1 Mobile Business (34 points)

Let’s say that you have a great idea for a new food truck, and in order to save money you decide to run it out of your RV so you can live where you work. Each day \( i \) there is some demand for your food in Baltimore and some demand in Washington – let’s say you would make \( B_i \) dollars by being in Baltimore and \( W_i \) dollars by being in Washington. However, if you wake up in one city (due to being there the previous day) and want to serve in the other city, it costs you \( M \) dollars to drive there.

The goal in this problem is to devise a maximum-profit schedule. A schedule is simply an assignment of locations to days – for each day \( i \), the schedule says whether to serve in Baltimore or Washington. The profit of a schedule is the total profit you make, minus \( M \) times the number of times you have to move between cities. For the starting case, you can assume that on day 1 you wake up in Baltimore.

For example, let \( M = 10 \) and suppose that \( B_1 = 1, B_2 = 3, B_3 = 20, B_4 = 30 \) and \( W_1 = 50, W_2 = 20, W_3 = 2, W_4 = 4 \). Then the profit of the schedule \( \langle \text{Washington, Washington, Baltimore, Baltimore} \rangle \) would be \( W_1 + W_2 + B_3 + B_4 - 2M = 100 \), where one of the \( M \)’s comes from driving from Baltimore to Washington on day 1, and the other comes from driving from Washington to Baltimore and day 3. The profit of the schedule \( \langle \text{Washington, Baltimore, Baltimore, Washington} \rangle \) would be \( W_1 + B_2 + B_3 + W_4 - 3M = 50 + 3 + 20 + 4 - 30 = 47 \).

Given the fixed driving cost \( M \) and profits \( B_1, \ldots B_n \) and \( W_1, \ldots, W_n \), devise an algorithm that runs in \( O(n) \) time and computes the profit of an optimal schedule. As always, prove correctness and running time.

Solution. We can solve this problem in \( O(n) \) time using dynamic programming. Let \( OPT_B(i) \) be the optimal schedule for days 1 through \( i \), conditioned on the scheduling ending in Baltimore at day \( i \). Let \( OPT_W(i) \) be the same thing, except that we end in Washington. Given a schedule \( S \), let \( P(S) \) denote its total profit. Then we get the following optimal substructure lemma.

**Lemma 1**

\[
P(OPT_B(i)) = B_i + \max\{P(OPT_B(i - 1)), P(OPT_W(i - 1)) - M)\}
\]

\[
P(OPT_W(i)) = W_i + \max\{P(OPT_W(i - 1)), P(OPT_B(i - 1)) - M\}
\]

**Proof:** We prove this by contradiction. Consider \( OPT_B(i) \). At time \( i - 1 \), it is either in Washington or in Baltimore. Suppose that it is in Baltimore. Then we claim that \( OPT_B(i) \) is just \( OPT_B(i - 1) \)
followed by staying in Baltimore at time $i$. To see this, suppose that it is false. Then since we are assuming that at time $i - 1$ the schedule $OPT_B(i)$ is in Baltimore, this means that it consists of some schedule $S$ which ends in Baltimore at time $i - 1$ together with staying in Baltimore at time $i$. If we use $OPT_B(i - 1)$ instead of $S$, then we get a schedule ending in Baltimore at time $i$ with larger total profit. Hence if $OPT_B(i)$ is in Baltimore at time $i - 1$, it is just $OPT_B(i - 1)$ followed by staying in Baltimore at time $i$.

Similarly, suppose that $OPT_B(i - 1)$ is in Washington at time $i - 1$. Then it must consist of $OPT_W(i - 1)$ followed by moving to Baltimore at time $i$, since if it uses some other schedule $S$ ending in Washington at time $i - 1$ as its prefix we would get a schedule with profit

$$P(OPT_W(i)) = P(S) + B_i - M < P(OPT_W(i - 1)) + B_i - M,$$

which contradicts the definition of $OPT_W(i)$.

Thus if $OPT_B(i)$ is in Baltimore at time $i - 1$ it has profit $B_i + P(OPT_B(i - 1))$, while if it is in Washington at time $i - 1$ it has profit $B_i + P(OPT_W(i - 1)) - M$. Since one of these two must be the case, we get that $P(OPT_B(i)) = B_i + \max\{P(OPT_B(i - 1)), P(OPT_W(i - 1)) - M\}$.

