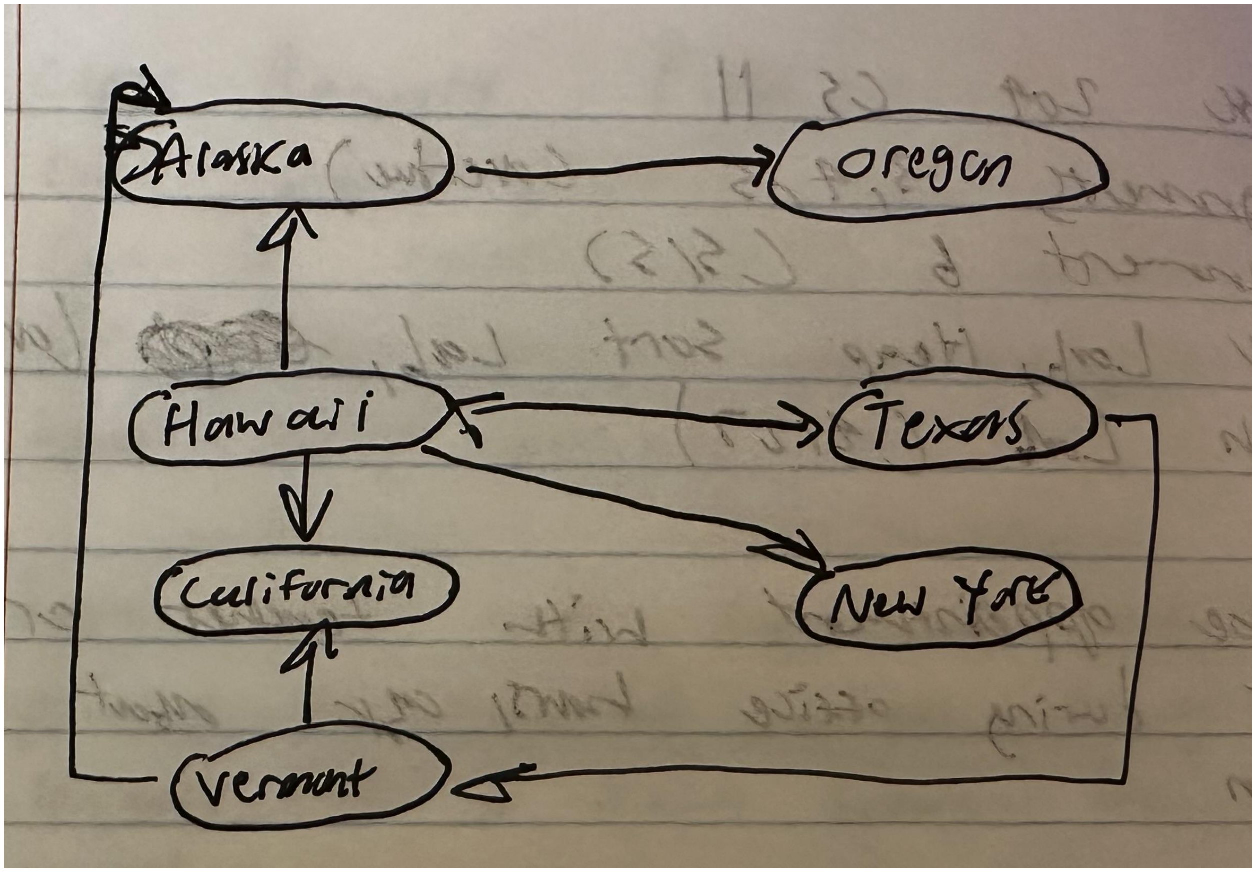
V(StateGraph) = {Oregon, Alaska, Texas, Hawaii, Vermont, NewYork, California}
E(StateGraph) = {(Alaska, Oregon), (Hawaii, Alaska), (Hawaii, Texas), (Texas, Hawaii), (Hawaii, California), (Hawaii, New York), (Texas, Vermont), (Vermont, California), (Vermont, Alaska)}

1. Draw the StateGraph



   a. Is there a path from Oregon to any other state in the graph?
      i. **No, because Oregon is a dead end. It is led to, rather than leading to. (no arrow is coming out of oregon)**
   b. Is there a path from Hawaii to every other state in the graph?
      i. **Yes**
   c. From which state(s) in the graph is there a path to Hawaii?
      i. **Texas**

