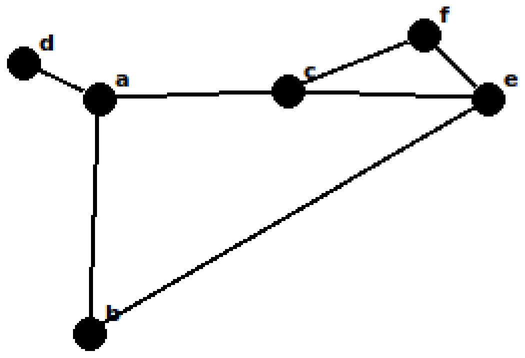
1. Draw the below graphs and then write the size,order, deg(v), adj(v) of each graph.

1. $E = \{\{a,b\},\{b,e\},\{a,c\},\{a,d\},\{c,e\},\{e,f\},\{c,f\}\}$

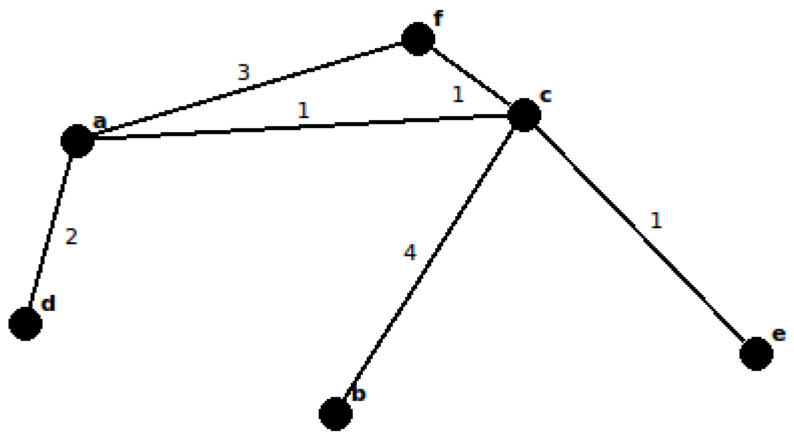


$$\begin{aligned} deg(a) &= 3 \\ deg(b) &= 2 \\ deg(c) &= 3 \\ deg(d) &= 1 \\ deg(e) &= 3 \\ deg(f) &= 2 \end{aligned}$$

$$\begin{aligned} adj(a) &= \{b,c,d\} \\ adj(b) &= \{a,e\} \\ adj(c) &= \{a,e,f\} \\ adj(d) &= \{a\} \\ adj(e) &= \{c,f\} \\ adj(f) &= \{c,e\} \end{aligned}$$

order(G)=6,size(G)=14

2. $E = \{\{a,f,3\},\{b,c,4\},\{a,c,1\},\{a,d,2\},\{c,e,3\},\{c,f,1\}\}$

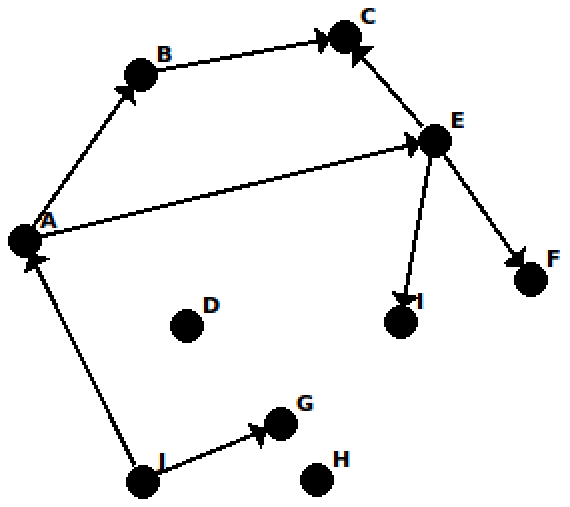


$$\begin{aligned} deg(a) &= 3 \\ deg(b) &= 1 \\ deg(c) &= 4 \\ deg(d) &= 1 \\ deg(e) &= 1 \\ deg(f) &= 2 \end{aligned}$$

$$\begin{aligned} adj(a) &= \{c, d, f\} \\ adj(b) &= \{c\} \\ adj(c) &= \{a, b, e, f\} \\ adj(d) &= \{a\} \\ adj(e) &= \{c\} \\ adj(f) &= \{a, c\} \end{aligned}$$

