Solutions to Problem Set 8

Ekkapot Charoenwanit
Efficient Algorithms

Problem 1. BFS and DFS

Consider the following undirected graph G.

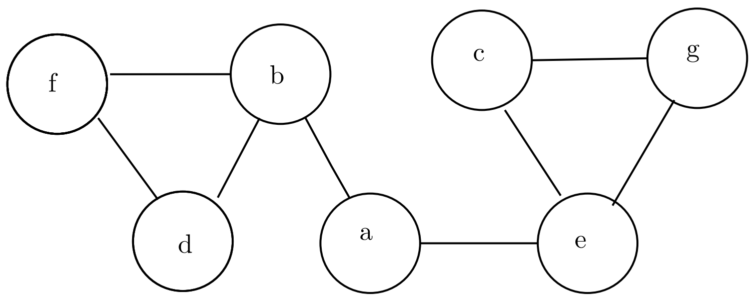

Figure 1: Undirected Graph $G = (V, E)$

(1) Use BFS to traverse the graph G in Figure 1 in the alphabetical order starting from vertex a. Construct a BFS search tree and identify the frontier set at each level.

Solution:
Level-0 frontier: $\{a\}$
Level-1 frontier: $\{b, e\}$
Level-2 frontier: $\{d, f, c, g\}$

Figure 2 shows the BFS tree produced by our BFS traversal on the graph G.

The contents of the Queue Q change as follows:
$t = 0$: []
$t = 1$: $[a]$
$t = 2$: $[b, e]$
$t = 3$: $[e, d, f]$
$t = 4$: $[d, f, c, g]$
$t = 5$: $[f, c, g]$
$t = 6$: $[c, g]$
$t = 7$: $[g]$
$t = 8$: []

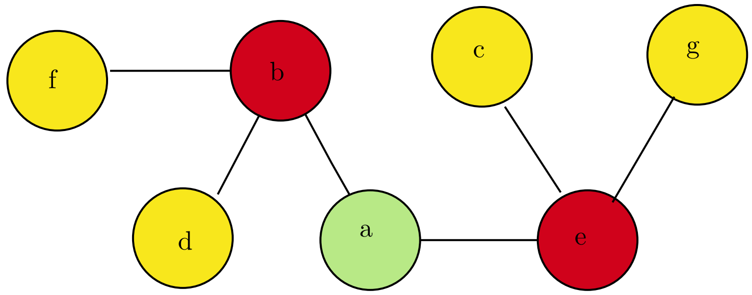

Figure 2: BFS Tree

(2) Use DFS to traverse the graph G in Figure 1 in the alphabetical order starting from vertex a. Identify the type of each edge of G.

Solution: Figure 3 shows the DFS tree produced by our DFS traversal on the graph G. All the edges that appear in Figure 3 are **tree** edges. The remaining edges (b, f) and (e, g), which appear only in Figure 1 but are excluded from Figure 3 are **back** edges.

Note: In undirected graphs, there are only two types of edges, namely, **tree** and **back** edges.

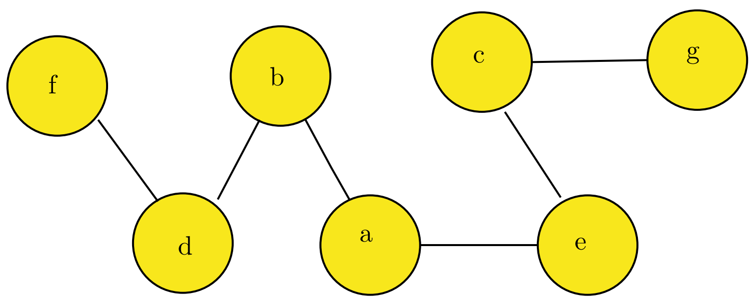

Figure 3: DFS Tree

Problem 2. Applications of BFS and DFS

(1) How can you detect a cycle in an **undirected** graph $G = (V, E)$ with BFS?