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

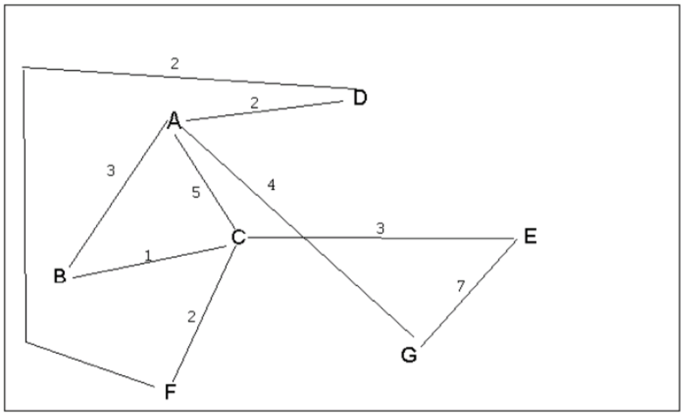
	Alaska	california	Hawaii	NY	oregon	Texas	vermont
Alaska	0	0	0	0	1	0	0
California	0	0	0	0	0	0	0
Hawaii	1	1	0	1	0	1	0
NewYork	0	0	0	0	0	0	0
oregon	0	0	0	0	0	0	0
Texas	0	0	1	0	0	0	1
vermont	1	1	0	0	0	0	0

adjacency matrix

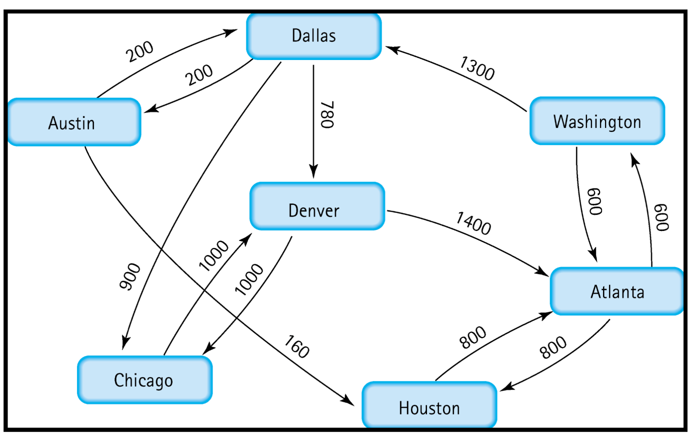
3. Show the adjacency lists that would describe the edges in the graph

adjacency ~~matrix~~ list

Alaska → oregon /
california /
Hawaii → Alaska → california → NewYork → Texas /
New York /
oregon /
Texas → Hawaii → vermont /
Vermont → Alaska → california /

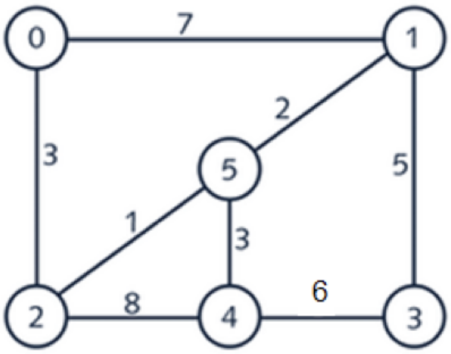


4. Which of the following lists the graph nodes in depth first order beginning with E?
    a. ~~E, G, F, C, D, B, A~~
    b. ~~G, A, E, C, B, F, D~~
    c. E, G, A, D, F, C, B
    d. ~~E, C, F, B, A, D, G~~
5. Which of the following lists the graph nodes in breadth first order beginning at F?
    a. 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~~



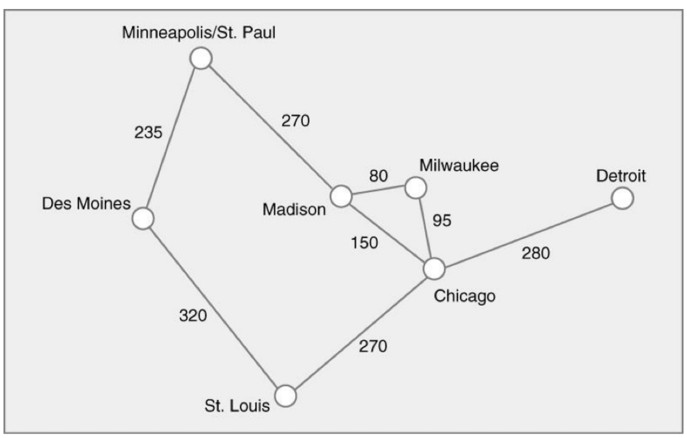
6. Find the shortest distance from Atlanta to every other city

