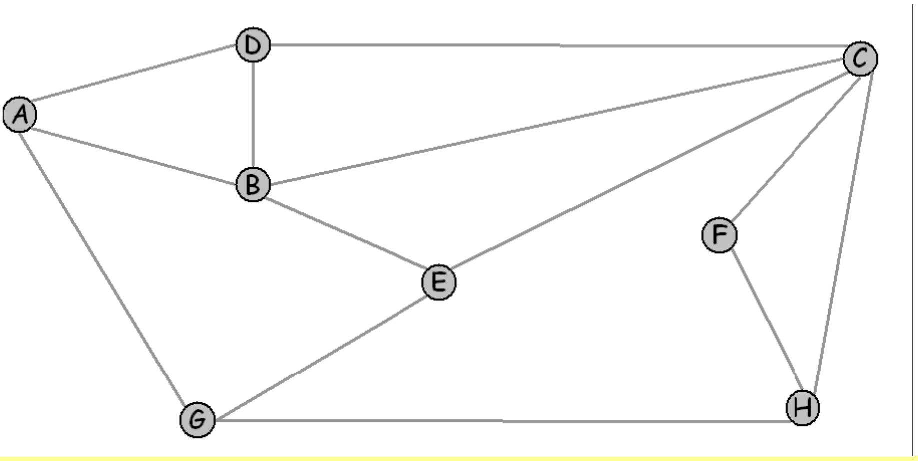
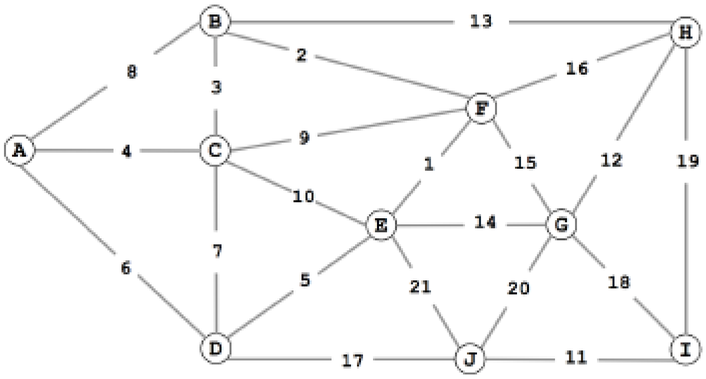
CSCE 2211 Exercises
Exercises (7): Graphs

1. Consider the following undirected network. Assume the adjacency lists are in lexicographic order, e.g., when exploring vertex A, use A-B before A-H. List the vertices in the order they are visited with DFS and with BFS.



Answer: Construct the adjacency matrix. From that, follow the algorithms.
DFS: A B C D E G H F **BFS:** A B D G C E H F

2. Consider the following weighted graph with 10 vertices and 21 edges. Note that the edge weights are distinct integers between 1 and 21.



(a) Complete the sequence of edges in the MST in the order that Kruskal's algorithm includes them.

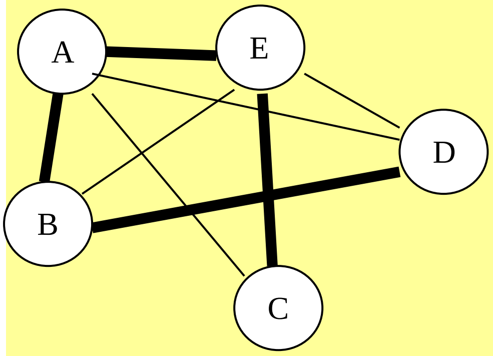
Answer :1 2 3 4 5 11 12 13 17

(b) Complete the sequence of edges in the MST in the order that Prim's algorithm includes them. Start Prim's algorithm from vertex A.

4 3 2 _____ _____ _____ _____ _____ _____ Left to you to exercise

3. Draw an undirected graph with 5 vertices (A, B, C, D, E) with all edges present except (B, C) and (C, D).
   - Represent the graph as an adjacency matrix and as an adjacency list.
   - Draw a possible Spanning Tree for such a graph.
   - Find the degree of each vertex and the degree of the graph.