Order(G)=6, Size(G)=12

3. $E = \{(j, a), (j, g), (a, b), (a, e), (b, c), (e, c), (e, f), (e, i)\}$

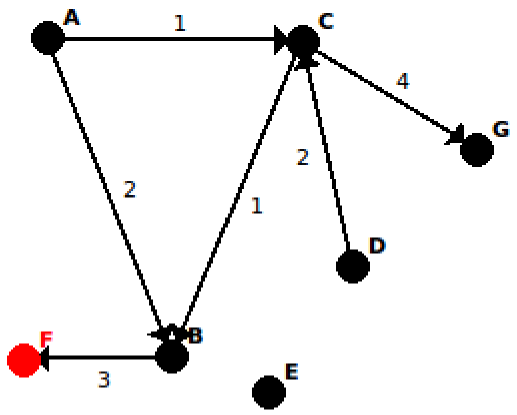


$$\begin{aligned}
deg^{-}(a) &= 1\\
deg^{+}(a) &= 2\\
deg^{-}(b) &= 1\\
deg^{+}(b) &= 1\\
deg^{-}(c) &= 2\\
deg^{+}(c) &= 0\\
deg^{-}(d) &= 0\\
deg^{+}(d) &= 0\\
deg^{-}(e) &= 1\\
deg^{+}(e) &= 3\\
deg^{-}(f) &= 1\\
deg^{+}(f) &= 0\\
deg^{-}(g) &= 1\\
deg^{+}(g) &= 0\\
deg^{-}(h) &= 0\\
deg^{+}(h) &= 0\\
deg^{-}(i) &= 1\\
deg^{+}(i) &= 0\\
deg^{-}(j) &= 0\\
deg^{+}(j) &= 2
\end{aligned}$$

$$
\begin{array}{rcl}
adj(a) & = & \{b, e\} \\
adj(b) & = & \{c\} \\
adj(c) & = & \{\} \\
adj(d) & = & \{\} \\
adj(e) & = & \{c, i, f\} \\
adj(f) & = & \{\} \\
adj(g) & = & \{\} \\
adj(h) & = & \{\} \\
adj(i) & = & \{\} \\
adj(j) & = & \{a, g\}
\end{array}
$$

$order(G) = 11, size(G) = 8$

4. $E = \{(a, b, 2), (a, c, 1), (b, f, 3), (c, b, 1), (c, g, 4), (d, c, 2)\}$



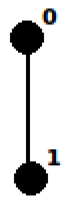
$$\begin{aligned}
deg^-(a) &= 0\\
deg^+(a) &= 2\\
deg^-(b) &= 2\\
deg^+(b) &= 1\\
deg^-(c) &= 2\\
deg^+(c) &= 2\\
deg^-(d) &= 0\\
deg^+(d) &= 1\\
deg^-(e) &= 0\\
deg^+(e) &= 0\\
deg^-(f) &= 1\\
deg^+(f) &= 0\\
deg^-(g) &= 1\\
deg^+(g) &= 0
\end{aligned}$$

$$\begin{aligned}
adj(a) &= \{b, c\}\\
adj(b) &= \{f\}\\
adj(c) &= \{b, g\}\\
adj(d) &= \{c\}\\
adj(e) &= \{\}\\
adj(f) &= \{\}\\
adj(g) &= \{\}
\end{aligned}$$

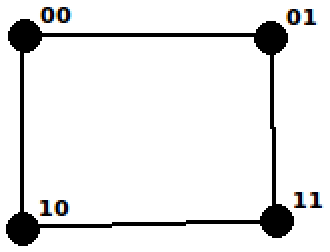
$order(G) = 7, size(g) = 6$

2. The graph Q_n, $n \geq 1$, has vertex set equal to the set of all binary strings of length n. Moreover, two vertices are adjacent iff they differ in at most one bit place. For example, in Q_3, 000 is adjacent to 010, but not to 011. Draw Q_1, Q_2 and Q_3. Show that Q_3 has a Hamilton cycle.

