7

Searching Graphs

Algorithms for searching graphs go back at least to the nineteenth century when they were used for mazes. Until the advent of computers they were an extremely obscure topic (as was graph theory). We are going to look at two algorithms which represent the two main approaches to searching graphs. By a graph search we mean some process that visits each node of a graph (arcs are merely the highways by which we perform the search). These algorithms known as *breadth first search* (BFS) and *depth first search* (DFS) are polar opposites. Think of them as two puppies. BFS is timid, he does not go ten yards from the den until he has explored everything within five yards. DFS is adventurous, she goes as far as she can in any one direction before she tries a different direction.

Preliminaries

We are going to consider only directed graphs. As a rule undirected graphs are easier. There are two ways to adapt the algorithms given in this chapter to undirected graphs. First, you can replace each undirected arc by two arcs going in opposite directions. Secondly, you can simply rewrite (and simplify) the algorithm. It is worth noting that the BFS algorithm is somewhat easier to perform in virtually any context than DFS. Both algorithms can be written in recursive versions, with the recursive version of DFS being somewhat sophisticated. The versions given here however, are non-recursive.

Each search has to begin with some node. This node is called a *root*. If the user has a specific root in mind then you need to modify the algorithms accordingly (this is trivial). Normally, for aesthetic reasons, we prefer as a root a node that has no incoming arcs: we say it has no *antecedents*. Such a node may not always be available. Also, search algorithms always require the labeling of nodes(sometimes we say we *mark* the node). This is essentially a record

keeping device. After searching from a given root, the search may terminate (temporarily) without having reached all of the nodes. Hence the search must be restarted from a new node. As a result when we are finished we are said to have created a *spanning forest* of the graph (the routes we follow involve no circuits).

The Breadth First Search Algorithm

{Initialization}
Label each node −1; {−1 means that the node has not been visited}
Set d←0;
{Main Program}
While there are nodes labeled −1 {choose a root}
    Choose such a node with no antecedents label it d;
    Else {there is no node labeled −1 and without antecedents} choose a node labeled −1 and label it d;
    d←d+1;
    While there are nodes labeled -1 that can be reached from node labeled d
        Mark each node d+1 {that is those nodes just referred to}
        d←d+1.
    {end While}
{end While}

As long as there are nodes that haven't been reached the algorithm must continue. The algorithm works in stages corresponding to the value of d. If an unreached node can be reached from a node labeled d, it is labeled d+1. Occasionally it might happen that no unreached nodes can be accessed from a node labeled d. In that case the algorithm goes back to the first **while** and chooses a new root.

Example The graph in **TR** will be used to illustrate both BFS and (later) DFS.

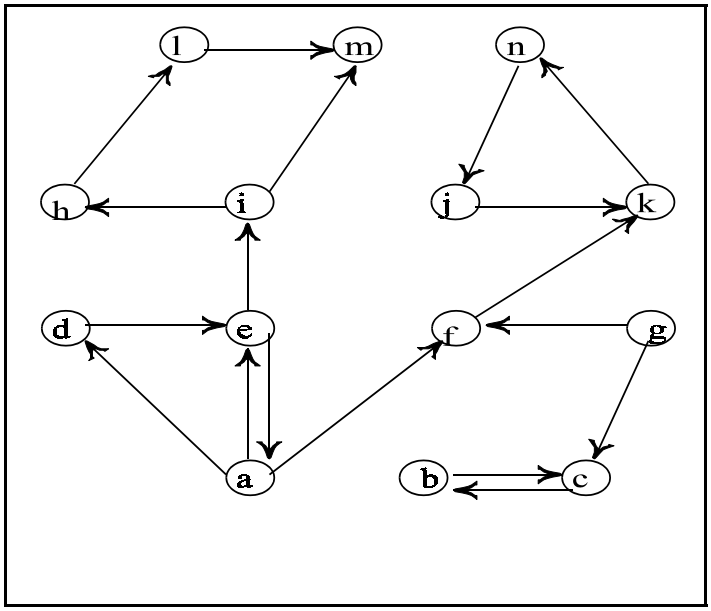


Figure 1 A Graph to Search

The following figures illustrate BFS. Note that the unreached nodes that would be labeled −1 are simply unlabeled.