Answer:



Adjacency Matrix

	A	B	C	D	E
A	0	1	1	1	1
B	1	0	0	1	1
C	1	0	0	0	1
D	1	1	0	0	1
E	1	1	1	1	0

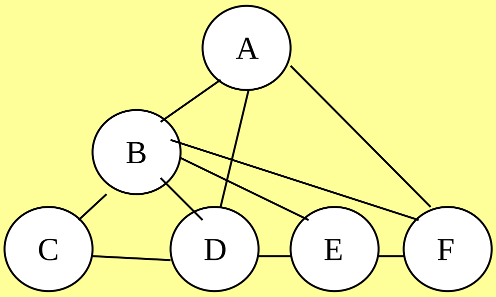
Adjacency List

A-B-C-D-E
B-A-D-E
C-A-E
D-A-B-E
E-A-B-C-D

Heavy edges represent a possible spanning tree.
Degrees of vertices: A:4 B:3 C:2 D:3 E: 4, Degree of graph is 4

4. Consider the undirected graph G (V, E) with V = (A . . F) and
   E = (AB , AD , AF , BC, BD , BE , BF , CD , DE , EF).
   - Draw the graph.
   - Represent the graph as an adjacency matrix
   - Trace the Depth-First Search (DFS) algorithm on the above graph starting from vertex (A).
   - What is the exact number of times the function Visit is invoked?

Answer:



Adjacency Matrix

	A	B	C	D	E	F
A	0	1	0	1	0	1
B	1	0	1	1	1	1
C	0	1	0	1	0	0
D	1	1	1	0	1	0
E	0	1	0	1	0	1
F	1	1	0	0	1	0

DFS Visit Order
A B C D E F

Visit will be invoked 6 times.

5. Rewrite the following DFS algorithm so that it returns the number of connected components in a graph:

```
// Assume order = 0 initially
void DFS()
{
      int k;  int unseen = -2;
      for (k = 1; k <= v; k++) val[k] = unseen;
      for (k = 1; k <= v; k++)
            if (val[k] == unseen) visit(k);
}
```

```
void Visit (int k)
{
      int t;
      val[k] = ++order;
      for (t = 1; t <= v; t++)
            if (a[k][t] != 0)
               if (val[t] == unseen) Visit (t);
```

Answer:

Re-write the main function DFS( ) to become:

```
int DFS()
{
      int k;  int unseen = -2; int Count = 0;
      for (k = 1; k <= v; k++) val[k] = unseen;
      for (k = 1; k <= v; k++)
            if (val[k] == unseen) {visit(k); Count++;}
       return Count;
}
```

6. Assume a network of (n) computers, such that there is a path from each machine to every other machine. An articulation node is a machine that if crashes, will cause the network to be disconnected into two or more isolated networks. If more than one such node exists, the one that leads to the largest number of isolated connected components is the weakest link in the network. We would like to identify such "weakest link" machine.

The number of isolated connected components (C_i) when machine (i) crashes (i = 1,2,..,n) can be stored in an array so that the weakest link is Max(C_1 , C_2 , .., C_n).

Modify the following DFS algorithm to do this job.

```
DFS( )
{         set all nodes as unvisited;
          for each node (k)  if( k not yet visited) Visit(k);  }
```

```
Visit (u)
{         mark (u) as visited;
          output u;
          for each edge (u,v) if( v not yet visited) Visit(v);  }
```

The input: The adjacency matrix representing the network.

The output: The array C.

Solution:

The DFS algorithm can be modified to return the number of connected components in a graph represented as an adjacency matrix A

```
    int DFS2(A )
    {  nc = 0;
    set all nodes as unvisited;
    for each node (k)  if( k not yet visited) {Visit(k); nc++  };
                        return nc; }
```

If the graph is connected, nc = 1. When we disconnect a node that is not an articulation point, we get nc = 2 (1 for the disconnected node and 1 for the remaining connected nodes). If we get nc > 2, the node is an articulation point.

We can disconnect a node form the rest of the graph by zeroing its row and its column in the adjacency matrix.

