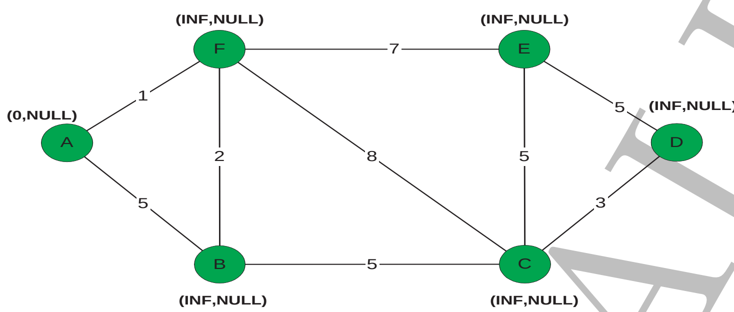
CSc 3102: Shortest Path Algorithms

Dijkstra's Single-source & Floyd's All-pairs Algorithms

Dijkstra Algorithm

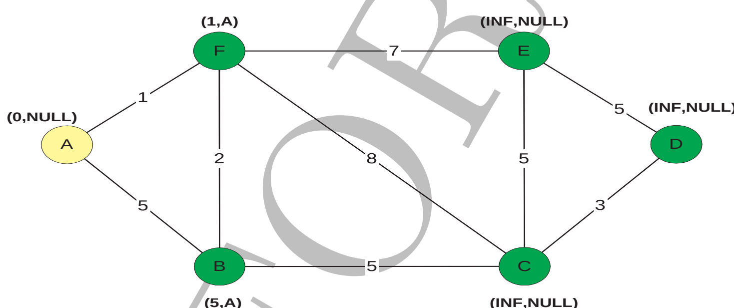
```
ALGORITHM: Dijkstra(G,src,dst)
{  Input: G - a weighted graph with n vertices
          src - a vertex of G, the source
          dst - a vertex of G, the destination
   Output: dist[]- the shortest distance from s to other vertices
           pred[]- the parent of the specified vertex}
   {Q is a min-priority-queue with -key-id comparator}
   Let dist[] ← ∞
   Let pred[] ← -1
   for v ∈ V(G) AND v ≠ src do
      Let v.key ← ∞
   endfor
   Let src.key ← 0
   Let Q ← V(G)
   while Q.empty()=false AND top ≠ dst
      Let top ← Extract-Min(Q)
      Let dist[top] ← top.key;
      for v ∈ Adj[top]
         if v in Q and top.key + weight(top,v) < v.key
            Decrease-Key(Q,v,top.key + weight(top,v))
            Let pred[v] ← top
            Let dist[v] ← v.key
      endif
   endWhile
   return dist[], pred[]
end Dijkstra
```

Analysis:
The worst-case performance of *Dijkstra* belongs to $O(n^2)$ or $O(m + n \log n)$ if a priority queue is used, where $n = |G| = |V|$ and $m = \|G\| = |E|$. It is a greedy algorithm.



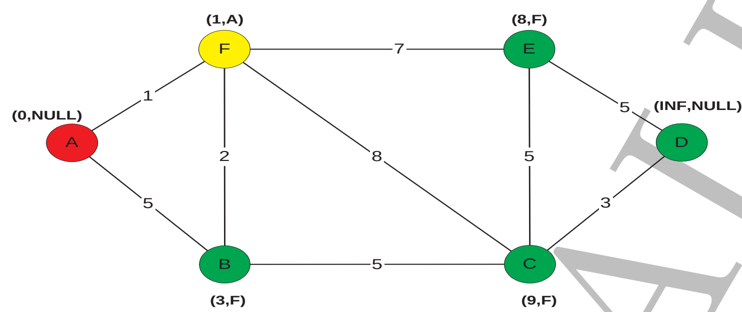
Vertex	A	B	C	D	E	F
Distance	0	∞	∞	∞	∞	∞
Predecessor	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
inTree	✗	✗	✗	✗	✗	✗

