1. **Floyd’s Algorithm** Given a labeled directed graph $G = (V, E)$, find for each pair of vertices $v, w \in V$ the cost of the shortest (that is, the least-cost) path from $v$ to $w$ in $G$. The standard dynamic programming algorithm for this, the “all-pairs shortest paths” problem, is Floyd’s algorithm, p. 310 in Levitin.

   (i) Use Floyd’s algorithm to fill in a cost matrix for each of the graphs shown below. Write out your initial matrix, and the matrix after each iteration. (If you generate the matrices using a program, paste the output from the program into your answer to this question.)

   (ii) *Using the matrices generated in your answer to part (i), find the shortest path from vertex 1 to vertex 5. That is: Floyd’s algorithm tells you the length of the shortest path: I’m asking you to reconstruct the route that this path takes. It’s essential that you explain how you extract the route from the matrices: eyeballing the figure and writing down the route will get zero marks!"
2. **Dijkstra’s Algorithm** Apply Dijkstra’s algorithm to find all the shortest paths from vertex $j$ in the graph shown below. Construct a table (similar to Levitin’s Figure 9.11) that shows, at each step, the vertex added to the tree $V_T$, the shortest path from that vertex back to $j$, and the vertices remaining in the priority queue $Q$ together with their priorities and the label of their closest tree node.

```
\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{graph.png}
\end{figure}
```

3. **Knapsack Algorithm** The Knapsack problem is defined as: given $n$ items of known weights $w_1, w_2, ..., w_n$ and values $v_1, v_2, ..., v_n$, and a knapsack of capacity $W$, find the most valuable subset of items that fit into the knapsack. This problem can be solved with dynamic programming using memoization.

(i) Using your favorite programming language, or by hand, construct the dynamic programming table for the following two instances of the Knapsack problem:

\begin{align*}
\begin{array}{|c|ccccccc|}
\hline
i & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
v_i & 43 & 35 & 21 & 17 & 24 & 49 \\
v_i & 6 & 5 & 2 & 1 & 4 & 3 \\
\hline
\end{array}
\end{align*}

for knapsacks of capacity $W = 7$ and $W = 9$.

Don’t compute table entries unnecessarily! In your table, mark entries that have not been computed at all with a dash, and entries that are re-used with an asterisk. Produce a table like the following (this one is for a different problem instance):

\begin{align*}
\begin{array}{|c|ccccccccc|}
\hline
W=4, £10 | & 0 & 0 & 0* & 10 & 10 & 10* & 10 & 10 & 10 \\
W=1, £17 | & 17 & - & 17* & - & 27* & 27 & - & 27 & - \\
W=3, £6 | & - & - & 33 & - & 33 & 33 & - & 43 & - \\
\hline
\end{array}
\end{align*}

(ii) Use the tables that you produce in part (i) to figure out what combination of items makes up the optimal solution.