1. Draw the StateGraph

2. a. Is there a path from Oregon to any other state in the graph?

Edges involving Oregon:

- Only one edge leads *to* Oregon: (Alaska → Oregon)
- **No edge leaves Oregon**, so:

 Answer: No, there is **no path from Oregon** to any other state.

b. Is there a path from Hawaii to every other state in the graph?

Let's examine paths starting from **Hawaii**:

Hawaii → Alaska

Hawaii → Texas

Hawaii → California

Hawaii → New York

Hawaii → Texas → Vermont

Hawaii → Texas → Vermont → Alaska → Oregon

Answer: Yes, there is a **path from Hawaii to every other state**.

c. From which state(s) in the graph is there a path to Hawaii?

We look for all directed paths ending at **Hawaii**:

- **Texas → Hawaii** (direct edge)
    - Any state that can reach Texas can also reach Hawaii.

So now check who can reach Texas:

- Hawaii → Texas → Hawaii (cycle)
- No other state leads to Texas directly.

Answer: Only **Texas** has a path to Hawaii.

3. a. Show the adjacency matrix that would describe the edges in the graph. Store the vertices in alphabetical order

Here is the **adjacency matrix** for the StateGraph, with vertices ordered alphabetically.
Each cell (i, j) contains a 1 if there is a directed edge from state i to state j, and 0 otherwise.

	Alaska	California	Hawaii	NewYork	Oregon	Texas	Vermont
Alaska	0	0	0	0	1	0	0
California	0	0	0	0	0	0	0
Hawaii	1	1	0	1	0	1	0
NewYork	0	0	0	0	0	0	0
Oregon	0	0	0	0	0	0	0
Texas	0	0	1	0	0	0	1
Vermont	1	1	0	0	0	0	0

3b. Show the adjacency lists
that would describe the edges in the graph
Here is the adjacency list for the StateGraph. Each row shows the neighboring (i.e., directly reachable) states from the corresponding state via outgoing edges.

	Neighbor 1	Neighbor 2	Neighbor 3	Neighbor 4
Alaska	Oregon			
California				
Hawaii	Alaska	Texas	California	NewYork
NewYork				
Oregon				
Texas	Hawaii	Vermont		
Vermont	California	Alaska		

4a. Which of the following lists the graph nodes in depth first order beginning with E?

A) E, G, F, C, D, B, A
B) G, A, E, C, B, F, D
C) E, G, A, D, F, C, B
D) E, C, F, B, A, D, G

None of the options are correct.
Correct BFS order from F is:
F, B, C, A, E, D, G

4b. Which of the following lists the graph nodes in breadth first order beginning at F?

A) F, C, D, A, B, E, G
B) F, D, C, A, B, C, G
C) F, C, D, B, G, A, E
D) a, b, and c are all breadth first traversals

Final BFS Order from F:
F, C, D, A, B, G, E

None of the above (not listed)

5. Find the shortest distance from Atlanta to every other city
Here are the shortest distances from **Atlanta** to each city using Dijkstra's algorithm

City	Shortest Distance from Atlanta
Atlanta	0
Austin	2100
Chicago	2400
Dallas	1900
Denver	1400
Houston	800
Washington	600

6. Find the minimal spanning tree using Prim's algorithm. Use
0 as the source vertex . Show the steps.
Let's walk through Prim's Algorithm step-by-step starting from vertex 0, using the weighted undirected graph in the image.

Step 0: Initialization
Start at vertex 0
Add edges connected to 0:
(0, 1, 7) and (0, 2, 3)
Choose the edge with the smallest weight: (0, 2, 3)

Step 1: Include Vertex 2
Tree: {0, 2}
Available edges:
(0, 1, 7), (2, 4, 8), (2, 5, 1)
Pick minimum edge: (2, 5, 1)

Step 2: Include Vertex 5
Tree: {0, 2, 5}
Available edges:
(0, 1, 7), (2, 4, 8), (5, 1, 2), (5, 4, 3)
Pick minimum edge: (5, 1, 2)

Step 3: Include Vertex 1
Tree: {0, 2, 5, 1}

Available edges:
(2, 4, 8), (5, 4, 3), (1, 3, 5)
Pick minimum edge: (5, 4, 3)

Step 4: Include Vertex 4
Tree: {0, 2, 5, 1, 4}
Available edges:
(2, 4, 8), (1, 3, 5), (4, 3, 6)
Pick minimum edge: (1, 3, 5)

Step 5: Include Vertex 3
All vertices included: {0, 1, 2, 3, 4, 5}

✅ Minimum Spanning Tree (MST) Edges:
(0, 2, 3)
(2, 5, 1)
(5, 1, 2)
(5, 4, 3)
(1, 3, 5)

✅ Total Weight = 3 + 1 + 2 + 3 + 5 = 14

From	To	Weight
0	2	3
2	5	1
5	1	2
5	4	3
1	3	5
	Total	14

7. Find the minimal spanning tree using Kruskal's algorithm.
Show the weights in order and the steps.

Let's apply Kruskal's Algorithm step-by-step to find the Minimum Spanning Tree (MST) of the graph shown.

