1. (10 pts.) Consider the following algorithm for single-source shortest paths.

\textbf{Exercise 1} \((G, s, w)\)
\begin{algorithmic}
\For {each \( v \in G.V \)}
\State \( v.d \gets \infty \)
\EndFor
\State \( s.d \gets 0 \)
\State \( continue \gets \text{TRUE} \)
\While {\( continue \neq \text{FALSE} \)}
\State \( continue \gets \text{FALSE} \)
\For {each \( (u, v) \in G.E \)}
\State \( x \gets v.d \)
\State \( v.d \gets \min(v.d, u.d + w(u, v)) \)
\If {\( x \neq v.d \)}
\State \( continue \gets \text{TRUE} \)
\EndIf
\EndFor
\EndWhile
\end{algorithmic}

What is the running time of this algorithm? Justify your answer. You might have different answers for different cases. Use \( V \) = number of vertices and \( E \) = number of edges.

2. (10 pts.) Consider the following algorithm for all-pairs shortest paths.

\textbf{Exercise 2} \((G, s, w)\)
\begin{algorithmic}
\For {each \( (u, v) \in G.V \times G.V \)}
\State \( D[u, v] \gets \infty \)
\EndFor
\For {each \( v \in G.V \)}
\State \( D[v, v] \gets 0 \)
\EndFor
\State \( continue \gets \text{TRUE} \)
\While {\( continue \neq \text{FALSE} \)}
\State \( continue \gets \text{FALSE} \)
\For {each \( (v_1, v_2) \in G.E \)}
\For {each \( u \in G.V \)}
\State \( x \gets D[u, v_2] \)
\State \( D[u, v_2] \gets \min(D[u, v_2], D[u, v_1] + w(v_1, v_2)) \)
\If {\( x \neq D[u, v_2] \)}
\State \( continue \gets \text{TRUE} \)
\EndIf
\EndFor
\EndFor
\EndWhile
\end{algorithmic}

What is the running time of this algorithm? Justify your answer. You might have different answers for different cases. Use \( V \) = number of vertices and \( E \) = number of edges.
3. (10 pts.) Show the execution of the Edmonds-Karp algorithm on the flow network of the following figure.

```
3. (10 pts.) Show the execution of the Edmonds-Karp algorithm on the flow network of the following figure.
```

4. (10 pts.) Instead of using breadth-first search to find an augmenting path, suppose that we find the path with the maximum flow. In pseudocode, provide an \( O((V + E) \log(V + E)) \) algorithm for finding this path.

5. (10 pts.) Construct the string-matching automaton for the pattern \( P = abaaa \) and illustrate its operation on the text string \( T = ababaaabaaaaabaaabbb \).

6. (10 pts.) Suppose that a set of \( n \) meetings need to be scheduled in two rooms over a period of \( T \) time units (each room can be scheduled for up to \( T \) time units). The length of meeting \( i \) lasts \( t_i \) time units, where \( t_i \) is one of two possible values \( a \) and \( b \), that is, each meeting lasts for either \( a \) units long or \( b \) units long. You are free to schedule any meeting at any time, but once meeting \( i \) starts, it stays in that room for \( t_i \) time units. Provide an efficient algorithm to determine if there is a feasible schedule.

7. (10 pts.) Suppose that a set of \( n \) meetings need to be scheduled in two rooms over a period of \( T \) time units. The length of meeting \( i \) lasts \( t_i \) time units, where \( t_i \) can be any positive integer. You are free to schedule any meeting at any time, but once meeting \( i \) starts, it stays in that room for \( t_i \) time units. Show that determining whether a feasible schedule exists is NP-complete. Hint: use SUBSET-SUM.

8. (10 pts.) Suppose that a student has \( n \) requirements, where requirement \( i \) can be satisfied by taking one of the courses in set \( S_i \). It happens that a single course might fulfill more than one requirement, so that the student can take fewer than \( n \) courses to satisfy the requirements. Suppose the student simply want to find a “non-redundant” set of courses \( C \), meaning the courses \( C \) satisfy all \( n \) requirements, but no proper subset of \( C \) satisfies all the requirements. Provide an efficient algorithm to find a non-redundant set.

9. (10 pts.) Consider the same problem as the previous exercise, but in this case, the student wants to find the minimum-size set of courses to meet the requirements. Show that determining whether the minimum-size is \( k \) or less is NP-complete. Hint: Use VERTEX-COVER.

10. (10 pts.) Show that finding the longest simple path between two vertices in a graph is NP-hard.