Solution: Let's say that BFS is now inspecting the neighbor of a vertex u. All the neighbors v of u need inspecting; $v \in Adj[u]$. If BFS finds any vertex v that has been visited and is not the parent of u, G then contains a cycle. If BFS cannot find such a vertex v for all $u \in V$, G contains no cycle.

(2) Given an **unweighted** graph $G = (V, E)$ and a designated root vertex r, explain how to compute a shortest path from r to every other nodes in V in linear time in the number of vertices $|V|$ and edges $|E|$. Assume that a shortest path from a vertex i to a vertex j in an **unweighted** graph is defined as a path with the minimum number of edges.

Solution: The BFS tree produced by BFS traversal provides a path with the minimum number of edges from the root r to every other reachable vertex.
Therefore, for an unweighted graph, BFS computes a shortest path from the root r to every other reachable vertex in G; we can assign 1 to the weights of all the edges.
Therefore, BFS can compute a shortest path from r to every other reachable vertex in $\mathcal{O}(V + E)$ time, which is linear in $|V|$ and $|E|$ as required.

(3) Based on your idea in (2), compute a shortest path from a to every other vertex $v \in V$ of the graph in Figure 1.

Solution: Figure 2 shows a shortest path from a to every other reachable vertex.
All the vertices in the Level-1 frontier set have a distance of 1 from a, whereas all the vertices in the Level-2 frontier set have a distance of 2 from a.
Note that since the graph is connected, there are no vertices that are not reachable from a.

Problem 3. Dijkstra's Algorithm

(1) Solve the following shortest path problem, with vertex a as a source.

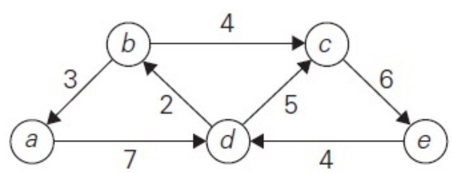

Figure 4: Solve the single-source shortest path with a as a source.

Solution:

