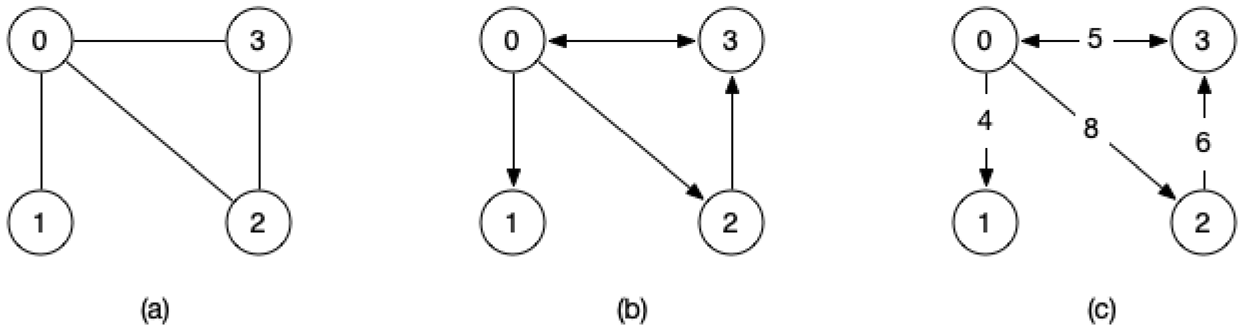
Week 07 Tutorial

DiGraphs, MST, Shortest Path

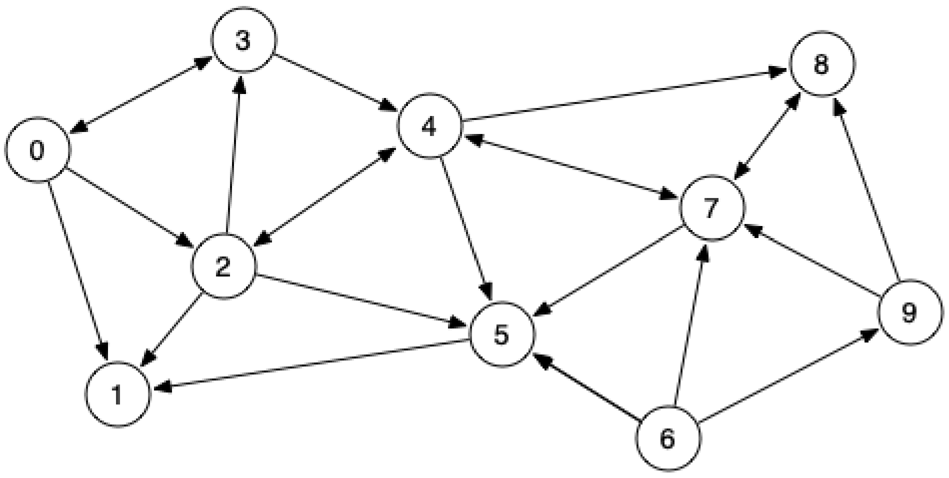
1. How is the adjacency matrix for a directed graph different to that for an undirected graph?

2. For each of the following graphs, give its adjacency matrix representaion:



Note that we use double-ended arrows (v↔w) to represent two single edges (v→w and w→v). If the edge is weighted, we assume that the weight is the same in both directions.

3. In the following graph:



    a. which vertices are reachable from vertex 0?
    b. which vertices are reachable from vertex 1?
    c. which vertices are reachable from vertex 5?
    d. which vertices are reachable from vertex 6?

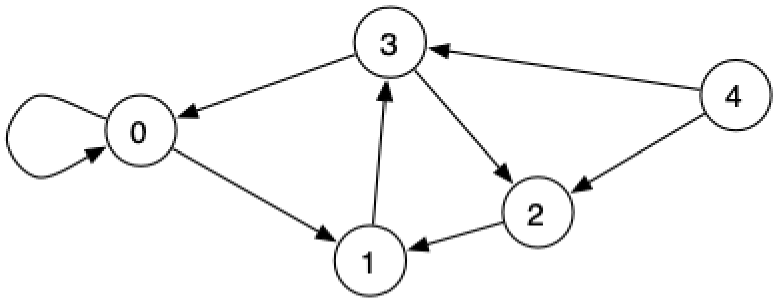
4. Facebook could be considered as a giant "social graph"

    a. what are the vertices?
    b. what are the edges?
    c. are edges directional?
    d. what does the degree of each vertex represent?
    e. what kind of graph algorithm could suggest potential friends?

