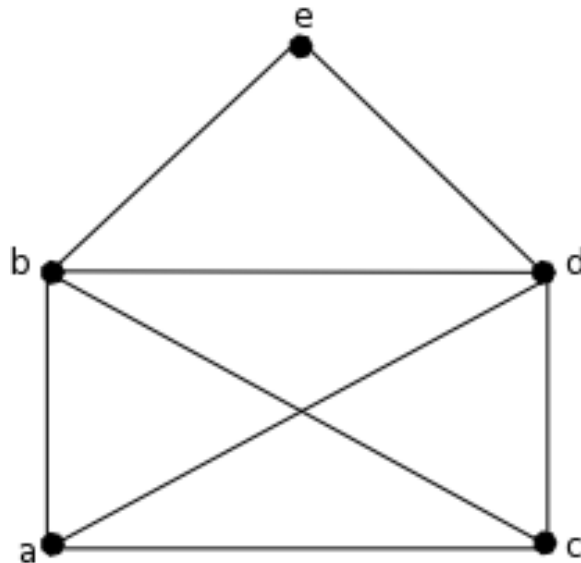


C241 Assignment 11: Graph Theory

Due Wednesday 12/9/09

1) Use the following undirected graph $G = (V, E)$ to answer the questions below:



- List the vertex set, V , and edge set, E , for this graph.
- The 'degree' of a vertex is the number of edges connected to it. Give an example of a vertex of degree 4.
- Give an example of a path of length greater than 3 from vertex b to d .
- Give an example of a cycle starting at a .
- Remember that a graph G' is a 'subgraph' of G if $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$. Give an example of a subgraph of graph G (draw it).
- We say that a graph is 'connected' if for every pair of vertices in the graph there is some *path* that connects them (in other words, every vertex is reachable from every other vertex). This graph is connected. Which edges could you eliminate to make it so this graph was not connected, so that it had at least two separate components (so no paths connect the vertices in one component to the vertices in the other component)? (draw the resulting graph)

g) A 'spanning tree' of G is a subgraph $T = (V_T, E_T)$ which does not include any cycles, and which does include all the vertices of G (so $V_T = V$). Give two examples of spanning trees for this graph (draw them).

h) For undirected graphs, a 'clique' of G is a subgraph of G in which every pair of vertices is adjacent. In other words, if $C = (V_C, E_C)$ is a clique in G , then C is a subgraph of G , and for all $u, v \in V_C$, if $u \neq v$ then the edge $(u, v) \in E_C$. What's the largest clique in G (draw it)? What's the smallest clique in G (draw it)?

i) What does it mean for a graph to be planar? Is this graph planar? (if yes, then prove it by re-drawing the graph appropriately, if no, explain why not).

2) An undirected complete graph is an undirected graph in which every vertex is directly connected by an edge to every other vertex.

a) Draw a complete graph with 5 vertices.

b) If you could check the degree of a vertex in constant time, how could you check whether a graph with n vertices (and no reflexive edges) was complete in $O(|V|)$ time?

c) What's the shortest spanning tree for a complete graph with n vertices?

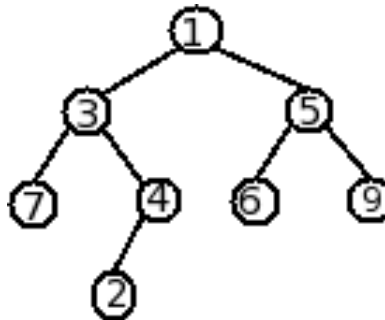
d) What's the tallest spanning tree for a complete graph with n vertices?

e) *Bonus* How many different cliques, of any size, does a complete graph of size n have?

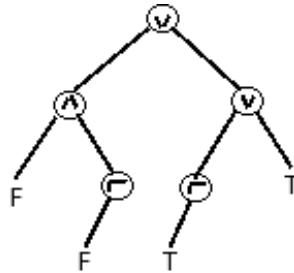
3) *Bonus:* An Euler Circuit is a cycle which includes every edge in the graph without repeating any edges. Does the graph from problem 1 have an Euler circuit? If it does have one, draw it. If it doesn't have one, explain why it doesn't. (Hint, what happens if the graph has a vertex with odd degree?)

4) A tree traversal is a recursive Depth-First Search algorithm for examining the values in every node of a tree. For binary trees there are three types of traversals: pre-order, in-order and post-order. Each algorithm starts with the root, at the top of the tree. A pre-order traversal first checks the value in the current node, and then recursively calls itself on the current node's left and then right subtrees (if they exist). A post-order traversal starts by recursively calling itself on the left and then the right subtrees (if they exist), and *then* it examines the value in the current node. An in-order traversal starts by calling itself on the left subtree, then checks the current node, and finally calls itself on the right-subtree (this traversal only makes sense when we're looking at binary trees, the others can be used in general).

- a) Write out all the values in this tree in the order they would be reached in a pre-order traversal.
- b) Write out all the values in this tree in the order they would be reached in a in-order traversal.
- c) Write out all the values in this tree in the order they would be reached in a post-order traversal.
- d) Which traversal should we use if we wanted to efficiently discover whether the number 13 was in the tree? (why?).
- e) If this tree were a binary search tree, which traversal could we use to print out the contents of the tree in sorted order? (why?).



5) A parse tree is used to represent the underlying structure of statements. For instance, one might represent the boolean statement: $(F \wedge \neg F) \vee (\neg T \vee T)$ as the tree drawn below (where the truth values are stored in the leaves of the tree, and the operations are stored in the interior nodes). Write psuedo-code which, given a tree like this, will return the value of the boolean statement that the tree represents (either T or F). What type of tree-traversal does your solution use?



6) Below is pseudo code for the basic Depth First Search algorithm on general directed graphs (not just trees). Use it to answer the following questions.

```

Outer_DFS(v)
{
    initialize_visited();    //the visited array keeps track of the
                            //vertices we've visited

    for each v in V;        //V is the set of all vertices in the graph
    {
        if not( visited[v] == 1 )
        { Inner_DFS(v); }
    }
}

Inner_DFS(v)
{

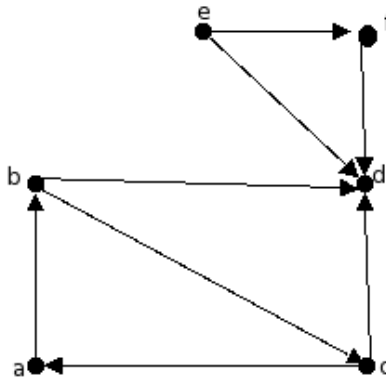
```

```

visited[v] = 1;

for each n in neighbors(v); //the neighbors of v are the vertices that
{
    //have an edge leading to them from v
    if not(visited[n] == 1)
    { Inner_DFS(n);}
}
}

```



- a) List the vertices of the graph above in the order they would be visited by the Depth-First-Search algorithm starting on vertex *a* (where a vertex is 'visited' when its spot in the Visited array is set to 1). Assume that the "for each" loops visit vertices in alphabetical order.
- b) Why is the OuterDFS function necessary?
- c) Why is the Visited array necessary?
- d) How could we use this algorithm to find what vertices can be reached by a path starting from *a*? How could we find out vertices can be reached from *e*? (Describe your algorithm in specific, detailed english; but don't write out the pseudo code).
- e) How could we adapt this algorithm to determine whether the graph *G* had any cycles? (Describe your algorithm in specific, detailed english) (Hint: look at your answer for c)