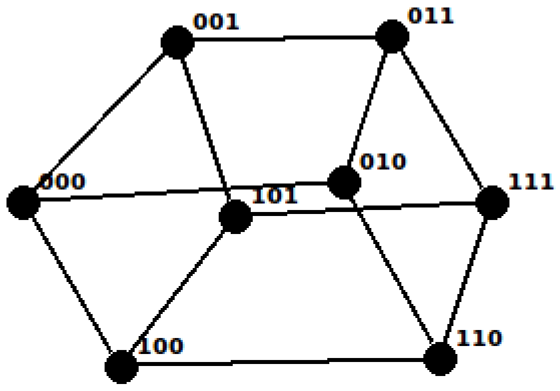
1. Q1



2. Q2



3. Q3



000 → 010 → 110 → 111 → 011 → 001 → 101 → 100 → 000 is a hamiltonian cycle

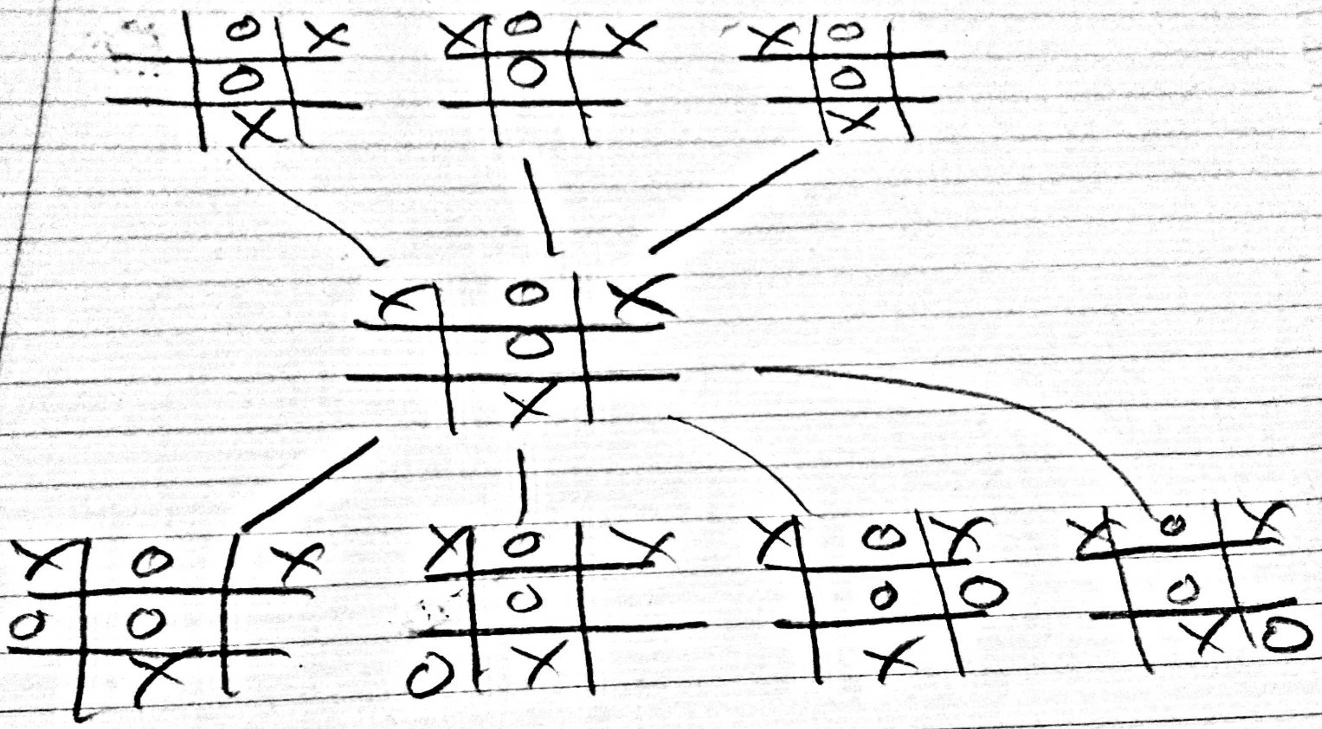
3. Provide formulas for both the order and size of Q_n. Explain.