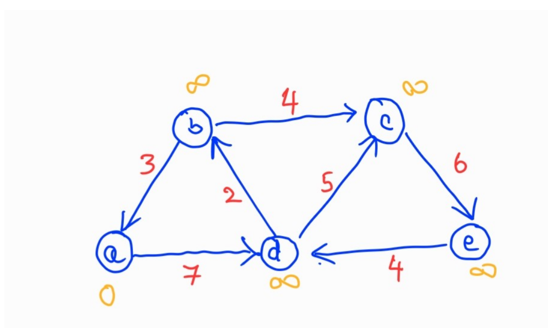

Figure 5: 1^{st} iteration

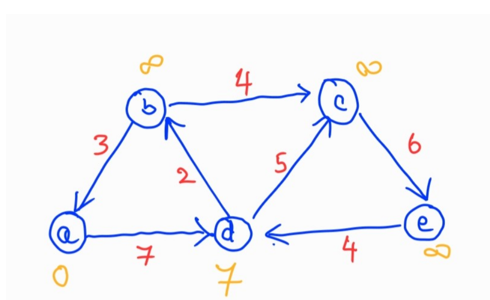

Figure 6: 2^{nd} iteration

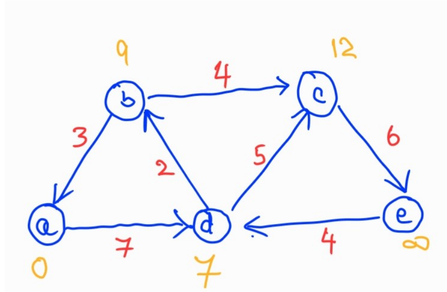

Figure 7: 3^{rd} iteration

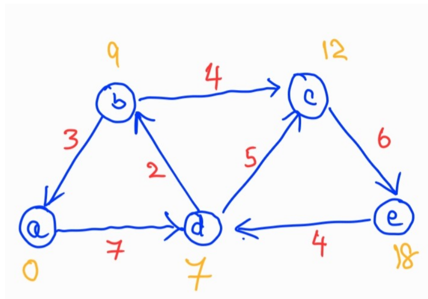

Figure 8: 4^{th} iteration

In each iteration, the algorithm extracts a vertex u with the minimum **d-value** and relaxes all the **outgoing** edges of u to update the d-values of the neighbor vertices $v \in Adj[u]$. According to the solution in Figure 8, $\delta(a, a) = 0$, $\delta(a, b) = 9$, $\delta(a, c) = 12$, $\delta(a, d) = 7$ and $\delta(a, e) = 18$.

(2) How do you apply Dijkstra's algorithm on a **directed** graph $G = (V, E, w)$ to find a shortest path from every vertex $v \in V$ to a given destination vertex $t \in V$ in the G?

Solution:

- Reverse the direction of all the edges in the original graph G to obtain the **transpose** graph G^T.
- Run Djkstra's algorithm on the new graph G^T, starting from the vertex t.

(3) Apply the algorithm you proposed in (2) to the graph in Figure 4 with a as the destination.

Solution:

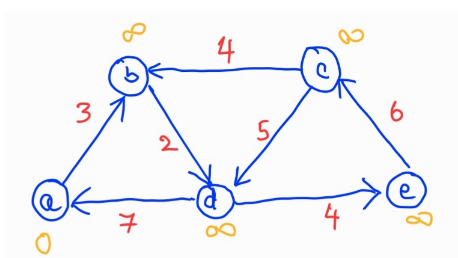

Figure 9: 1^{st} iteration

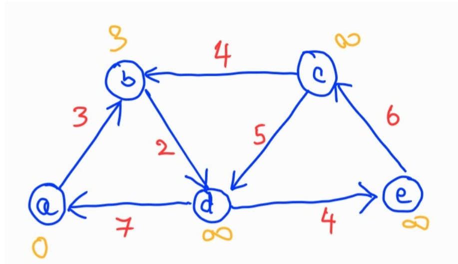

Figure 10: 2^{nd} iteration

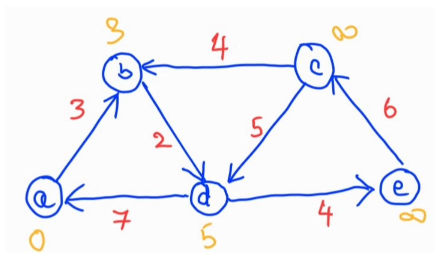

Figure 11: 3^{rd} iteration

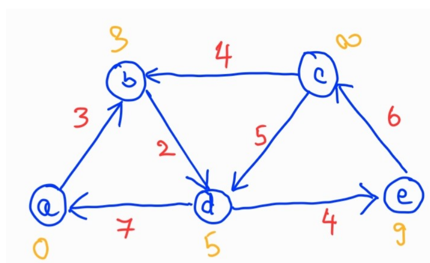

Figure 12: 4^{th} iteration

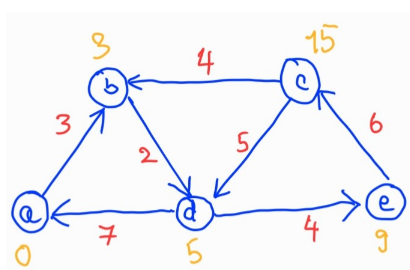

Figure 13: 5^{th} iteration

(4) Analyze the running time of your algorithm in (2) in terms of $|V|$ and $|E|$.

Solution: In Algorithm 1, reversing the direction of the edges (lines 2-5) requires $\Theta(V + E)$ time and Djkstra's algorithm (line 7) (implemented with a **min-heap-based** priority queue) requires $\mathcal{O}((V + E)\log V)$ time. In total, the algorithm takes $\mathcal{O}((V + E)\log V)$ time.