A symmetric argument proves that $P(OPT_W(i)) = W_i + \max\{P(OPT_W(i - 1)), P(OPT_B(i - 1)) - M\}$.

This gives rise to the following natural DP algorithm. Here is pseudocode for filling in these tables and extracting the optimal schedule from them:

```
B ← array of length $n + 1$;
W ← array of length $n + 1$;
B[0] ← 0;
W[0] ← $-\infty$;

\\ fill in the tables
for $i$ from 1 to $n$ {
  if $B[i - 1] > W[i - 1] - M$
    $B[i] = B[i - 1] + B_i$;
  else
    $B[i] = W[i - 1] - M + B_i$;
  if $W[i - 1] > B[i - 1] - M$
    $W[i] = W[i - 1] + W_i$;
  else
    $W[i] = B[i - 1] - M + W_i$;
}
Return $\max\{B[n], W[n]\}$
```

The runtime analysis is very straightforward. The loop runs $n$ times, and each iteration takes a constant amount of time, so its runtime is $O(n)$.

We can prove correctness using induction. In particular, we will prove by induction that $B[i] = P(OPT_B(i))$ and $W[i] = P(OPT_W(i))$. For the base case, we set $B[0]$ and $W[0]$ appropriately. For the inductive step, we have that

$$B[i] = \max\{B[i - 1] + B_i, W[i - 1] - M + B_i\}$$
$$= \max\{P(OPT_B(i - 1)) + B_i, P(OPT_W(i - 1)) - M + B_i\}$$
$$= P(OPT_B(i))$$
where the first equality is by the functioning of the algorithm, the second is by induction, and the third is by Lemma 1. A similar induction gives that $W[i] = P(OPT_W(i))$.

2 Consistent Lifetimes (34 points)

Suppose that you are an historian, and you are trying to figure out possible dates for when various historical figures may have lived. In particular, there are $n$ people $P_1, P_2, \ldots, P_n$ who you are studying. Based on your research, you have discovered a collection of $m$ facts about when these people lived relative to each other. Each of these facts has one of the following two forms:

- For some $i$ and $j$, person $P_i$ died before person $P_j$ was born; or
- For some $i$ and $j$, the lifetimes of $P_i$ and $P_j$ overlapped.

Unfortunately, since the historical record is never fully trustworthy, it’s possible that some of these facts are incorrect. Design an $O(n + m)$ time algorithm to at least check whether they are internally consistent, i.e., whether there is a birth and a death date for each person so that all of the facts are true. As always, prove running time and correctness (i.e., prove that if there is a possible set of dates then your algorithm will return yes, and if no such dates are possible then your algorithm will return no).

**Solution.** We first create a directed graph $G = (V, E)$ where there are two vertices $s_i, t_i$ for each person $P_i$. We place a directed edges from $s_i$ to $t_i$ for all $i \in [n]$. For every fact of the form “$P_i$ died before $P_j$ was born”, we put a directed edge from $t_i$ to $s_j$. And for every fact of the form “the lifetimes of $P_i$ and $P_j$ overlapped”, we add a directed edge $(s_j, t_i)$ and a directed edge $(s_i, t_j)$. We then use DFS to try to construct a topological sort of this graph. If we create a topological sort then we return “yes”, and otherwise (if we cannot create a topological sort, i.e., we find a back edge) we return “no”.

We first analyze the running time of this algorithm. Creating the nodes and the $(s_i, t_i)$ edges takes $O(n)$ time. Then we can iterate through the fact and for each fact create the appropriate one or two directed edges in $O(1)$ time per fact, for a total of $O(m)$ time. Thus creating the graph takes $O(n + m)$ time, and the number of edges in the graph is $O(n + m)$. So the running time of our topological sorting algorithm is $O(n + m)$, and hence the total running time of the algorithm is $O(n + m)$.