Figure 1: Original Weighted Graph and Distance and Predecessor Arrays



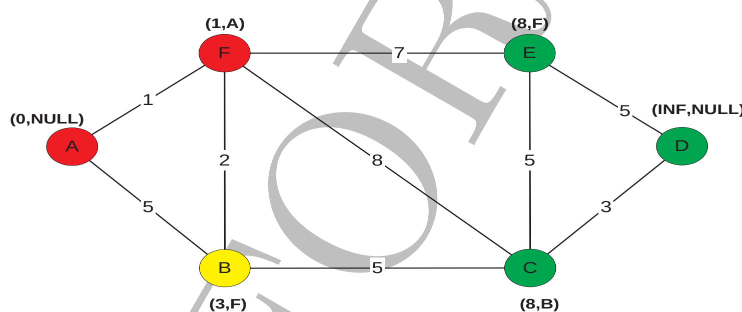
Vertex	A	B	C	D	E	F
Distance	0	5	∞	∞	∞	1
Predecessor	ϕ	A	ϕ	ϕ	ϕ	A
inTree	✓	✗	✗	✗	✗	✗

Figure 2: Stage 1 after Selecting Vertex A



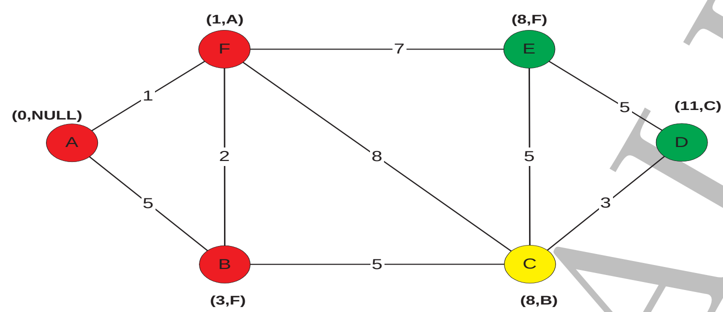
Vertex	A	B	C	D	E	F
Distance	0	3	9	∞	8	1
Predecessor	ϕ	F	F	ϕ	F	A
inTree	✓	✗	✗	✗	✗	✓

Figure 3: Stage 2 after Selecting Vertex F



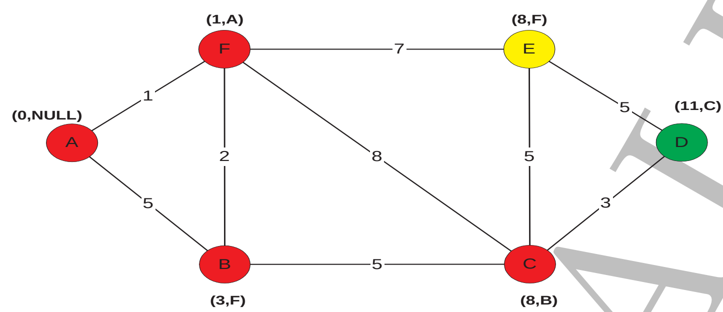
Vertex	A	B	C	D	E	F
Distance	0	3	8	∞	8	1
Predecessor	ϕ	F	B	ϕ	F	A
inTree	✓	✓	✗	✗	✗	✓

Figure 4: Stage 3 after Selecting Vertex B



Vertex	A	B	C	D	E	F
Distance	0	3	8	11	8	1
Predecessor	ϕ	F	B	C	F	A
inTree	✓	✓	✓	✗	✗	✓

Figure 5: Stage 4 after Selecting Vertex C



Vertex	A	B	C	D	E	F
Distance	0	3	8	11	8	1
Predecessor	ϕ	F	B	C	F	A
inTree	✓	✓	✓	✗	✓	✓

Figure 6: Stage 5 after Selecting Vertex E

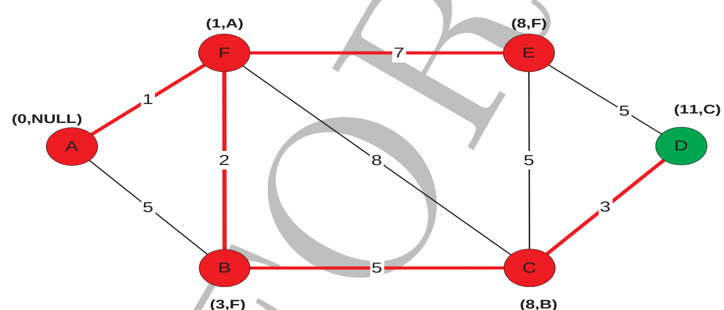
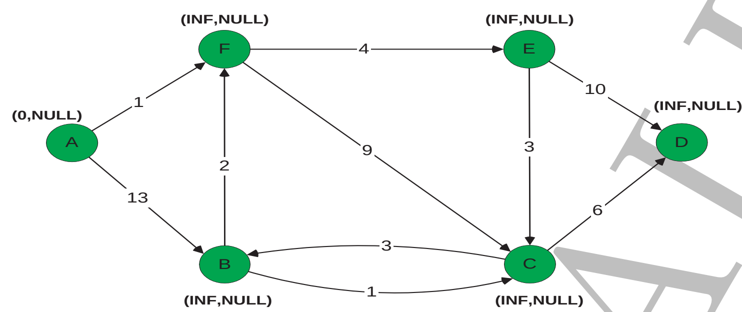