Note:

- the complexity of reversing the direction of the edges is asymptotically subsumed by that of Djkstra's algorithm
- the complexity of constructing a new adjacency list (line 2) is $\mathcal{O}(1)$

Algorithm 1 implements a single-destination shortest-path algorithm

1: **procedure** SINGLE-DESTINATION($G = (V, E)$)
2:     CREATE $rAdj$
3:     **for** each $u \in V$ **do**
4:         **for** each $v \in Adj[u]$ **do**
5:             APPEND u to $rAdj[v]$
6:     DEFINE G^T as $rAdj[v]$
7:     DJKSTRA(G^T)

(5) How do you apply Dijkstra's algorithm on a **undirected** graph $G = (V, E, w)$ to find a shortest path from every vertex $v \in V$ to a given destination vertex $t \in V$ in G?

Solution: We can treat an undirected graph as a directed one where every edge is **bidirectional**. Therefore, we can simply run Djkstra's algorithm on G, starting from the vertex t, to compute a shortest path from t to every other vertex in G.

(6) Analyze the running time of your algorithm in (5) in terms of $|V|$ and $|E|$.

Solution: This is simply Djkstra's algorithm so its running time is exactly that of Djkstra's algorithm. With a **min-heap-based** priority queue, the running time is $\mathcal{O}((|V| + |E|) \log |V|)$.

Problem 4. Bellman-Ford

(1) Apply the Bellman-Ford algorithm to the graph in Figure 4 with vertex a as a source vertex. Show your work in each pass. Determine whether $d[v] = \delta(s, v)$ before $|V| - 1$ passes for all $v \in V$.

Solution: Since there are $|V| = 5$ vertices, the Bellman-Ford algorithm requires $|V| - 1 = 4$ passes. In this problem, we will relax the edges $(u, v) \in E$ in the following order in each pass of the algorithm: (b, a), (a, d), (d, b), (b, c), (d, c), (e, d) and (c, e).

In this problem, after the first pass, all the d-values converge to the delta-values so we show only the first pass of the algorithm while all the d-values stay the same throughout the remaining three passes.

1^{st} Pass:

Relaxing (b, a) does not update the d-value of a.

Relaxing (a, d) updates the d-value of d from ∞ to 7 as shown in Figure 14.

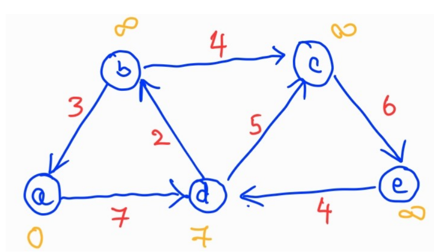

Figure 14: Relax (a, d) in the 1^{st} pass

Relaxing (d, b) updates the d-value of d from ∞ to 9 as shown in Figure 15.

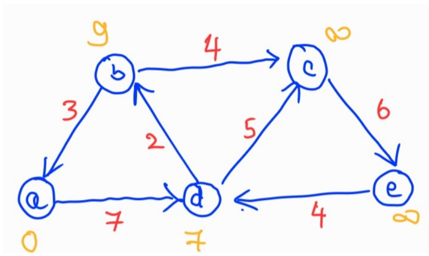

Figure 15: Relax (d, b) in the 1^{st} pass

Relaxing (b, c) updates the d-value of c from ∞ to 13 as shown in Figure 16.

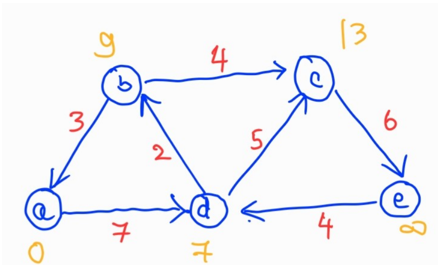

Figure 16: Relax (b, c) in the 1^{st} pass

Relaxing (d, c) updates the d-value of c from 13 to 12 as shown in Figure 17.