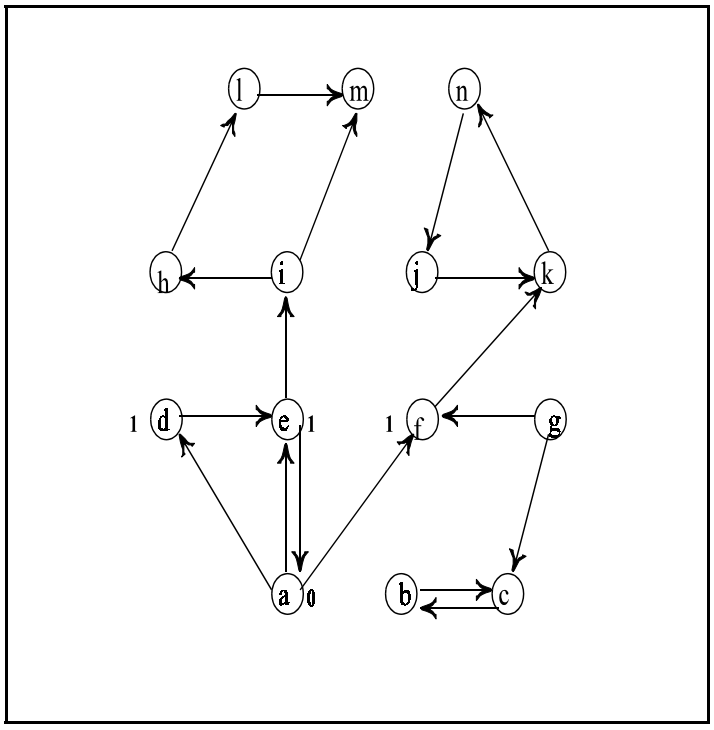


Figure 2 The Root is node a

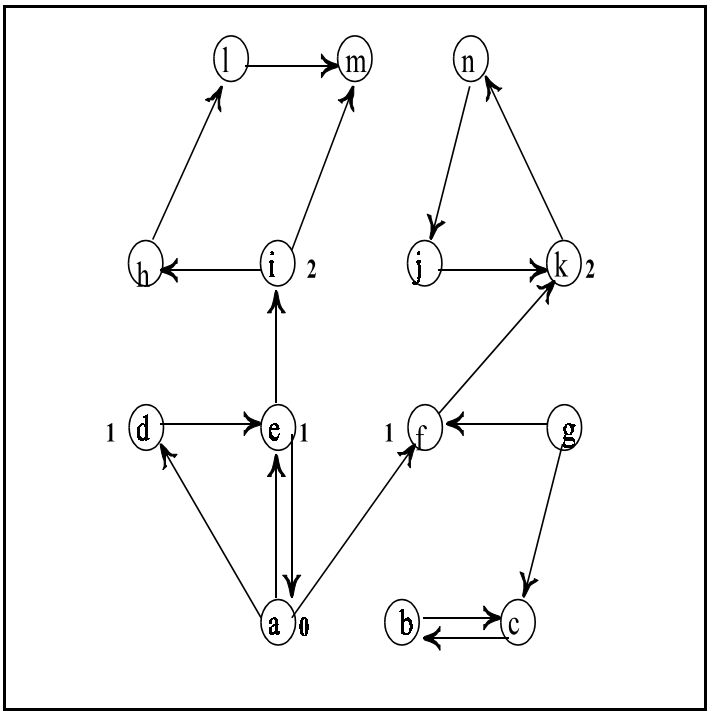


Figure 3 d = 2

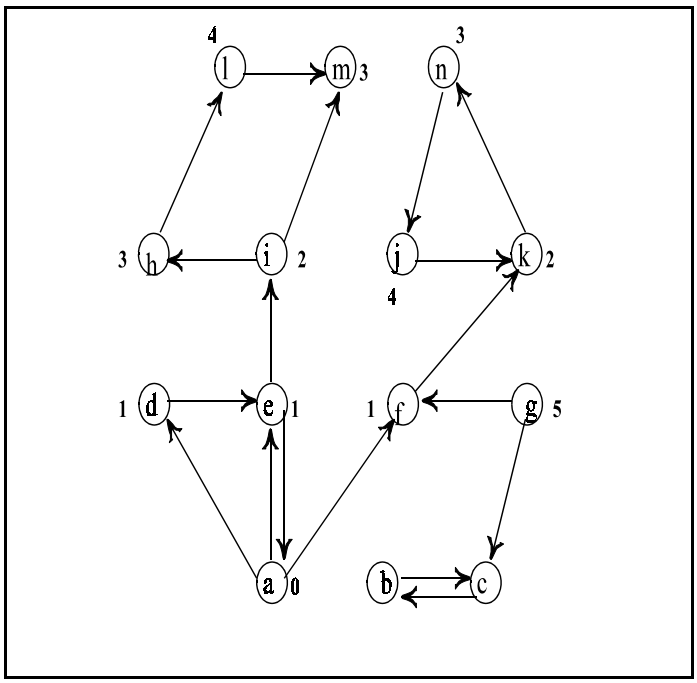


Figure 4 There is a Second Root at g

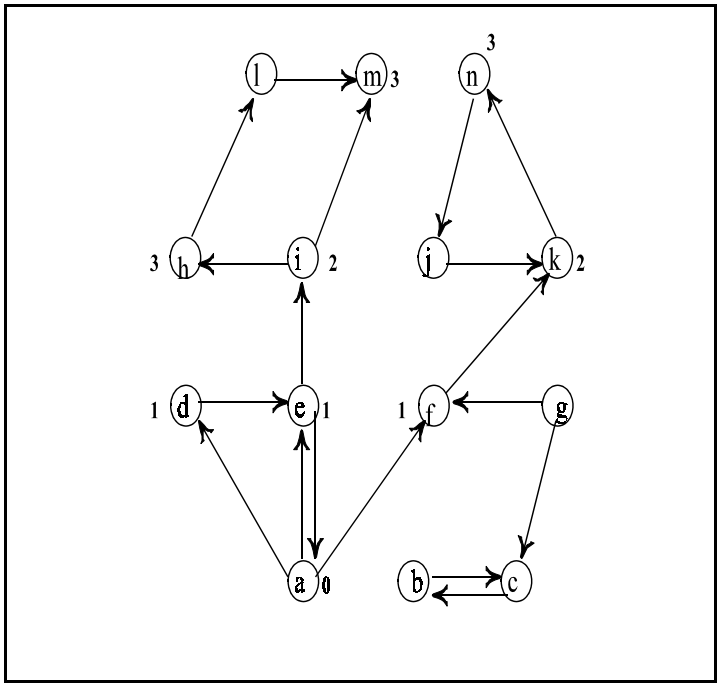


Figure 5 d = 3

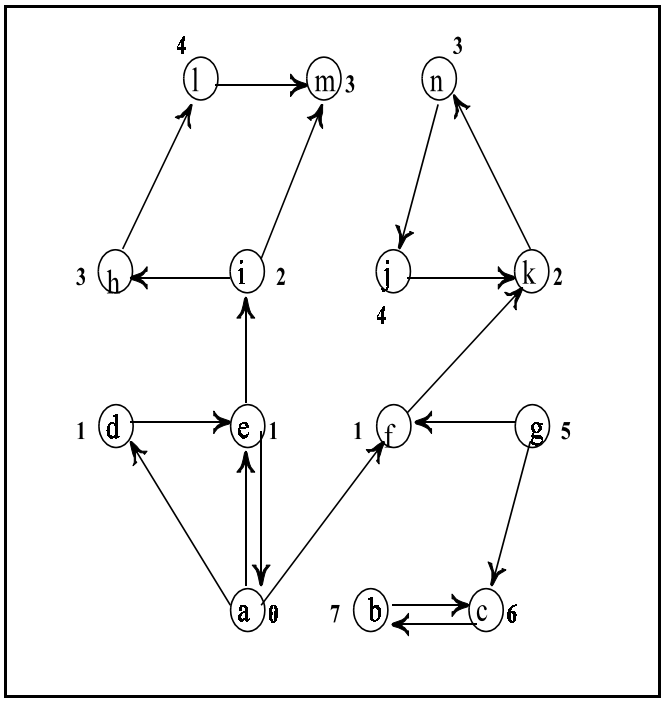


Figure 6 The Finished BFS Product

The Depth First Search Algorithm

In DFS each node has two labels *v* and *l*. The label v refers to when the node was first visited and the label l (for *leave*) refers to when it was last visited. Once the l-value of a node is set to a positive number then v<l for that node. At the beginning of the algorithm each node is marked (0,0). Once the algorithm is finished, the only 0 is the v coordinate of the first root. Given two nodes labeled (v_1,l_1) and (v_2,l_2) there are two possibilities. Either one pair is within the other pair as here: $v_1<v_2<l_2< l_1$ or the pairs are disjoint as here: $v_1,l_1 < v_2,l_2$. If the pairs are disjoint then they fall along different branches of the search tree. Otherwise, the inner pair is deeper in the same tree as the outer pair (by *deeper* we mean further from the root). We will separate the DFS algorithm into four distinct procedures: *Get Root, Go Forward, Go Backward* and *DFS*. Note that once the algorithm is underway, there is a *current* node, C. If a node N has label (x,y) we write V(N) = x and L(N) = y. At any given time the algorithm is designed to go