Figure 7: Single-Source Shortest Path Tree Rooted at Vertex A

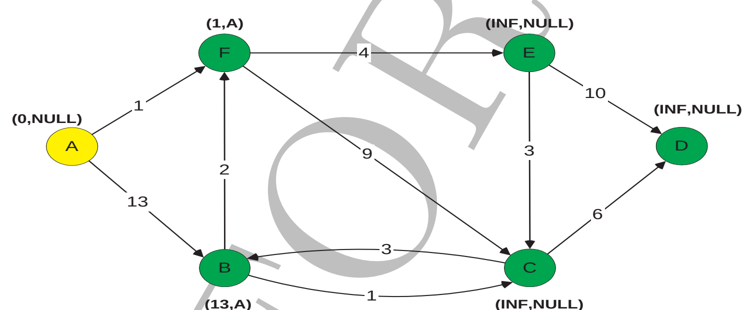
Observe that given a graph with n vertices, when $n-1$ vertices have been chosen the generation of the single-source shortest paths rooted at the starting vertex is complete since selecting the n^{th} vertex cannot lead to any update in the distance and predecessor arrays. We can then use the final arrays or graph to generate a shortest path from vertex A. For example, the shortest distance from A to D is $\delta(A, D) = 11$ and the shortest path from A to D is $\pi[A, D] = A \to F \to B \to C \to D$.

We now present another hand-trace of the action of Dijkstra's single-source shortest paths algorithm from Vertex A, this time using a digraph.



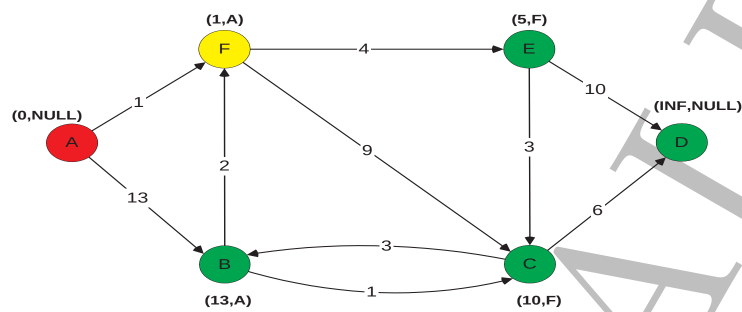
Vertex	A	B	C	D	E	F
Distance	0	∞	∞	∞	∞	∞
Predecessor	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
inTree	✗	✗	✗	✗	✗	✗

Figure 8: Original Weighted Digraph and Distance and Predecessor Arrays



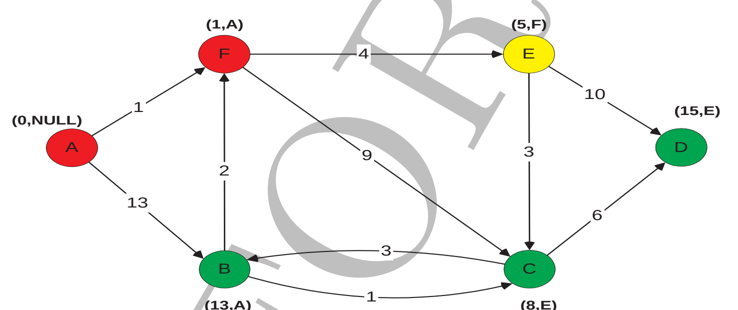
Vertex	A	B	C	D	E	F
Distance	0	13	∞	∞	∞	1
Predecessor	ϕ	A	ϕ	ϕ	ϕ	A
inTree	✓	✗	✗	✗	✗	✗