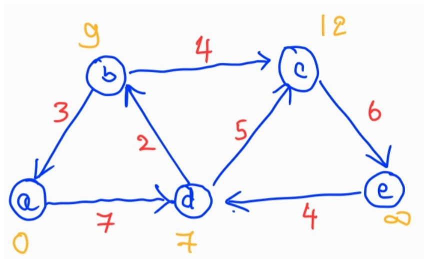

Figure 17: Relax (d, c) in the 1^{st} pass

Relaxing (e, d) does not update the d-value of d.
Relaxing (c, e) updates the d-value of e from ∞ to 18 as shown in Figure 18.

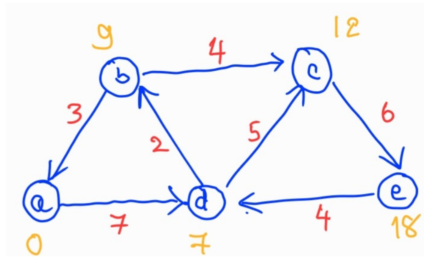

Figure 18: Relax (c, e) in the 1^{st} pass

After the 1^{st} pass, the d-values stay the same throughout the remaining three passes. Although, in fact, the first pass suffices to discover shortest paths from a to every other vertex, the Bellman-Ford algorithm needs to pessimistically execute the remaining three passes.

(2) Given a directed graph $G = (V, E)$, suppose that G contains negative-weight cycles. Modify the Bellman-Ford algorithm so that it sets $v.d = -\infty$ for all vertices v for which there is a **negative-weight cycle** on some path from s to v.

Solution: In the extra pass over all the edges, instead of simply returning true, the modified algorithm marks vertices whose d-values can still be improved, hence indicating they are on some negative-weight cycles. This marking process is carried out recursively to update the d-values of these marked vertices to $-\infty$.

Algorithm 2 implements the marking of vertices on a negative-weight cycle

procedure MARK(v)
    if $v \neq NILL$ & $d[v] \neq -\infty$ **then**
        $d[v] = -\infty$
        MARK($\pi[v]$)

Algorithm 3 implements Modified Bellman-Ford

procedure MODIFIED-BELLMAN-FORD($G = (V, E, w), s$)
    INITIALIZE(G, s)
    for $i = 1$ to $i = |V|$ **do**
        for each edge $(u, v) \in E$ **do**
            RELAX(u, v, w)
    for each edge $(u, v) \in E$ **do**
        if $d[v] > d[u] + w(u, v)$ **then**
            $c[v] = True$
    for each vertex $v \in V$ **do**
        if $c[v]$ **then**
            MARK($\pi[v]$)

(3) How do you detect if a directed graph $G = (V, E)$ has a **negative-weight cycle** using the Bellman-Ford algorithm? Analyze the running time of your solution in terms of $|V|$ and $|E|$.

Solution: Note that one run of the Bellman Ford algorithm may not suffice to detect the presence of a negative-weight cycle if that cycle is not reachable from the given source.
To avoid the need to run the Bellman-Ford algorithm multiple times from multiple sources, we can simply add one extra vertex and connect it to the other $|V|$ vertices in the original graph G with directed edges with weight 0. Doing so does not introduce any new cycle into the new graph $G^{'} = (V^{'}, E^{'})$, and this is what we do in Johnson's algorithm.

We can simply run the Bellman-Ford algorithm on the new graph G' with the new vertex as a source so the running time is $\mathcal{O}((|V'|)(|E'|)) = \mathcal{O}((|V|+1)(|E|+|V|)) = O(|V||E|+|V|^2)$.

Problem 5. Floyd-Warshall and Johnson's algorithm

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Figure 19: Apply the Floyd-Warshall algorithm to the following weight matrix.

(1) Apply the Floyd-Warshall algorithm to the graph represented by the weight matrix in Figure 19 to find shortest paths amoung all pairs of vertices $u, v \in V$. Give the matrix $D^{(k)}$ in each step k.

