Topological Sort Algorithm

- Given a graph G with n vertices, generate a list L of vertices arranged in topological order.

```
// set indegree for each vertex
for(i = 0; i < n; i++)
   for(j = 0; j < n; j++)
        v[j].indegree += A[i][j];
// apply topological sort
for(i = 0; i < n; i++)
{
  find an unvisited vertex v with no predecessors;
  if(v != null)        // there is one with no predecessors
  {
    insert v at back of L;
    v.wasVisited = true;
    decrement indegree of each vertex adjacent to v;
  }
}
```

Algorithm to find an unvisited vertex with no predecessors:

```
for(i = 0; i < n; i++)
   if((v[i].indegree == 0) && (v[i].wasVisited == false))
        return v[i];
return null;           // every vertex has a predecessor
```

Bridges of Konigsberg Problem

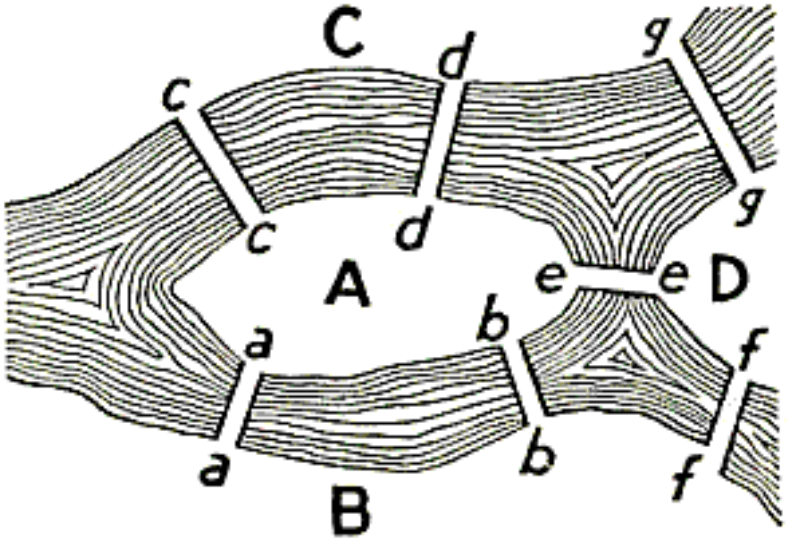
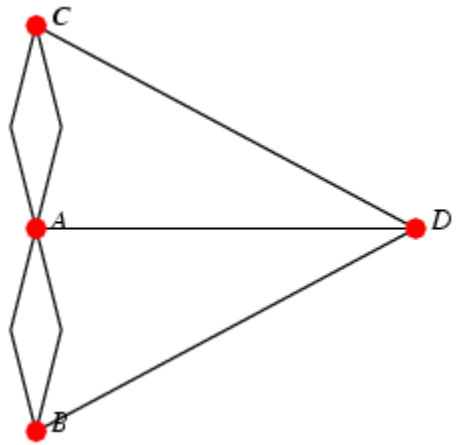


FIGURE 98. *Geographic Map: The Königsberg Bridges.*

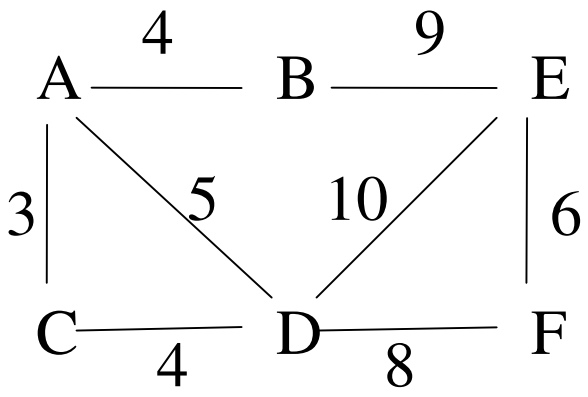


- There were 7 bridges in Konigsberg, arranged as above (pictures from `http://mathworld.wolfram.com`).

- Is it possible to travel across every bridge exactly once, and return to the starting point?

Weighted Graphs

- Adjacency matrix representation:
  - A[i][j] = w if there is an edge between vertex i and vertex j with weight w
  - A[i][j] = ∞ otherwise



	A	**B**	**C**	**D**	**E**	**F**
A	-	4	3	5	∞	∞
B	4	-	∞	∞	9	∞
C	3	∞	-	4	∞	∞
D	5	∞	4	-	10	8
E	∞	9	∞	10	-	6
F	∞	∞	∞	8	6	-

Unweighted Shortest Path Algorithm

- We wish to find the shortest path from s to all other vertices.
- Assume that for every vertex v, we have initialized v.known = false and v.dist = ∞.

```
Q.enqueue(s);
s.dist = 0;
while Q is not empty
{
  v = Q.dequeue();
  v.known = true;
  for (each vertex w adjacent to v)
    if(w.dist > v.dist+1)
    {
      w.dist = v.dist + 1;
      w.path = v;
      Q.enqueue(w);
    }
}
```

Weighted Shortest Path Algorithm (Dijkstra's Algorithm)

- We will find the weighted shortest path from s to all other vertices.
- Assume that for every vertex v, we have initialized v.known = false and v.dist = infinity.

```
s.dist = 0;
PQ.enqueue(s, 0);
while PQ is not empty
{
  v = PQ.dequeue();
  if(!v.known)
  {
    v.known = true;
    for (each vertex w adjacent to v)
      if(w.dist > v.dist + A[v][w])
      {
        w.dist = v.dist + A[v][w];
        w.path = v;
        PQ.enqueue(w, w.dist);
      }
  }
}
```