Figure 9: Stage 1 after Selecting Vertex A



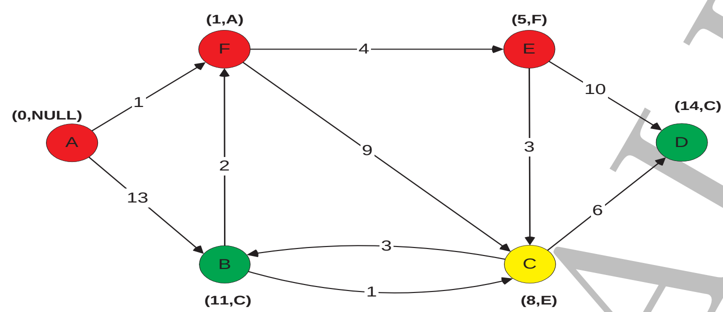
Vertex	A	B	C	D	E	F
Distance	0	13	10	∞	5	1
Predecessor	ϕ	A	F	ϕ	F	A
inTree	✓	✗	✗	✗	✗	✓

Figure 10: Stage 2 after Selecting Vertex F



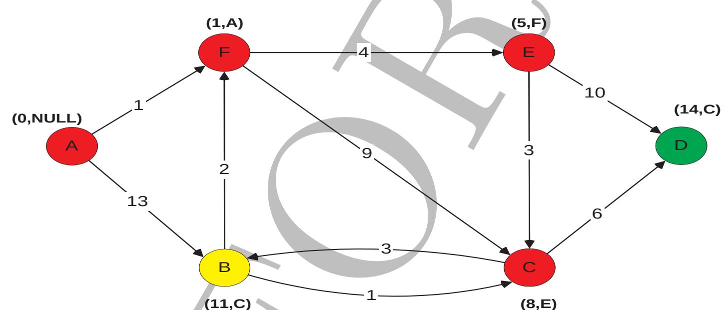
Vertex	A	B	C	D	E	F
Distance	0	13	8	15	5	1
Predecessor	ϕ	A	E	E	F	A
inTree	✓	✗	✗	✗	✓	✓

Figure 11: Stage 3 after Selecting Vertex E



Vertex	A	B	C	D	E	F
Distance	0	11	8	14	5	1
Predecessor	ϕ	C	E	C	F	A
inTree	✓	✗	✓	✗	✓	✓

Figure 12: Stage 4 after Selecting Vertex C



Vertex	A	B	C	D	E	F
Distance	0	11	8	14	5	1
Predecessor	ϕ	C	E	C	F	A
inTree	✓	✗	✓	✗	✓	✓

Figure 13: No Updates of Arrays after Selecting Vertex B

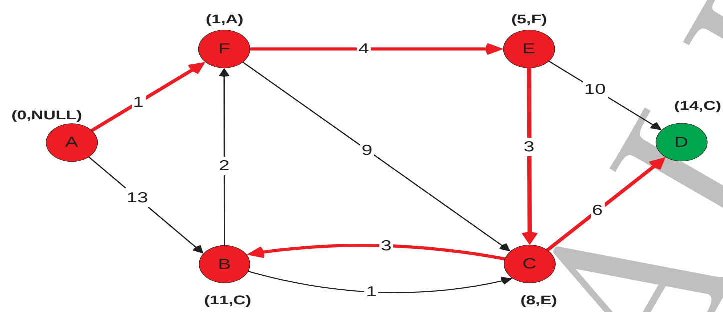


Figure 14: Single-Source Shortest Path Tree Rooted at Vertex A

Again, observe that after $n-1$ vertices have been selected in a graph with n vertices the generation of the single-source shortest paths rooted at the starting vertex is complete since selecting the n^{th} vertex leads to no updates of the distance and predecessor arrays.

As was done with the undirected graph in the previous hand-trace of the algorithm, we can use use the final arrays or digraph to generate a shortest path from vertex A. For example, the shortest distance from A to B is $\delta(A, B) = 11$ and the shortest path from A to B is $\pi[A, B] = A \rightarrow F \rightarrow E \rightarrow C \rightarrow B$.