Step 1: List all edges with weights
Edges are undirected and weights:
Edge, Weight
(2, 5), 1
(1, 5), 2
(4, 5), 3
(0, 2), 3
(1, 3), 5
(3, 4), 6
(0, 1), 7
(2, 4), 8

Step 2: Sort edges by ascending weight
Sorted edge list:
(2, 5) → 1
(1, 5) → 2
(4, 5) → 3
(0, 2) → 3
(1, 3) → 5
(3, 4) → 6
(0, 1) → 7
(2, 4) → 8

Step 3: Build the MST using Kruskal's
Use Union-Find to avoid cycles:
(2, 5) → Add ✅
(1, 5) → Add ✅
(4, 5) → Add ✅
(0, 2) → Add ✅
(1, 3) → Add ✅
Now we have 6 vertices connected with 5 edges → MST complete

✅ MST Edges (Kruskal's Algorithm):
(2, 5) → 1
(1, 5) → 2
(4, 5) → 3
(0, 2) → 3

(1, 3) → 5

✅ Total MST Weight = 1 + 2 + 3 + 3 + 5 = 14

From	To	Weight
2	5	1
1	5	2
4	5	3
0	2	3
1	3	5
	Total	14

8. Find the minimal spanning tree using the algorithm you prefer. Use Minneapolis/St. Paul as the source vertex
We'll find the Minimum Spanning Tree (MST) starting from Minneapolis/St. Paul, using Prim's Algorithm (since a source is specified).

Step 1: List All Edges
Edge, Weight
Des Moines – Minneapolis, 235
Des Moines – St. Louis, 320
Minneapolis – Madison, 270
Madison – Milwaukee, 80
Madison – Chicago, 150
Milwaukee – Chicago, 95
Chicago – Detroit, 280
Chicago – St. Louis, 270

Step 2: Start Prim's Algorithm at Minneapolis/St. Paul
Step-by-step inclusion:
Start at Minneapolis
Possible edges: (Minneapolis – Madison) 270, (Minneapolis – Des Moines) 235
Pick (Minneapolis – Des Moines) → 235
Add Des Moines

Edges: (Des Moines – St. Louis) 320, (Minneapolis – Madison) 270
Pick (Minneapolis – Madison) → 270
Add Madison
Edges: (Madison – Milwaukee) 80, (Madison – Chicago) 150
Pick (Madison – Milwaukee) → 80
Add Milwaukee
Edge: (Milwaukee – Chicago) 95, (Madison – Chicago) 150
Pick (Milwaukee – Chicago) → 95
Add Chicago
Edges: (Chicago – St. Louis) 270, (Chicago – Detroit) 280
Pick (Chicago – St. Louis) → 270
Add St. Louis
Only edge left: (Chicago – Detroit) 280
Pick (Chicago – Detroit) → 280

✅ MST Edges:
Minneapolis – Des Moines → 235
Minneapolis – Madison → 270
Madison – Milwaukee → 80
Milwaukee – Chicago → 95
Chicago – St. Louis → 270
Chicago – Detroit → 280

✅ Total Weight = 235 + 270 + 80 + 95 + 270 + 280 = 1230

From	To	Weight
Minneapolis	Des Moines	235
Minneapolis	Madison	270
Madison	Milwaukee	80
Milwaukee	Chicago	95
Chicago	St. Louis	270
Chicago	Detroit	280
	Total	1230

9. List the nodes of the graph in a breadth first topological ordering. Show the steps using arrays predCount, topologicalOrder and a queue

We will perform a Breadth-First Topological Sort (Kahn's Algorithm) on the directed graph.

Step 1: List edges to compute predCount (in-degree of each node)
From the image, we list the directed edges:
(0 → 1)
(1 → 2)
(1 → 4)
(1 → 6)
(2 → 4)
(3 → 4)
(3 → 7)
(4 → 7)
(5 → 1)
(5 → 6)
(6 → 3)
(6 → 8)
(7 → 9)
(8 → 9)