Atlanta to Washington: 600
Atlanta to Houston: 800
Atlanta to Dallas: 1900
Atlanta to Denver: 2680
Atlanta to Austin: 2100
Atlanta to Chicago: 2800



7. Find the minimal spanning tree using Prim's algorithm. Use 0 as the source vertex. Show the steps.
    a. Visited [0], MST []
    b. Look at edges connecting 0 to unvisited nodes (1, 2, 3, 4, 5): 0-1(7), 0-2(3)
    c. Take edge with smallest weight, 0-2(3), add to MST: [0-2]. Add 2 to visited: [0, 2].
    d. Look at edges connecting visited nodes (0, 2) to unvisited nodes (1, 3, 4, 5): 0-1(7), 2-5(1), 2-4(8).
    e. Take edge with smallest weight, 2-5(1), add to MST: [0-2, 2-5]. Add 5 to visited: [0, 2, 5].
    f. Look at edges connecting visited nodes (0, 2, 5) to unvisited nodes (1, 3, 4): 0-1(7), 2-4(8), 5-4(3), 5-1(2).
    g. Take edge with smallest weight, 5-1(2), add to MST: [0-2, 2-5, 5-1]. Add 1 to visited: [0, 2, 5, 1].
    h. Look at edges connecting visited nodes (0, 2, 5, 1) to unvisited nodes (3, 4): 2-4(8), 5-4(3), 1-3(5).
    i. Take edge with smallest weight, 5-4(3), add to MST: [0-2, 2-5, 5-1, 5-4]. Add 4 to visited: [0, 2, 5, 1, 4].
    j. Look at edges connecting visited nodes (0, 2, 5, 1, 4) to unvisited nodes (3): 4-3(6), 1-3(5).
    k. Take edge with smallest weight, 1-3(5), add to MST: [0-2, 2-5, 5-1, 5-4, 1-3]. Add 3 to visited: [0, 2, 5, 1, 4, 3]
    l. All nodes have been visited, so the MST is complete
8. Find the minimal spanning tree using Kruskal's algorithm. Show the weights in order and the steps.
    a. Create a list of all edges, sorted by weight in ascending order: [2-5(1), 5-1(2), 5-4(3), 0-2(3), 1-3(5), 4-3(6), 0-1(7), 2-4(8)]. MST: []
    b. Disjoint set data structure: {0}, {1}, {2}, {3}. {4}, {5}
    c. Iterate through sorted edge list:

i. For edge 2-5(1), nodes 2 and 5 belong to different connected components, so the edge will be added to the MST list: [2-5]. Merge the connected components: {0}, {1}, {2, 5}, {3}, {4}
ii. For edge 5-1(2), nodes 5 and 1 belong to different connected components, so the edge will be added to the MST list: [2-5, 5-1]. Merge the connected components: {0}, {1, 2, 5}, {3}, {4}.
iii. For edge 5-4(3), nodes 5 and 4 belong to different connected components, so the edge will be added to the MST list: [2-5, 5-1, 5-4]. Merge the connected components: {0}, {1, 2, 4, 5}, {3}.
iv. For edge 0-2(3), nodes 0 and 2 belong to different connected components, so the edge will be added to the MST list: [2-5, 5-1, 5-4, 0-2]. Merge the connected components: {0, 1, 2, 4, 5}, {3}.
v. For edge 1-3(5), nodes 1 and 3 belong to different connected components, so the edge will be added to the MST list: [2-5, 5-1, 5-4, 0-2, 1-3]. Merge the connected components: {0, 1, 2, 3, 4, 5}.
vi. All connected components have been joined into one, and the MST list is of length 5 which is one less than the number of nodes in the graph, so the MST is complete