Floyd-Warshall Algorithm

Unlike the previous algorithm, the Floyd's algorithm finds all-pairs shortest paths in a graph. It is a dynamic-programming-based algorithm.

```
ALGORITHM: Floyd(G,P,S)
  {Input:  G – a weighted digraph with n vertices.
   Output: P – n–by–n matrix implementing shortest paths.
           S – n–by–n distance matrix where S[u,v] is the
               length of a shortest path from u to v in G}
–
   for i ← 1:n do
      for j ← 1:n do
         if i = j
            P[i,j] ← j
            S[i,j] ← 0
         else if (i,j) ∈ E(G)
            P[i,j] ← j
            S[i,j] ← weight(i,j)
         else
            P[i,j] ←φ
            S[i,j] ← ∞
         endif
      endfor
   endfor
   for i ← 1:n do
      S[i,i] ← 0
      P[i,i] ← i
   endfor
   for k ← 1:n do
      for i ← 1:n do
         for j ← 1:n do
            if S[i,j] > S[i,k] + S[k,j] then
               P[i,j] ← P[i,k]
               S[i,j] ← S[i,k] + S[k,j]
            endif
         endfor
      endfor
   endfor
end Floyd
```

Analysis:
Convince yourself that Floyd has a best-case, average-case and worst-case belonging to O(n^3).

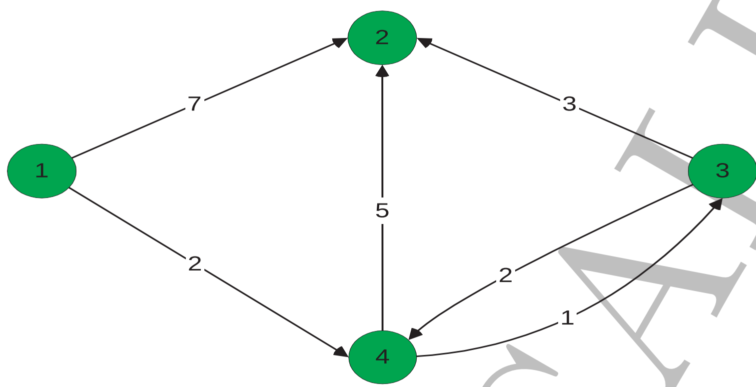


Figure 15: A Weighted Diraph

The matrices below are the distance and path (successor) matrices at various stages during the execution of the Floyd-Warshall all-pairs shortest path algorithm on the graph in Figure 15.

$$\Delta_0 = \begin{pmatrix} \text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \\ \infty & \text{[illegible]} & \infty & \infty \\ \infty & 3 & \text{[illegible]} & 2 \\ \infty & 5 & 1 & \text{[illegible]} \end{pmatrix} \Pi_0 = \begin{pmatrix} \text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \\ \text{[illegible]} & \text{[illegible]} & \phi & \phi \\ \text{[illegible]} & 2 & \text{[illegible]} & 4 \\ \text{[illegible]} & 2 & 3 & \text{[illegible]} \end{pmatrix} \quad (1)$$

$$\Delta_1 = \begin{pmatrix} \text{[illegible]} & \text{[illegible]} & \infty & 2 \\ \infty & \text{[illegible]} & \text{[illegible]} & \infty \\ \infty & \text{[illegible]} & \text{[illegible]} & 2 \\ \infty & \text{[illegible]} & 1 & \text{[illegible]} \end{pmatrix} \Pi_1 = \begin{pmatrix} \text{[illegible]} & \text{[illegible]} & \phi & 4 \\ \text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \\ \phi & \text{[illegible]} & \text{[illegible]} & 4 \\ \phi & \text{[illegible]} & 3 & \text{[illegible]} \end{pmatrix} \quad (2)$$

$$\Delta_2 = \begin{pmatrix} \text{[illegible]} & 7 & \infty & 2 \\ \infty & \text{[illegible]} & \infty & \infty \\ \infty & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \\ \infty & 5 & \text{[illegible]} & \text{[illegible]} \end{pmatrix} \Pi_2 = \begin{pmatrix} \text{[illegible]} & 2 & \text{[illegible]} & 4 \\ \phi & \text{[illegible]} & \text{[illegible]} & \phi \\ \text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \\ \phi & 2 & \text{[illegible]} & \text{[illegible]} \end{pmatrix} \quad (3)$$

$$\Delta_3 = \begin{pmatrix} \text{[illegible]} & 7 & \infty & \text{[illegible]} \\ \infty & \text{[illegible]} & \infty & \infty \\ \infty & 3 & \text{[illegible]} & \text{[illegible]} \\ \infty & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \end{pmatrix} \Pi_3 = \begin{pmatrix} \text{[illegible]} & 2 & \phi & \text{[illegible]} \\ \phi & \text{[illegible]} & \phi & \text{[illegible]} \\ \phi & 2 & \text{[illegible]} & \text{[illegible]} \\ \text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \end{pmatrix} \quad (4)$$

$$\Delta_4 = \begin{pmatrix} 0 & 6 & 3 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 3 & 0 & 2 \\ \infty & 4 & 1 & 0 \end{pmatrix} \Pi_4 = \begin{pmatrix} 1 & 4 & 4 & 4 \\ \phi & 2 & \phi & \phi \\ \phi & 2 & 3 & 4 \\ \phi & 3 & 3 & 4 \end{pmatrix} \quad (5)$$