Step 2: Compute predCount[]
Initialize predCount[10] = 0 for all nodes:
predCount = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Now count how many edges point *to* each node:
1 ← 0, 5 → predCount[1] = 2
2 ← 1 → predCount[2] = 1
3 ← 6 → predCount[3] = 1
4 ← 1, 2, 3 → predCount[4] = 3
5 ← none → predCount[5] = 0
6 ← 1, 5 → predCount[6] = 2
7 ← 3, 4 → predCount[7] = 2
8 ← 6 → predCount[8] = 1
9 ← 7, 8 → predCount[9] = 2
0 ← none → predCount[0] = 0
Final predCount[]:
[0, 2, 1, 1, 3, 0, 2, 2, 1, 2]

Step 3: Initialize queue with nodes having predCount = 0
Queue = [0, 5]

Step 4: Build topologicalOrder[]
Follow these steps:

Pop 0 → add to topologicalOrder = [0]
Decrement predCount[1] → 1
Pop 5 → add to topologicalOrder = [0, 5]
Decrement predCount[1] → 0 → enqueue 1
Decrement predCount[6] → 1
Pop 1 → [0, 5, 1]
Decrement predCount[2] → 0 → enqueue 2
Decrement predCount[4] → 2
Decrement predCount[6] → 0 → enqueue 6
Pop 2 → [0, 5, 1, 2]
Decrement predCount[4] → 1
Pop 6 → [0, 5, 1, 2, 6]
Decrement predCount[3] → 0 → enqueue 3
Decrement predCount[8] → 0 → enqueue 8
Pop 3 → [0, 5, 1, 2, 6, 3]
Decrement predCount[4] → 0 → enqueue 4
Decrement predCount[7] → 1
Pop 8 → [0, 5, 1, 2, 6, 3, 8]
Decrement predCount[9] → 1
Pop 4 → [0, 5, 1, 2, 6, 3, 8, 4]
Decrement predCount[7] → 0 → enqueue 7
Pop 7 → [0, 5, 1, 2, 6, 3, 8, 4, 7]
Decrement predCount[9] → 0 → enqueue 9
Pop 9 → [0, 5, 1, 2, 6, 3, 8, 4, 7, 9]

[0, 5, 1, 2, 6, 3, 8, 4, 7, 9]

10. List the nodes of the graph in a breadth first topological ordering.
We'll determine a Breadth-First Topological Order for the course dependency graph using Kahn’s Algorithm.

Step 1: List all edges (dependencies)
Each edge goes from a prerequisite to a dependent course.
Start → Programming 1
Programming 1 → Programming 2
Programming 2 → Algorithms
Programming 2 → Computer Organization
Discrete Math → Algorithms
Algorithms → Theory of Computation
Computer Organization → Operating Systems
Computer Organization → High-Level Languages
High-Level Languages → Senior Seminar

Operating Systems → Senior Seminar
Compilers → Senior Seminar

Step 2: Compute predCount (in-degree)

Course	In-Degree
Start	0
Programming 1	1 (Start)
Programming 2	1
Algorithms	2 (Programming 2, Discrete Math)
Discrete Math	0
Computer Organization	1
Theory of Computation	1
Operating Systems	1
High-Level Languages	1
Senior Seminar	3
Compilers	0
End	1 (Senior Seminar)

Step 3: Initialize queue with 0 in-degree nodes
Queue = [Start, Discrete Math, Compilers]

Step 4: Process queue in BFS Topological Order
Proceed step-by-step:
Start → topOrder = [Start]
Programming 1: in-degree 0 → enqueue
Discrete Math → topOrder = [Start, Discrete Math]
Algorithms: in-degree → 1
Compilers → topOrder = [Start, Discrete Math, Compilers]
Senior Seminar: in-degree → 2
Programming 1 → topOrder = [Start, Discrete Math, Compilers, Programming 1]
Programming 2: in-degree → 0 → enqueue
Programming 2 → topOrder = [Start, Discrete Math, Compilers, Programming 1, Programming 2]
Algorithms: in-degree → 0 → enqueue

Computer Organization: in-degree → 0 → enqueue
Algorithms → topOrder = [Start, Discrete Math, Compilers, Programming 1, Programming 2, Algorithms]
Theory of Computation: in-degree → 0 → enqueue
Computer Organization → topOrder = [..., Computer Organization]
Operating Systems: in-degree → 0 → enqueue
High-Level Languages: in-degree → 0 → enqueue
Theory of Computation → topOrder = [..., Theory of Computation]
Operating Systems → topOrder = [..., Operating Systems]
Senior Seminar: in-degree → 1
High-Level Languages → topOrder = [..., High-Level Languages]
Senior Seminar: in-degree → 0 → enqueue
Senior Seminar → topOrder = [..., Senior Seminar]
End: in-degree → 0 → enqueue
End → topOrder = [..., End]

Final Breadth-First Topological Order:
Start, Discrete Math, Compilers, Programming 1, Programming 2, Algorithms, Computer Organization, Theory of Computation, Operating Systems, High-Level Languages, Senior Seminar, End