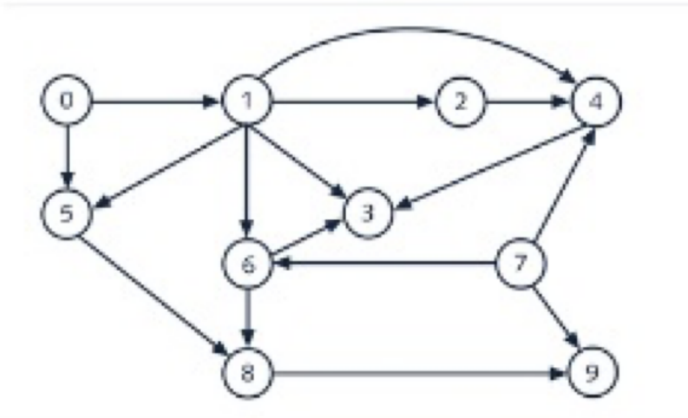


9. Find the minimal spanning tree using the algorithm you prefer. Use Minneapolis/St. Paul as the source vertex

Kruskal's

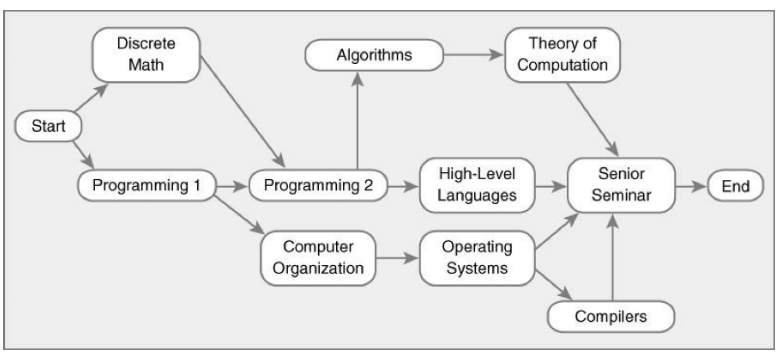
a. Create a list of all edges, sorted by weight in ascending order: [Madison-Milwaukee(80), Milwaukee-Chicago(95), Madison-Chicago(150), Minneapolis/St. Paul-Des Moines(235), Minneapolis/St. Paul-Madison(270), St. Louis-Chicago(270), Chicago-Detroit(280), Des Moines-St. Louis(320]. MST: []
b. Disjoint set data structure: {Chicago}, {Des Moines}, {Detroit}, {Madison}, {Milwaukee}, {Minneapolis/St. Paul}, {St. Louis}
c. Iterate through sorted edge list:

i. For edge Madison-Milwaukee(80), nodes Madison and Milwaukee belong to different connected components, so the edge will be added to the MST list: [Madison-Milwaukee]. Merge the connected components: {Chicago}, {Des Moines}, {Detroit}, {Madison, Milwaukee}, {Minneapolis/St. Paul}, {St. Louis}
ii. For edge Milwaukee-Chicago(95), nodes Chicago and Milwaukee belong to different connected components, so the edge will be added to the MST list: [Madison-Milwaukee, Milwaukee-Chicago]. Merge the connected components: {Des Moines}, {Detroit}, {Chicago, Madison, Milwaukee}, {Minneapolis/St. Paul}, {St. Louis}
iii. For edge Madison-Chicago(150), nodes Chicago and Madison belong to the same connected component, so the edge will be ignored.
iv. For edge Minneapolis/St. Paul-Des Moines(235), nodes Minneapolis/St. Paul and Des Moines belong to different connected components, so the edge will be added to the MST list: [Madison-Milwaukee, Milwaukee-Chicago, Minneapolis/St. Paul-Des Moines]. Merge the connected components: {Des Moines, Minneapolis/St. Paul}, {Detroit}, {Chicago, Madison, Milwaukee}, {St. Louis}
v. For edge Minneapolis/St. Paul-Madison(270), nodes Minneapolis/St. Paul and Madison belong to different connected components, so the edge will be added to the MST list: [Madison-Milwaukee, Milwaukee-Chicago, Minneapolis/St. Paul-Des Moines, Minneapolis/St. Paul-Madison]. Merge the connected components: {Detroit}, {Chicago, Des Moines, Madison, Milwaukee, Minneapolis/St. Paul}, {St. Louis}
vi. For edge St. Louis-Chicago(270), nodes St. Louis and Chicago belong to different connected components, so the edge will be added to the MST list: [Madison-Milwaukee, Milwaukee-Chicago, Minneapolis/St. Paul-Des Moines, Minneapolis/St. Paul-Madison, St. Louis-Chicago]. Merge the connected components: {Detroit}, {Chicago, Des Moines, Madison, Milwaukee, Minneapolis/St. Paul, St. Louis}
vii. For edge Chicago-Detroit(280), nodes Detroit and Chicago belong to different connected components, so the edge will be added to the MST list: [Madison-Milwaukee, Milwaukee-Chicago, Minneapolis/St. Paul-Des Moines, Minneapolis/St. Paul-Madison, St. Louis-Chicago, Chicago-Detroit]. Merge the connected components: {Chicago, Des Moines, Detroit, Madison, Milwaukee, Minneapolis/St. Paul, St. Louis}
viii. All connected components have been joined into one, and the MST list is of length 5 which is one less than the number of nodes in the graph, so the MST is complete