To find the weakest link using the modified DFS:

```
for each node ( i )  in the graph
    • Copy matrix A to a matrix B
    • Zero row (i) and column (i) in B
    • $C_i$ = DFS2(B);
 cc = 2;   Weakest = 0;
 for (i = 1 to n)   if ( $C_i$ > cc) { cc = $C_i$ ;  Weakest = i};
```

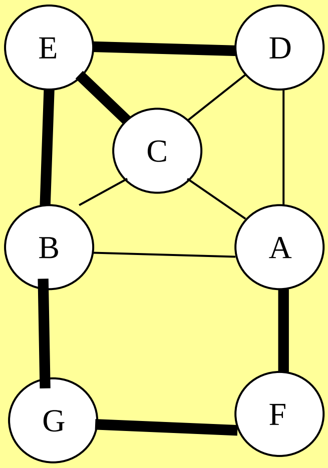
7. A weighted graph contains 7 vertices (A .. G) with the following edges :
   AB = 10 , AC = 38 , AD = 50 , AF = 5 , BC = 48 , BE = 14 , BG = 2 ,
   CD = 26 , CE = 20 , DE = 15 , FG = 2
   - Draw the graph
   - Trace Kruskal's algorithm to find a Minimum Spanning Tree (MST) for the graph.
   - Trace Dijkstra's algorithm to find the shortest paths from vertex (A) to all other vertices.

Answer:

Using Kruskal's Algorithm: Y: Accept edge, N: Reject edge

BG:Y, FG:Y, AF:Y, AB:N, BE:Y, DE:Y, CE:Y

Heavy edges represent the MST with a minimum cost of 58



Tracing Dijkstra's algorithm to find the shortest paths from vertex (A):

Initial Table for paths Final Table

	A	B	C	D	E	F	G			A	B	C	D	E	F	G
Dist	0	10	38	50	inf	5	inf		Dist	0	~~10~~ 9	38	~~50~~ 38	~~inf~~ 23	5	~~inf~~ 7
Proc	yes	No	No	No	No	No	No		Proc	yes	yes	yes	yes	yes	yes	yes
Via	A	A	A	A		A			Via	A	~~A~~ G	A	~~A~~ E	B	A	F

The shortest paths are:

A F G B	9	A C	38	A F G B E D	38
A F G B E	23	A F	5	A F G	7

8. A weighted graph contains 7 vertices (A .. G) with the following edges :
AF = 10 , CD = 12 , BG = 14 , BC = 16 , DG = 18 , DE = 16 , EG = 8 ,EF = 25 , AB = 2
(a) Draw the graph
(b) The following algorithm finds a Minimum Spanning Tree for a graph:

```
Given a graph with V > = 1 vertices, T = MST , TV = set of vertices
1. set T = empty , TV = first vertex , done = false
2. while T contains less than V-1 edges and not done do
        begin
          choose an edge (u,v) with (u) in TV and (v) not in TV such that
            (u,v) is of least cost ;
          if there is no such edge then done = true
                else add (v) to TV and (u,v) to T;
    end {while}
```

Trace this algorithm on the above graph. What is the difference between this algorithm and Kruskal's algorithm for finding the MST?

Solution:

This is Prim's algorithm for the MST. The partial tree is always one tree, in contrast to Kruskal's algorithm which builds up a forest of trees then joins them into one tree.

9. Consider a complete weighted graph with 512 vertices.
(a) Which of the following will closely represent the worst cost (in number of operations) required to build the graph's Minimum Spanning Tree (MST) using Kruskal's algorithm:

(1) 2^{18} (2) 2^{20} (3) 2^{22}

(b) A faster algorithm to do the above task is to divide the adjacency matrix into equal submatrices and assign each to a processor to compute the MST for the corresponding subgraph in parallel with the other processors. After that, we combine the sub-trees into one MST. If we have 256 such processors, and the combine cost is negligible, which of the following will closely represent the worst cost (in number of operations) required to build the total graph's MST:

(1) 2^{13} (2) 2^{15} (3) 2^{17}

Important: in both (a) and (b) give the reasons for your choice.

Solution:

a) The complexity of Kruskal's algorithm is O(E log V), where V is the number of vertices and E is the number of edges. The complete graph has $V(V-1)/2 \approx V^2/2$ edges. Now $V = 2^9$, $E = 2^{17}$ Hence, the number of operations is of the order of 2^{20}.

b) The adjacency matrix will be of size $V^2 = 2^{18}$. If we divide it among $M = 256 = 2^8$ processors, then each processor will have a subgraph with $V_s= 2^5$ vertices and edges $E_s = 2^{10}$, so that the number of operations done in parallel is 5×2^{10}. The closest cost is therefore 2^{13} operations per processor.