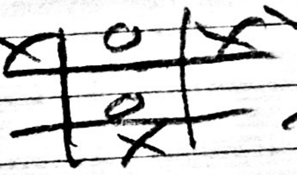
There are a total of 2^n bitstrings of length n, hence the order is 2^n. For any bitstring there are n different for a bitstring to differ in one bit, hence the degree of any vertex is n.

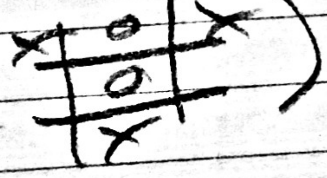
$$\begin{aligned}
\sum_{v \in V(Q_n)} deg(v) &= 2 \cdot |E(Q_n)| \\
\sum_{v \in V(Q_n)} n &= 2 \cdot |E(Q_n)| \\
2^n \cdot n &= 2 \cdot |E(Q_n)| \\
2^{n-1} \cdot n &= |E(Q_n)|
\end{aligned}$$

Thus the size of the graph Q_n is $2^{n-1} \cdot n$.

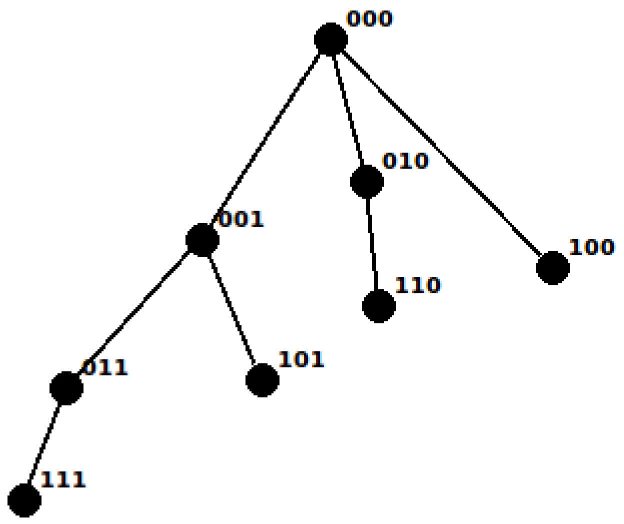
4. Consider the directed graph where vertices are reachable tic-tac-toe board positions and edges represent valid moves. What are the in-degree and out-degree of the following vertex?



$\deg^-$() $= 3$

$\deg^+$() $= 4$

5. Starting at vertex 000, perform a BFS of Q3. Assume all adjacency lists are in numerical order. Show the resulting spanning tree.



7. What is the running time of

1. Breadth-first search
2. Depth-first search

as a function of $|V|$ and $|E|$, if the input graph is represented by an adjacency matrix, instead of an adjaceny list

The running time of both algorithms will be $O(|V|^2)$.The alteration of the data structure alters the time it takes to check the neighbors, it now takes $O(V)$ as opposed to $deg(v_i)$ for any $v_i \in V$. Thus the sum $\sum_{v\in V} deg(v) = O(|E|)$, for the adjacency list representation, becomes $\sum_{v\in V}|V| = O(|V|^2)$ for the adjacency matrix. We have $O(|V| + |V|^2) = O(|V|^2)$.

9. Calculate the worst-case running time for Dijkstra's algorithm if nodes are stored in a binary-heap.

The worst case running time of dijkstra's algorithm if nodes are stored in a binary-heap is $O((V + E) \cdot log(V))$. We first initialize distance array and parent pointer array this is $O(V)$. We construct a min-heap from the vertex set, this is

$O(V)$. The number of iterations is at most V and any removal of an element is $O(log(V))$ hence $O(V \cdot log(V))$. Any vertex checks all its neighbors and alter the distance if a better path has been found changed, then repriortizes the neightbor in the min-heap with a heapify operation. We have $\sum_{v \in V(G)} deg(v_i) \cdot log(V) = O(E \cdot log(V))$. In total we have $O((V + E) \cdot log(V))$.