10. List the nodes of the graph in a breadth first topological ordering. Show the steps using arrays predCount, topologicalOrder and a queue
    a. predCount: [0, 1, 1, 3, 3, 2, 2, 0, 2, 2]. topologicalOrder: []. queue: [].
    b. Add all nodes with predCount of 0 to queue: [0, 7].
    c. Process nodes in queue:
        i. Dequeue the next node (0) from the queue and add it to the topologicalOrder list: topologicalOrder: [0], queue: [7].
        ii. Iterate through the neighbors of 0 (1, 5). Decrease predCount values for 1 and 5 by 1: [0, 0, 1, 3, 3, 1, 2, 0, 2, 2]
        iii. Enqueue 1 to the queue since its updated predCount value is 0: [7, 1]
        iv. Dequeue the next node (7) from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7], queue: [1]
        v. Iterate through the neighbors of 7 (4, 6, 9). Decrease predCount values for 4, 6, and 9 by 1: [0, 0, 1, 3, 2, 1, 1, 0, 2, 1]
        vi. Dequeue the next node (1) from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1], queue: [].
        vii. Iterate through the neighbors of 1 (2, 3, 4, 5, 6). Decrease predCount values for them by 1: [0, 0, 0, 2, 1, 0, 0, 0, 2, 1]
        viii. Enqueue 2, 5, and 6 to the queue since their updated predCount values are 0: [2, 5, 6]
        ix. Dequeue the next node from the queue and add it to the topologicalOrder list: topologcalOrder: [0, 7, 1, 2], queue: [5, 6].
        x. Iterate through the neighbors of 2 (4). Decrease the predCount value of it by 1: [0, 0, 0, 2, 0, 0, 0, 0, 2, 1].
        xi. Enqueue 4 to the queue since its updated predCount value is 0: [5, 6, 4].
        xii. Dequeue the next node from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1, 2, 5], queue: [6, 4].

xiii. Iterate through the neighbors of 5 (8). Decrease the predCount value of it by 1: [0, 0, 0, 2, 0, 0, 0, 0, 1, 1].
xiv. Dequeue the next node from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1, 2, 5, 6], queue: [4].
xv. Iterate through the neighbors of 6 (3, 8). Decrease the predCount value of them by 1: [0, 0, 0, 1, 0, 0, 0, 0, 0, 1].
xvi. Enqueue 8 to the queue since its updated predCount value is 0: [4, 8]
xvii. Dequeue the next node from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1, 2, 5, 6, 4], queue: [8].
xviii. Iterate through the neighbors of 4 (3). Decrease the predCount value of it by 1: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
xix. Enqueue 3 to the queue since its updated predCount value is 0: [8, 3]
xx. Dequeue the next node from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1, 2, 5, 6, 4, 8], queue: [3]
xxi. Iterate through the neighbors of 8 (9). Decrease the predCount value of it by 1: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0].
xxii. Enqueue 9 to the queue since its updated predCount value is 0: [3, 9].
xxiii. Dequeue the next node from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1, 2, 5, 6, 4, 8, 3], queue: [9].
xxiv. Dequeue the next node from the queue and add it to the topologicalOrder list: topologicalOrder: [0, 7, 1, 2, 5, 6, 4, 8, 3, 9], queue: [].



11. List the nodes of the graph in a breadth first topological ordering

[Start, Discrete Math, Programming 1, Programming 2, Computer Organization, Algorithms, High Level Language, Operating System, Theory of Computation, Compilers, Senior Seminar, End]