10. The highway distances and the weather conditions over the highways between 7 cities named (A .. G) are given as follows:

AB = 10 (clear) , AC = 10 (rainy) , AD = 50 (clear) , AF = 5 (clear) ,
BC = 2 (rainy) , BE = 14 (clear), BG = 2 (clear), BF = 3 (rainy),
CD = 26 (clear), DE = 15 (clear), DF = 6 (storm), DG = 4 (storm), FG = 2 (clear)

Trace Dijkstra's algorithm to determine the shortest least risk paths from city (A) to all other cities. Consider a risk factor (by which we multiply the actual distance) to be 1 for clear weather, 3 for rainy weather and ∞ for a storm.

Solution:

We setup the initial paths table after multiplying the weights by the risk factors:

	A	B	C	D	E	F	G
Dist	0	10	30	50	inf	5	inf
Processed	yes	No	No	No	No	No	No
Via	A	A	A	A		A	

The final table will be:

	A	B	C	D	E	F	G
Dist	0	~~10~~ 9	~~30~~ 15	~~50~~ ~~41~~ 38	~~inf~~ 23	5	~~inf~~ 7
Processed	yes	yes	yes	yes	yes	yes	yes
Via	A	~~A~~ G	~~A~~ B	~~A~~ ~~C~~ E	B	A	F

The final shortest paths will be:

A →F →G →B	clear all the way
A →F →G →B →C	clear all the way except B-C (rainy)
A →F →G →B →E →D	clear all the way
A →F →G →B →E	clear all the way
A →F	clear all the way
A →F →G	clear all the way

11. Consider a graph (G) with the following edges: (A,B) , (B,C) , (B,E) , (C,D) , (A,F) , (F,G) , (F,H). Let L = dynamic list of edges and (s , u , v) to be vertices in G. The following algorithm traverses the graph:

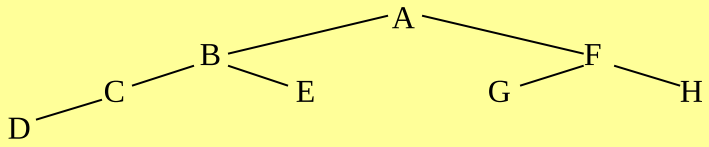
```
Traverse1 (G)
{
    list L = empty;
    set all vertices as unvisited;
    choose starting vertex (s);
    search (s);
    while ( L nonempty)
    {
        remove the edge (u,v) from the start of L;  ⇐
        if ( v not yet visited) search (v);
    }
}
```

```
search (u)
{
    mark (u) as visited;
    output u;
    for each edge (u,v) add edge to end of L;
}
```

- Trace the above algorithm for the given graph to show the order of visiting the nodes starting from node (A). What kind of traversal is that?
- Consider the action pointed to by the arrow in the function **Traverse1 (G)**. If we change **"start"** to **"end"**, what would be the order of node visits? What kind of traversal is that?

Solution:

The graph is



The algorithm gives the following traversal order: A B F C E G H

This is clearly BFS

If we change **"start"** to **"end"**, the order will be: A F H G B E C D

This is clearly DFS

12. Describe a situation when you would need to use breadth-first search instead of depth-first search (choose out of the following):

    i. Find a Euler tour
    ii. Find a directed cycle
    iii. Find a path between two vertices in an undirected graph
    iv. Find a path between two vertices with the fewest number of edges

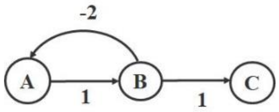
Solution:

The most probable answer is (iii)

13. A weighted graph contains 7 vertices (A .. G) with the following edges :
AF = 10 , CD = 12 , BG = 14 , BC = 16 , DG = 18 , DE = 16 , EG = 8 ,EF = 25 , AB = 2
Trace both *Kruskal's* and *Prim's* algorithm to find a *MST* for the graph.

14. The graph shown has a negative cycle of length -1.

- Construct its edge cost matrix and show that the update rule for Floyd's algorithm does not hold for this graph.
- Show that $A^2(1,3) = - \infty$



15. Solve the all-pairs shortest-path problem for the digraph with the following weight matrix:

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

16. Consider the following directed graph.

- Construct the cost matrix
- Write down the update rule to obtain the all-pairs shortest paths using Floyd's Algorithm.
- Show the steps to obtain the final all pairs shortest paths matrix.