Solution: $D = D^{(0)} = \begin{pmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{pmatrix}$

In the following, we will show only the d-values $d_{ij}^{(k)}$ of iterations k that change from the d-values $d_{ij}^{(k-1}$ from the previous iterations $k-1$.

$k = 1$:
$d_{25}^{(1)} = \min(d_{25}^{(0)}, d_{21}^{(0)} + d_{15}^{(0)}) = \min(\infty, 6+8) = 14$
$d_{52}^{(1)} = \min(d_{52}^{(0)}, d_{51}^{(0)} + d_{12}^{(0)}) = \min(\infty, 3+2) = 5$
$d_{54}^{(1)} = \min(d_{54}^{(0)}, d_{51}^{(0)} + d_{14}^{(0)}) = \min(\infty, 3+1) = 4$

Thus, we get $D^{(1)} = \begin{pmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \mathbf{14} \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \mathbf{5} & \infty & \mathbf{4} & 0 \end{pmatrix}$ after iteration $k = 1$.

$k = 2$:
$d_{13}^{(2)} = \min(d_{13}^{(1)}, d_{12}^{(1)} + d_{23}^{(1)}) = \min(\infty, 2+3) = 5$
$d_{53}^{(2)} = \min(d_{53}^{(1)}, d_{52}^{(1)} + d_{23}^{(1)}) = \min(\infty, 5+3) = 8$

Thus, we get $D^{(2)} = \begin{pmatrix} 0 & 2 & \mathbf{5} & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & 14 \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \mathbf{8} & 4 & 0 \end{pmatrix}$ after iteration $k = 2$.

$k = 3$: no changes

Thus, we get $D^{(3)} = \begin{pmatrix} 0 & 2 & 5 & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & 14 \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & 8 & 4 & 0 \end{pmatrix}$ after iteration $k = 3$.

$k = 4$:

$d_{13}^{(4)} = \min(d_{13}^{(3)}, d_{14}^{(3)} + d_{43}^{(3)}) = \min(5, 1+2) = 3$
$d_{15}^{(4)} = \min(d_{15}^{(3)}, d_{14}^{(3)} + d_{45}^{(3)}) = \min(8, 1+3) = 4$
$d_{25}^{(4)} = \min(d_{25}^{(3)}, d_{24}^{(3)} + d_{45}^{(3)}) = \min(14, 2+3) = 5$
$d_{35}^{(4)} = \min(d_{35}^{(3)}, d_{34}^{(3)} + d_{45}^{(3)}) = \min(\infty, 4+3) = 7$
$d_{53}^{(4)} = \min(d_{53}^{(3)}, d_{54}^{(3)} + d_{43}^{(3)}) = \min(8, 4+2) = 6$

Thus, we get $D^{(4)} = \begin{pmatrix} 0 & 2 & \mathbf{3} & 1 & \mathbf{4} \\ 6 & 0 & 3 & 2 & \mathbf{5} \\ \infty & \infty & 0 & 4 & \mathbf{7} \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \mathbf{6} & 4 & 0 \end{pmatrix}$ after iteration $k = 4$.

$k = 5$:

$d_{31}^{(5)} = \min(d_{31}^{(4)}, d_{35}^{(4)} + d_{51}^{(4)}) = \min(\infty, 7+3) = 10$
$d_{32}^{(5)} = \min(d_{32}^{(4)}, d_{35}^{(4)} + d_{52}^{(4)}) = \min(\infty, 7+5) = 12$
$d_{41}^{(5)} = \min(d_{41}^{(4)}, d_{45}^{(4)} + d_{51}^{(4)}) = \min(\infty, 3+3) = 6$
$d_{42}^{(5)} = \min(d_{42}^{(4)}, d_{45}^{(4)} + d_{52}^{(4)}) = \min(\infty, 3+5) = 8$

Thus, we get $D^{(5)} = \begin{pmatrix} 0 & 2 & 3 & 1 & 4 \\ 6 & 0 & 3 & 2 & 5 \\ \mathbf{10} & \mathbf{12} & 0 & 4 & 7 \\ \mathbf{6} & \mathbf{8} & 2 & 0 & 3 \\ 3 & 5 & 6 & 4 & 0 \end{pmatrix}$ after iteration $k = 5$.

