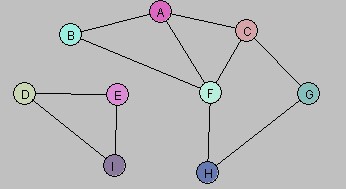
**Lab 11 Graph Theory**

Q1.

Answer questions about the G = (V,E) displayed below.



1. Let U = {A, B}. Draw G[U].

A group of circles with letters and a line

AI-generated content may be incorrect.

1. Let W = {A, C, G, F}. Draw G[W].

A diagram of a network

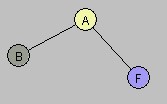
AI-generated content may be incorrect.

1. Let Y = {A, B, D, E}. Draw G[Y].

A diagram of a diagram

AI-generated content may be incorrect.

1. Consider the following subgraph H of G:



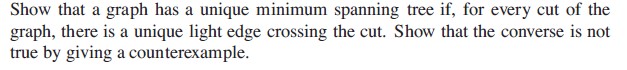
Is there a subset X of the vertex set V so that H = G[X]? Explain.

Ans:

No, there is no subset X of the vertex set V such that H = G[X], because H is missing the edge between B and F, which exists in G.

Therefore, H is not an induced subgraph of G.

Q2.



**Ans:**

A graph has a unique minimum spanning tree (MST) if, for every cut, the lightest edge crossing the cut is unique.  
This is because such edges must be included in any MST (cut property), and if they're unique, there's no alternative MST that could avoid them — so the MST is forced to include the same edges, making it unique.

However, the converse is not true.

Counterexample:

Take this graph:

* A—B (weight 1)
* B—C (weight 1)
* A—C (weight 2)

The unique MST is A—B and B—C (total weight = 2).

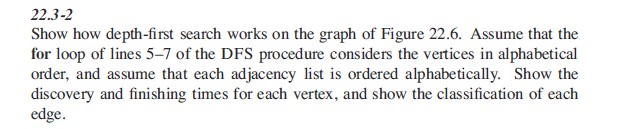
But the cut between {A, C} and {B} has two equal minimum edges (A—B and B—C, both weight 1), so the lightest edge across this cut is not unique.

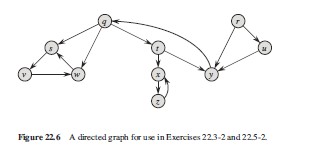
So, the graph has a unique MST, but not all cuts have a unique light edge — proving the converse is false.

Q1. A.

Additional Lab 11 Question:

Study the discovery and finishing time and then solve the following problem.





(Corman)

**Step-by-Step DFS Walkthrough**

Starting from q (alphabetically first), DFS proceeds as follows:

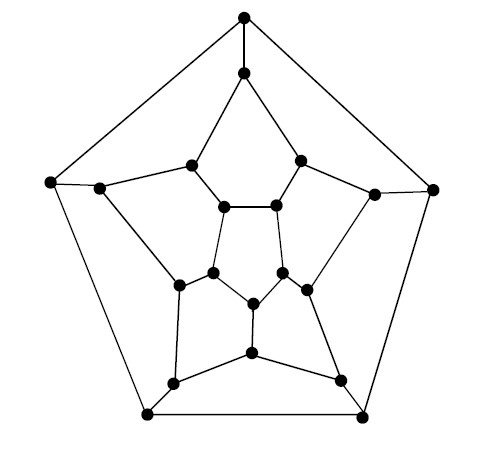
| **Vertex** | **Discovery** | **Finish** |
| --- | --- | --- |
| q | 1 | 16 |
| s | 2 | 7 |
| v | 3 | 4 |
| w | 5 | 6 |
| i | 8 | 15 |
| x | 9 | 13 |
| z | 10 | 12 |
| y | 14 | 15 |
| t | 17 | 20 |
| u | 18 | 19 |
| r | 21 | 24 |

**Edge Classifications**

| **Edge** | **Type** | **Explanation** |
| --- | --- | --- |
| q → s | Tree | Discovered s from q |
| s → v | Tree | Discovered v from s |
| s → w | Tree | Discovered w from s |
| w → q | Back | Points back to ancestor q |
| q → i | Tree | Discovered i from q |
| i → x | Tree | Discovered x from i |
| x → z | Tree | Discovered z from x |
| z → x | Back | Cycle back to x |
| i → y | Tree | Discovered y from i |
| q → t | Tree | Discovered t from q |
| t → u | Tree | Discovered u from t |
| r → y | Cross | y was already fully explored when r visited it |

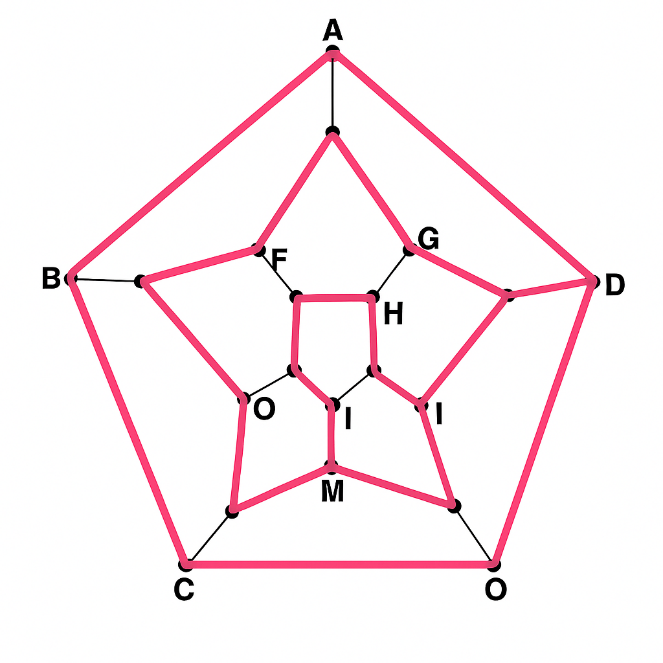
Q3.

The following graph has a Hamiltonian cycle. Find it.



We can trace the outermost pentagon and "zigzag" inward and back out, hitting every vertex once. One such valid Hamiltonian cycle (described in clockwise direction) is:

1. Start at the top vertex
2. Move down-right into the next layer
3. Follow the path in a spiral-like fashion, always choosing edges that move you around the center while pulling into inner rings and then back out
4. End up back at the starting vertex after hitting all vertices



Q4.

Consider the problem of computing a *maximum* spanning tree, namely the spanning tree that maximizes the sum of edge costs. Do Prim and Kruskal’s algorithm work for this problem (assuming of course that we choose the crossing edge with maximum cost)?

Ans:

Yes, both Prim’s and Kruskal’s algorithms still work for finding a maximum spanning tree — we just need to flip how we choose edges.

Normally, these algorithms are used to find a minimum spanning tree by picking the smallest edge at each step. But if we simply pick the largest edge instead, everything still works the same way.

* Kruskal’s Algorithm:  
  Sort the edges from largest to smallest. Then, add the biggest edge that doesn't form a cycle. Keep going until all vertices are connected. This gives us a maximum spanning tree.
* Prim’s Algorithm:  
  Start at any node, and always add the heaviest edge that connects to a new vertex. Keep growing the tree until all nodes are included.

So overall, both algorithms work just fine for maximum spanning trees, as long as we always choose the maximum-cost edge instead of the minimum.