For correctness, first suppose that the correct answer is “yes”. Then consistent dates exist. Order the vertices of $V$ so that $s_i$ corresponds to the birth date of person $P_i$ and $t_i$ corresponds to the death of person $P_i$. In other words, order the nodes according to the true dates (which must exist by assumption). Let $\pi : V \to \{1, 2, \ldots, 2n\}$ be this ordering. We claim that this is a topological ordering, which will imply that our algorithm will correctly return “yes” (since a topological ordering exists, our algorithm will find one). To see that this is a topological ordering, let’s consider the various edges. Clearly any edge of the form $(s_i, t_i)$ has $\pi(s_i) < \pi(t_i)$, since every person is born before they die. Any edge of the form $(t_i, s_j)$ is because of a fact that $P_i$ died before $P_j$ was born, and since the dates are consistent with the facts we know that $\pi(t_i) < \pi(s_j)$. Now consider an edge of the form $(s_i, t_j)$, which is due to a fact that the lifetimes of $P_i$ and $P_j$ overlap. Since the dates are consistent with the fact, we know that their lifetimes do overlap, and hence $P_i$ was born before $P_j$ dies, and thus $\pi(s_i) < \pi(t_j)$.
Now to prove that our algorithm is correct when the answer is “no”, we will prove the contrapositive, i.e., that if our algorithm returns “yes” then there is a set of consistent dates. So suppose that our algorithm returns “yes”. This means that it found a topological sort of $G$. Assign dates in order of this topological sort. To see that these are consistent dates, first note that since the $(s_i, t_i)$ edges point forward in this ordering, everyone is born before they die. Now consider some fact of the first form. Then since there is an edge $(t_i, s_j)$ in the graph by construction, we must have assigned $P_i$ a death date before the birthdate of $P_j$. So the dates are consistent with that fact. Now consider a fact of the second form. Then since we included an edge $(s_i, t_j)$, we know that in according to our dates, $P_i$ was born before $P_j$ died. And similarly, since we included an edge $(s_j, t_i)$, we know that $P_j$ was born before $P_i$ died according to our dates. Thus our dates are consistent with this fact. Since we have shown this for all facts, this implies that our dates are consistent with all facts and thus there is a set of consistent dates.

3 Shortest Paths (33 points)

(a) We saw in class that Dijkstra’s algorithm for computing shortest paths in a directed graph does not work if there are negative edge lengths, even if there are no negative-length cycles. Consider the following idea to fix this.

First add a large enough positive constant to every edge weight so that the “revised” edge lengths are all positive (for example, you could add the value $1 + \max_{v \in V} |\text{len}(v)|$). To find the shortest path between two nodes $s$ and $t$ in the original graph, run Dijkstra’s algorithm using the revised edge lengths to find the shortest path between $s$ and $t$ using the revised lengths, and return the path which it finds.

Does this work? That is, will this always return the shortest path under the original edge lengths (assuming that there are not any negative-weight cycles initially)? If so, give a proof. If not, give a counterexample (and explain it).

Solution. False. Consider the following graph:
If we add $\alpha = 1 + \max_{v \in V} |\text{len}(v)|$ to every edge in this graph, then when we run Dijkstra’s algorithm to find the shortest path from $s$ to $t$ on the revised weights we will get back the bottom path, with length $1 + \alpha + 1 + \alpha = 6$, since the top path will have length $1 + \alpha + 1 + \alpha + 1 + \alpha - 1 + \alpha - 1 + \alpha = 1 + 5\alpha = 11$. But under the original weights, the unique shortest path from $s$ to $t$ is the top path with total length 1.

(b) Now suppose that instead of adding the same value to every length, we instead multiply every length by the same value $\alpha > 0$ (note that this preserves the sign of each edge length). Let $P$ be a shortest path from $s$ to $t$ under the original edge lengths. Is $P$ still a shortest path from $s$ to $t$ under the new edge lengths? If yes, give a proof. If no, give a counterexample (and explain it).

Solution. True. Consider some arbitrary path $P'$ from $s$ to $t$. Under the original length, the length of $P'$ is $\text{len}(P') = \sum_{e \in P'} \text{len}(e)$. Under the new edge lengths, the length of $P'$ is $\sum_{e \in P'} \alpha \cdot \text{len}(e) = \alpha \sum_{e \in P'} \text{len}(e) = \alpha \cdot \text{len}(P')$. So the length of every path under the new lengths is exactly $\alpha$ times its length under the old weights. Since $\alpha > 0$, this implies that if $\text{len}(P) \leq \text{len}(P')$ (under the old weights), then the length of $P$ is at most the length of $P'$ under the new weights. Hence if $P$ is a shortest path under the original lengths, then it is also a shortest path under the new lengths.