We can use use the final Δ and Π matrices of the digraph to determine the shortest distance and generate a shortest path between any pair of vertices in the graph. For example, the shortest distance from **1** to **2** is $\Delta(1,2) = 6$ and the shortest path from **1** to **2** is $\Pi[1,2] = 1 \rightarrow \Pi[1,2] = 1 \rightarrow 4 \rightarrow \Pi[4,2] = 1 \rightarrow 4 \rightarrow 3 \rightarrow \Pi[3,2] = 1 \rightarrow 4 \rightarrow 3 \rightarrow 2$ since $\Pi[3,2] = 2$ and **2** is the destination vertex. Therefore, we get $\Pi[1,2] = 1 \rightarrow 4 \rightarrow 3 \rightarrow 2$

Problem 1.

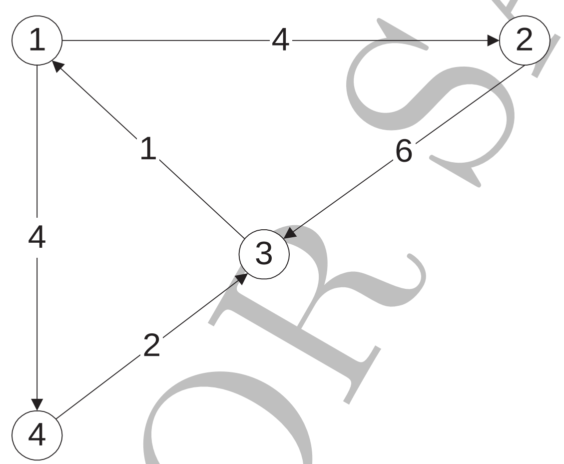


Figure 16: A Weighted Digraph

1. Trace the action of Dijkstra algorithm in finding the shortest paths from vertex 2 to every other vertex in the digraph in Figure 16 by giving the initial contents of the *dist* and *pred* arrays and their contents after each vertex is selected.

2. Using the final contents of those arrays, give the shortest path and distance from vertex 2 to 4. Explain how you obtained them from the arrays.

3. Trace the action of Dijkstra algorithm in finding the shortest paths from vertex 1 to every other vertex in the digraph in Figure 16 by giving the initial contents of the *dist* and *pred* arrays and their contents after each vertex is selected.

4. Using the final contents of those arrays, give the shortest path and distance from vertex 1 to 3. Explain how you obtained them from the arrays.

5. Draw the shortest path tree rooted at vertex **E** for the graph shown in Figure 1 and give its corresponding parent/predecessor-array representation.

6. Trace the action of Floyd-Warshall algorithm in finding the shortest paths between every pair of vertices in the digraph in Figure 16 by giving the initial contents of the distance matrix Δ and the path matrix Π and their contents at every stage of the algorithm ($k = 1, \cdots, 4$).

7. Using the final contents of the path and distance matrices, give the shortest path and distance from vertex 2 to 4. Explain how you obtained them from the matrices.

8. Again, using the final contents of those matrices, give the shortest path and distance from vertex 1 to 3. Explain how you obtained them from the matrices.

9. Explain why a depth-first-search or breadth-first-search based strategy is generally more efficient in determining whether or not there is a path between two vertices than the use of Dijkstra or Floyd-Warshall algorithm. List the strategies in non-increasing order of efficiency.

10. Draw the shortest path tree rooted at vertex 1 that is induced by the Dijkstra's algorithm on the graph in Figure 16.

11. Explain how the *Dijkstra* can be used to determine the shortest paths between each pair of vertices in a graph.

12. Suppose a min-priority-queue based implementation of *Dijkstra* is used to determine the shortest paths between pairs of vertices in a graph. How does the asymptotic upper bound of this approach compare to *Floyd* on sparse graphs and on dense graphs?

13. Is the approach in problem 1.11 always a feasible alternative to *Floyd*? If yes, why? If not, when is it infeasible?