5. Warshall's algorithm for computing the transitive closure (reachability matrix) of a graph can be expressed as follows:

```
void makeTC(Graph g)
{
   for (int i = 0; i < g->V; i++) {
      for (int j = 0; j < g->V; j++) {
         g->tc[i][j] = g->edge[i][j];
      }
   }
   for (int i = 0; i < g->V; i++) {
      for (int s = 0; s < g->V; s++) {
         for (int t = 0; t < g->V; t++) {
            if (g->tc[s][i] && g->tc[i][t])
               gc->tc[s][t] = 1;
         }
      }
   }
}
```

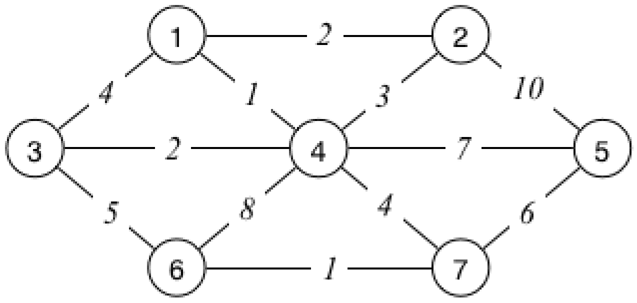
Show the adjacency matrix (`edges`) for the following graph and the transitive closure matrix (`tc[][]`) that would be produced by applying Warshall's algorithm to the following graph:



6. Kruskal's algorithm for finding a minimum spanning tree of a graph can be expressed as follows:

```
typedef Graph MSTree;
MSTree kruskalFindMST(Graph g)
{
    MSTree mst = newGraph(); // MST initially empty
    Edge eList[g->nV]; // sorted array of edges
    edges(eList, g->nE, g);
    sortEdgeList(eList, g->nE);
    for (int i = 0; mst->nE < g->nV - 1; i++) {
        Edge e = eList[i];
        insertE (mst, e);
        if (hasCycle(mst)) removeE (mst, e);
    }
    return mst;
}
```

This algorithm effectively constructs the MST by gradually joining together the connected graphs in a forest that starts with each subgraph being a single node. On each iteration, it add a new edge to the forest, and reduces the number of subgraphs by one. Show how it would construct the MST for the graph below:

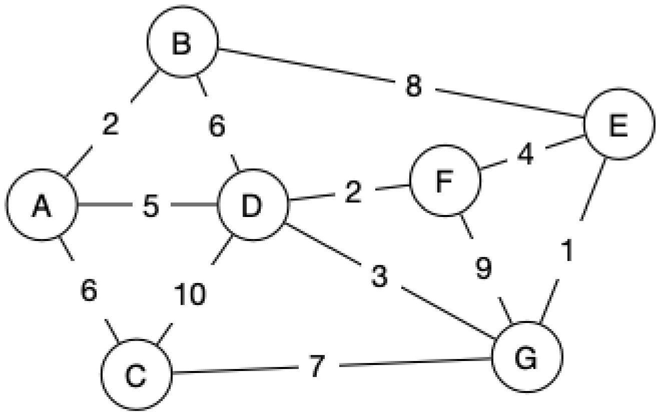


How many edges did we have to consider? For a graph $G(V, E)$, what is the least number of edges we might need to consider? What is the most number of edges we might have to consider? Add another edge to the above graph to force Kruskal's algorithm to the worst case.

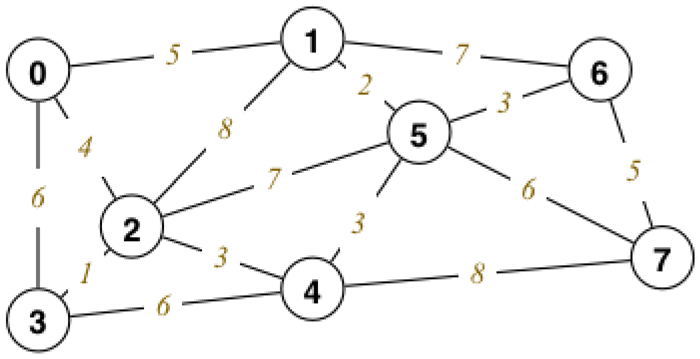
7. Prim's algorithm for finding an MST of a graph can be expressed abstractly as follows:

   1. start from any vertex *v* and empty MST
   2. choose edge not already in MST, satisfying
      - incident on a vertex *s* already in MST
      - incident on a vertex *t* not already in MST
      - with minimal weight of all such edges
   3. add chosen edge to MST
   4. repeat until MST covers all vertices

   Show how Prim's algorithm produces an MST on the following graph:



8. Trace the execution of Dijkstra's algorithm on the following graph to compute the minimum distances from source node 0 to all other vertices:



Show the values of `vSet`, `dist[]` and `pred[]` after each iteration.