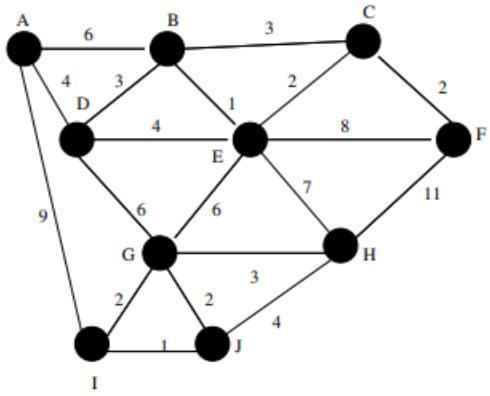
5.1



G1

BFS: A -> B -> D -> I -> C -> E -> G -> J -> F -> H

DFS: A -> B -> C -> E -> D -> G -> H -> F -> J -> I

5.4

To prove this, we can look at following cases:

Assume (u,v) is a backward edge. This indicates that in the T-tree, v is an ancestor of u. Since there is no direction to the edges in an undirected graph, this is not possible. As a result, (u, v) is not a back edge.

Assume (u,v) is a forward edge. It would have already been visited before we got to v and tried to visit u throughout our search. There can be no forward edges in an undirected graph because you cannot visit the descendant before visiting the ancestor.

Assume (u,v) is a cross edge. Then there are relationships between siblings in the tree. If cross edges aren't formally defined in terms of real connections, they're visible in how our output array of BFS exploration looks.

If none of these cases apply for then (u,v) must be a tree edge or cross edge in the BFS traversal and every edge in the BFS on an undirected graph is either a tree edge or cross edge.

5.6

(a) In BFS we use Queue data structure. According to the definition, a node is said to be discovered when it is inserted into queue and is said to be processed when it exits the queue. To add the whole element of node into queue, discovered state, we have to deque at least one element, which have all other node as it's adjacent. So, when our current node is dequeued then all adjacent node(n-1) will be added to the queue, discovered state. If we have a graph(G) with n nodes and starting node is adjacent to all other nodes (n-1) nodes then a BFS on such graph can give us theta(n) nodes in discovered state.

(b) In Depth First Search we use stack data structure. According to the definition a node is in discovered state while it is present in the stack, and in DFS, we only pop any node, if it does not have any unvisited adjacent nodes or while backtracking. Thus, if a graph (G) with n nodes is connected and every node is adjacent to only two nodes, we can get the DFS which will give us theta(n) nodes in discovered state.

5.18

This problem can be modeled as a bipartite graph where the left partition consists of the movies M1, M2, ..., Mk and the right partition consists of the customers. There is an edge between a movie and a customer if the customer wants to see that movie. If a perfect matching exists, we can assign all movies that are matched on the left side to be screened on Saturday and the movies that are matched on the right side to be screened on Sunday. This ensures that each customer gets to see the two movies they desire, and each movie is shown at most once.

Algorithm:

Pick a source vertex u where and assign a color red putting into a set Saturday

Color all the neighbors of u blue putting into a set Sunday

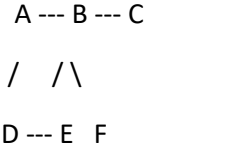
Color all the neighbor’s neighbor red and putting into set Saturday

Assign colors to all vertices such that no two vertices with same colors have an edge between them

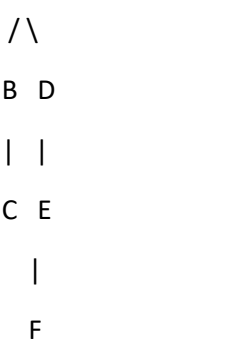
If a neighbor is found which is colored with the same color as the current vertex, then the graph is not bipartite.

5.30

No every bridge e must be an edge in a DFS tree of G. Consider the counter example below.



we perform a DFS traversal of G starting from vertex A. The order in which the vertices are visited is A, B, C, E, D, F. The DFS tree rooted at A looks like this:



In this DFS tree, the edges AB, BD, DE, and EF are tree edges, while the edges AC, BE, and BF are back edges. The edge BC is a cross edge. If we remove the edge BC, the resulting graph is disconnected, with two connected components: {A, B, C} and {D, E, F}. Therefore, the edge BC is a bridge in G.