Problem 1. [15 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 maximal 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 $v_1, v_2, ..., v_n$, 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.

```
The "heaviest-first" greedy algorithm
Start with S equal to the empty set
While some node remains in G
    Pick a node $v_i$ of maximum weight
    Add $v_i$ to S
    Delete $v_i$ and its neighbors from G
Endwhile
Return S
```

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

```
Let $S_1$ be the set of all $v_i$ where $i$ is an odd number
Let $S_2$ be the set of all $v_i$ where $i$ is an even number
(Note that $S_1$ and $S_2$ are both independent sets)
Determine which of $S_1$ or $S_2$ has greater total weight,
and return this one
```

(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.

(d) Implement the algorithm you wrote in part c. The input of the algorithm should be a 2D adjacency matrix representing the graph. Assume that each vertex is known by its index in the array. The output of the algorithm should be the maximal independent set in the graph.

Problem 2. [10 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>
<tr>
<td>${s}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>

**Problem 3. [15 points]**
Suppose you’re running a lightweight consulting business—just you, two associates, and some rented equipment. Your clients are distributed between the East Coast and the West Coast, and this leads to the following question.

Each month, you can either run your business from an office in New York (NY) or from an office in San Francisco (SF). In month $i$, you’ll incur an operating cost of $N_i$ if you run the business out of NY; you’ll incur an operating cost of $S_i$ if you run the business out of SF. (It depends on the distribution of client demands for that month.)

However, if you run the business out of one city in month $i$, and then out of the other city in month $i + 1$, then you incur a fixed moving cost of $M$ to switch base offices.

Given a sequence of $n$ months, a plan is a sequence of $n$ locations—each one equal to either NY or SF—such that the $i$th location indicates the city in which you will be based in the $i$th month. The cost of a plan is the sum of the operating costs for each of the $n$ months, plus a moving cost of $M$ for each time you switch cities. The plan can begin in either city.
The problem. Given a value for the moving cost $M$, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, find a plan of minimum cost. (Such a plan will be called optimal.)

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For $i = 1$ to $n$
  If $N_i < S_i$ then
    Output "NY in Month $i"
  Else
    Output "SF in Month $i"
End
```

(b) Implement an efficient algorithm that takes values for $n$, $M$, and arrays of operating costs $[N_1, \ldots, N_n]$ and $[S_1, \ldots, S_n]$, and returns the cost of an optimal plan.

Problem 4. [10 points]
Let us say that a graph $G = (V, E)$ is a near-tree if it is connected and has at most $n + 8$ edges, where $n = |V|$. Give an algorithm with running time $O(n)$ that takes a near-tree $G$ with costs on its edges, and returns a minimum spanning tree of $G$. You may assume that all the edge costs are distinct. You have to explain why the running time of your algorithm is $O(n)$. 