10. Draw the weighted unweighted graph whose edges-weights are given by

$$\begin{aligned} E &= \{\{a,b,2\},\{a,c,1\},\{b,f,3\},\{c,b,1\},\{c,g,4\},\{d,c,2\},\{d,e,3\},\{e,f,7\},\{g,f,5\},\{g,h,4\}\} \\ E &= \{\{a,b,3\},\{a,d,4\},\{a,e,4\},\{b,c,10\},\{b,e,2\},\{c,f,6\},\{c,g,1\},\{d,e,5\},\{d,h,6\},\{e,f,11\}\} \end{aligned}$$

1. Perform Kruskal's algorithm
2. Perform Prim's algorithm

11. Does Prim's and Kruskal's algorithm work if negative weights are allowed? Explain.

The minimum spanning tree for a connected weighted graph is a spanning tree that has the least possible weights among all possible spanning tree for the graph. The definition of a minimum spanning tree does not exclude having negative weights in a spanning tree. Prim's and Kruskal's will both work when a negative weight is in a graph and will produce a valid MST.

12. Explain how Prim's and/or Kruskal's algorithm can be used to find a maximum spanning tree.

Go through the graph marking and negating an edge when you visit it. Now when Kruskal's or Prim's is used, the algorithm will select the largest possible valid edge in the candidate set in the original graph for every iteration resuling in a maximum spanning tree.

13. Suppose that we represent the graph $G = (V, E)$ as an adjacency matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(V^2)$.

Algorithm 1 Prim for Adjacency Matrix

```
Prim(G, s) :
cost = [inf for v in G]
pred = [-1 for v in G]
cost[s] = 0
$Q = \{s\}$
while $Q \neq \emptyset$:
    curr = Q.pop()
    for(i = 0;i < |V(G)|;i++):
        if G[curr][v] < wt[v]:
            wt[v] = G[curr][v]
            pred[v] = curr
            Q.push(v)
```

14. What is the running time fo the most efficient algorithm you know for finding the shortest path between two vertices in a directed graph, where the weight of all edges are equal?

We could compute the shortest path from one vertex to another vertex in $O(|V|+|E|)$ using breadth-first search in the case where the weights of a weighted graph are equal.

15. Give an algorithm that determines where or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time.

The most edges that a connected graph that is acyclic can contain is $|V|-1$ edges. If the graph has more than $|V|-1$ edges we can immediately conclude that the graph contains a cycle. Now we run the DFS cycle detection algorithm we already have from lecture, this algorithm runs in $O(|V|+|E|)$ time but we know $|E| \leq |V|-1$ hence we have $O(|V|)$.

16. Give a linear-time algorithm that determines if a simple graph has any odd cycles.

Perform a breadth-first traversal and mark the visited notes with one of two colors(either red or blue). Whenever a node that is removd from the queeu reaches an unvisited node, mark that node and give it the opposite color of its parent. If a node has been already visited and marked the opposite color continue the algorithm. If the node has been already visited and marked as the same color as the current node being processed then the graph is not two-colorable and hence not bipartite. Since the graph is not bipartite it must have an odd length cycle. This algorithm uses breadth-first search and is hence $O(|V|+|E|)$ given an adjancency-list representation.

Algorithm 2 odd cycle detection

```
ColorGraph(G) :
 1: $Q = \{\}$
 2: for $v \in V$ :
 3:     if $v$ isn't colored :
 4:         v.color = blue
 5:         $Q.push(v)$
 6:         while $Q$ not empty :
 7:             x = pop($Q$)
 8:             for $w = adj(x)$ :
 9:                 if $w$ isn't colored :
10:                     color it the opposite color of x
11:                     $Q.push(w)$
12:                 if $w$ is colored and same color as x:
13:                     return True
14: return False
```