Problem 1. [30 points]
In class, we discussed a dynamic program to solve the weighted interval scheduling problem. Use the algorithm presented in class, which can also be found in page 258, to find optimal schedule for the following jobs:

- Job 1: s1 = 1, f1 = 5, v1 = 20
- Job 2: s1 = 4, f1 = 7, v1 = 10
- Job 3: s1 = 6, f1 = 9, v1 = 5
- Job 4: s1 = 9, f1 = 15, v1 = 15
- Job 5: s1 = 12, f1 = 16, v1 = 10

Show the vector \( M \) as generated by the algorithm. Then,

1. Describe the order in which you entered values into the vector.
2. What does each entry represent?
3. The output of this algorithm is the value of the best schedule of compatible jobs. What is the schedule itself?

Problem 2. [20 points]
Let \( G = (V, E) \) be an undirected graph with \( n \) nodes. A subset of the nodes is called an independent set if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we’ll see that it can be done efficiently if the graph is “simple” enough.

Call a graph \( G = (V, E) \) a path if its nodes can be written as \( v_1, v_2, \ldots, v_n \), with an edge between \( v_i \) and \( v_j \) if and only if the numbers \( i \) and \( j \) differ by exactly 1. With each node \( v_i \), we associate a positive integer weight \( w_i \).

Consider, for example, the five-node path drawn in Figure 6.28. The weights are the numbers drawn inside the nodes. The goal in this question is to solve the following problem: Find an independent set in a path \( G \) whose total weight is as large as possible.

(a) Give an example to show that the following algorithm does not always find an independent set of maximum total weight.

(b) Give an example to show that the following algorithm also does not always find an independent set of maximum total weight.

(c) Give an algorithm that takes an \( n \)-node path \( G \) with weights and returns an independent set of maximum total weight. Analyze the algorithm. The running time should be polynomial in \( n \), independent of the values of the weights.
Problem 3. [30 points - Proof problem]

Your friend is working as a camp counselor, and he is in charge of organizing activities for a set of junior-high-school-age campers. One of his plans is the following mini-triathalon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, then run 3 miles.

The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he or she is out and starts biking, a third contestant begins swimming . . . and so on.

Each contestant has a projected swimming time (the expected time it will take him or her to complete the 20 laps), a projected biking time (the expected time it will take him or her to complete the 10 miles of bicycling), and a projected running time (the time it will take him or her to complete the 3 miles of running).

Your friend wants to decide on a schedule for the triathalon: an order in which to sequence the starts of the contestants. Let’s say that the completion time of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathalon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts.

Note that participants can bike and run simultaneously, but at most one person can be in the pool at any time.

(a) What are the inputs and goal of this problem? Try to formalize your notations, to help you later with the proof.

(b) What’s the best order for sending people out, if one wants the whole competition to be over as early as possible?

(c) [Proof part - Submit separately] Prove that your proposed order is optimal.

Problem 4. [20 points]

Given the code for Dijkstra’s algorithm shared with you in class, and in your textbook, find the cost of the shortest paths from node $s$ to the rest of the nodes in the graph shown below. Make sure to trace the algorithm exactly as shown in the algorithm. To help you trace it, I have provided you with this table to complete with every iteration.
**Hint:** An initial dummy iteration is already done for your to get you started, with all temporary distances initially set to $\infty$.

<table>
<thead>
<tr>
<th>$S$</th>
<th>$V - S$</th>
<th>$d'(v), \forall v \in V - S$</th>
<th>Node to be added to $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>${s, A, B, C, D, t}$</td>
<td>$d'(s) = 0, d'(A) = d'(B) = d'(C) = d'(D) = d'(t) = \infty$</td>
<td>$s$</td>
</tr>
</tbody>
</table>

Finally, fill out the distances table below.

<table>
<thead>
<tr>
<th>Node</th>
<th>Shortest Distance from s</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td></td>
</tr>
</tbody>
</table>