(2) Apply the Floyd-Warshall algorithm to the graph represented by the weight matrix in Figure 19 to find the transitive closure of G. Give the matrix $T^{(k)}$ in each step k.

Solution: $T = T^{(0)} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

$T^{(1)} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}$ after iteration $k = 1$.

$T^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ after iteration $k = 2$.

$T^{(3)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ after iteration $k = 3$.

$T^{(4)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ after iteration $k = 4$.

$$T^{(5)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ after iteration } k = 5.$$

Therefore, we can conclude from $T^{(5)}$ that there are always paths between all pairs of vertices in this directed graph.

Note that it is recommended you solve this problem with a computer program as solving it on paper can be error-prone.

(3) How can you detect the presence of negative-weight cycles in a directed graph using the output matrix of the Floyd-Warshall algorithm?

Solution:
Method I: Modify the Floyd-Warshall algorithm so that it runs one more iteration and check if any of the **d-values** changes during the extra iteration. If this is the case, it means there is at least one negative-weight cycle in the graph.

Method II: Run the Floyd-Warhsall algorithm and check if any of the diagonal entries $d_{ii}^{(n)}$ are negative.

(4) Apply Johnson's algorithm to the graph in Figure 20. Show the values of h and $\hat{w}$.

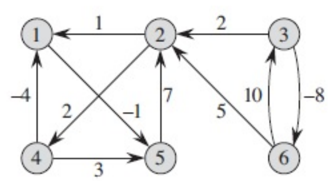

Figure 20: Apply Johnson's algorithm to the graph.

Solution:
Add one extra vertex and directed edges with weight 0 connecting this vertex to the other vertices in the original graph G to obtain $G^{'}$ as shown in Figure 21 and then run the Bellman-Ford algorithm on $G^{'}$.

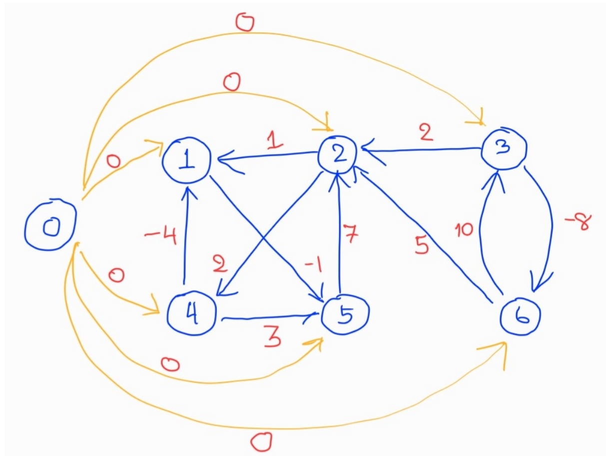

Figure 21: Apply the Bellman-Ford algorithm on the augmented graph $G^{'}$, starting from vertex 0 as a source.

The δ-value for each vertex $v \in V$ computed by the Bellman-Ford algorithm denoted by $h(v)$ is shown in Table 1.

v	$h(v)$
1	-5
2	-3
3	0
4	-1
5	-6
6	-8

Table 1: $h(v)$ for each $v \in V$

We can now re-weight all the edges with the following equation:

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

Then, we apply Djkstra's algorithm on the re-weighted graph, starting from each vertex as a source, as shown in Figure 22.

