - (1 - r_i)^{m_i} \) and the probability that the whole system works is \( \prod_{i=1}^{n} (1 - (1 - r_i)^{m_i}) \). Each machine \( M_i \) has a cost \( c_i \), and there is a total budget \( B \) to buy machines. (Assume that \( B \) and \( c_i \) are positive integers.) Given the probabilities \( r_1, \ldots, r_n \), the costs \( c_1, \ldots, c_n \), and the budget \( B \), find the redundancies \( m_1, \ldots, m_n \) that are within the available budget and that maximize the probability that the system works correctly.

   **Hint:** Use Dynamic Programming. Approach the problem as in Problem 2: first design a recursive algorithm, identify the subproblems, and then memoize (you don’t need to give an iterative algorithm).

5. **Sequence alignment.** When a new gene is discovered, a standard approach to understanding its function is to look through a database of known genes and find close matches. The closeness of two genes is measured within the available budget and that maximize the probability that the system works correctly.

   **Hint:** Use Dynamic Programming. Approach the problem as in Problem 2: first design a recursive algorithm, identify the subproblems, and then memoize (you don’t need to give an iterative algorithm).

6. **Interval scheduling with weights.** Consider the interval scheduling problem discussed in class. In that problem, we wanted to scheduling as many non-overlapping jobs as possible, and we saw that a greedy algorithm based on the “earliest finish time first” strategy worked.

   Now suppose that each job \( j \) has a non-negative weight \( w_j \), and the goal is to schedule a set \( A \) of non-overlapping jobs such that the sum of weights \( \sum_{j \in A} w_j \) is maximized.

   (a) Show that the greedy algorithm based on “earliest finish time first” does not work in this setting. You should give a counter-example for which the greedy algorithm fails to find the best solution.

   (b) Show how to efficiently solve this problem using Dynamic Programming. As in Problem 2, first design a recursive algorithm, identify the subproblems, and then memoize.
7. **Road trip (II)**. Consider again Problem 2. Suppose at each mile post \(a_0, a_1, \ldots, a_n\) there is a gas station. You start out with \(u\) units of gas in your gas tank at mile post \(a_0\) and you want to get to mile post \(a_n\). Your car's gas tank has a capacity of \(c\) units of gas and your car gets \(R\) miles per unit of gas. You can stop at any mile post along the way and buy as much gas as you like. However, the following constraint must always be satisfied:

\[
\text{The amount of gas in your tank should never go below 0 or above } c. \quad (\ast)
\]

(a) Suppose the goal is to plan your trip so that you make as few stops as possible along the way to buy gas. Consider the following simple greedy strategy:

\[
\begin{align*}
\text{while you are not at the destination do} & \\
& \quad \text{fill up your tank at the current location} \\
& \quad \text{drive to the farthest gas station you can reach on a full tank of gas}
\end{align*}
\]

Prove that this strategy results in an itinerary that minimizes the number of stops. To do this, you should proceed as follows:

- Assume that greedy is not optimal. Specifically, assume there is a counter-example, that is, an input \(I = (u; a_0, \ldots, a_n; c, R)\) and a strategy \(S\) that beats the greedy strategy on input \(I\). Here, a strategy is any plan that indicates where and how much gas to buy, without violating the constraint \((\ast)\).
- Moreover, assume this counter-example is minimal, in the sense that there is no counter-example with a smaller number of mile posts.
- Argue that \(n > 0\) (this is trivial).
- Show how to construct a new input of the form \(I' = (u'; a_i, \ldots, a_n; c, R)\), where \(i > 0\), and a new strategy \(S'\) that beats the greedy strategy on input \(I'\). You should do this by showing how to transform \((I, S)\) into \((I', S')\).
- Conclude that this contradicts the minimality of the counter-example, and hence the greedy strategy must be optimal.

(b) Now assume that the gas stations sell gas at different prices, and that the goal is to plan the trip to minimize the total amount of money you spend on gas. To this end, assume that gas costs \(p_i > 0\) dollars per unit of gas at mile post \(a_i\), for \(i = 0, \ldots, n - 1\), and set \(p_n := 0\).

Consider the following greedy strategy:

\[
\begin{align*}
\text{while you are not the destination do} & \\
(1) & \quad \text{if you could reach a gas station with a cheaper price on a full tank of gas, then:} \\
& \quad \quad \text{buy the minimal amount of gas (possibly none) needed to the reach the first such} \\
& \quad \quad \text{station, and drive to it;} \\
(2) & \quad \text{otherwise, fill up your tank and drive to the next gas station}
\end{align*}
\]

Prove that this greedy strategy is optimal. To do this, you should proceed as follows:

- Assume that greedy is not optimal. Specifically, assume there is a counter-example, that is, an input \(I = (u; a_0, \ldots, a_n; p_0, \ldots, p_n; c, R)\) and a strategy \(S\) that beats the greedy strategy on input \(I\).
- Again, assume this counter-example is minimal, in the sense that there is no counter-example with a smaller number of mile posts.
- Argue that \(n > 0\) (again, this is trivial).
- Show how to construct a new input of the form \(I' = (u'; a_i, \ldots, a_n; p_i, \ldots, p_n; c, R)\), where \(i > 0\), and a new strategy \(S'\) that beats the greedy strategy on input \(I'\). You should do this by showing how to transform \((I, S)\) into \((I', S')\).
- Conclude that this contradicts the minimality of the counter-example, and hence the greedy strategy must be optimal.

(c) Suppose that the greedy algorithm in part (b) is modified so that case (1) reads as follows:

\[
\text{(1) if you could reach a gas station with a cheaper price on a full tank of gas, then:} \\
\quad \text{buy the minimal amount of gas (possibly none) needed to the reach the first such station, and drive to it;} \\
\]

Prove that this greedy strategy is optimal. To do this, you should proceed as follows:

- Assume that greedy is not optimal. Specifically, assume there is a counter-example, that is, an input \(I = (u; a_0, \ldots, a_n; p_0, \ldots, p_n; c, R)\) and a strategy \(S\) that beats the greedy strategy on input \(I\).
- Again, assume this counter-example is minimal, in the sense that there is no counter-example with a smaller number of mile posts.
- Argue that \(n > 0\) (again, this is trivial).
- Show how to construct a new input of the form \(I' = (u'; a_i, \ldots, a_n; p_i, \ldots, p_n; c, R)\), where \(i > 0\), and a new strategy \(S'\) that beats the greedy strategy on input \(I'\). You should do this by showing how to transform \((I, S)\) into \((I', S')\).
- Conclude that this contradicts the minimality of the counter-example, and hence the greedy strategy must be optimal.
(1) if you could reach a gas station with a cheaper price on a full tank of gas, then:
   buy the minimal amount of gas (possibly none) needed to reach the cheapest such station, and drive to it;

Show that this strategy is not optimal. You should do this by providing an explicit counter-example.