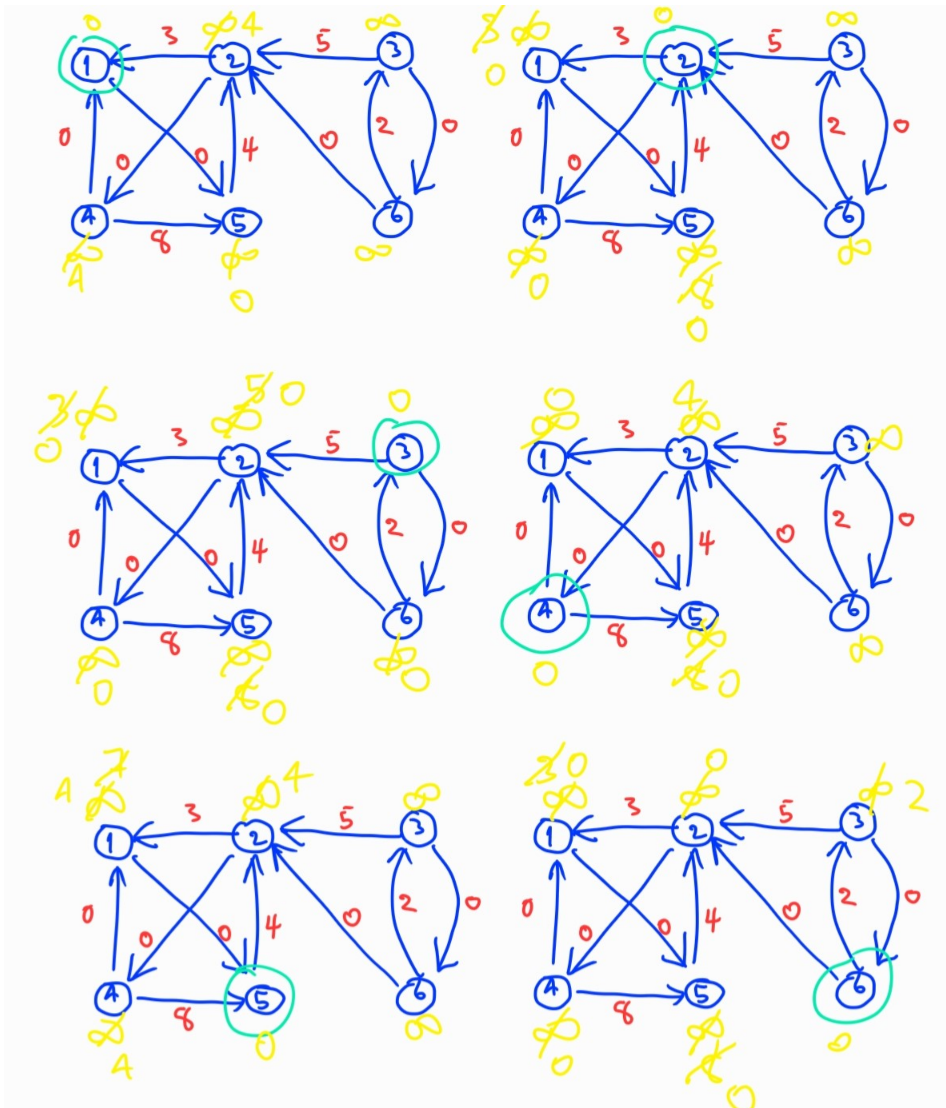

Figure 22: Apply Dijkstra's algorithm on the re-weighted graph G, starting from each vertex as a source.

The value computed by Djkstra's algorithm for each pair of vertices (u, v) is denoted by $\hat{\delta}(u, v)$. In Figure 22, $\hat{\delta}(u, v)$ for each pair (u, v) is shown in yellow. We need to convert these values back to the δ-values for the original graph G using the following equation:

$$\delta(u, v) = \hat{\delta}(u, v) + h(v) - h(u)$$

Therefore, shortest paths between all pairs of vertices can be represented by $\Delta = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$,

where each entry Δ_{ij} corresponds to $\delta(i, j)$.