forward (that is further from the root). When it can't go further it backs up to the prior node and tries to go further down another path. When the algorithm cannot go further and cannot back up then the algorithm looks for another root. When the algorithm looks for another root and fails to find one, it terminates. The DFS algorithm is as follows:

```
Get Root {This procedure is only run when it is called from DFS}
    If there is a node labeled (0,0) with no antecedent
        Then make this node C {it is current} and set V(C)←T;       {T for time}
        Else If there is a node labeled (0,0) {and all such nodes have antecedents}
            Make it C and set V(C)←T;
        Else Stop {algorithm is finished};
        T←T+1. {end Get Root}


Go Forward {This procedure is only run when it is called from DFS}
    Iterate2 ← True;  Iterate←False
        While Iterate2
            If an arc goes from C to a node labeled (0,0)
            Then make it C and set V(C)←T; Iterate←True; T←T+1;
            Else Iterate2 = False;
    {end Go Forward}.


Go Backward {This procedure is only run when it is called from DFS}
    Iterate←False;
    L(C)←T
    If there is an antecedent node
        Then
            Find an antecedent node, N, with V(N) = V(C)-1;
            Set L(C)←T;
            C←N; {make N current}
        Else    Iterate←False;
    T←T+1.{End Go Backward}.



DFS {This is the main procedure}
    Mark each node (0,0);
    Set T←0
    Repeat
        Get Root {termination occurs in Get Root}
        Iterate←True
        While Iterate
            Go Forward
            Go Backward
    {end DFS}.
```

Example The graph in **TR** is used to demonstrate DFS. unmarked vertices represent those labeled (0,0). The label *2,* is short for (2,0). Note that when node i is first reached the algorithm can proceed in two different ways with two different results.

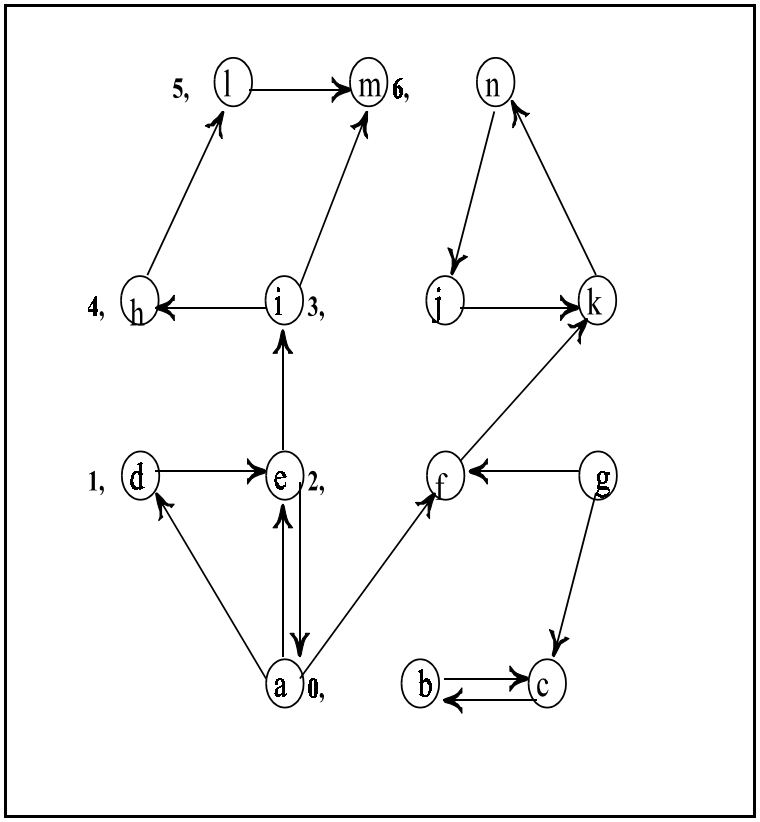


Figure 7 The First Forward Search.

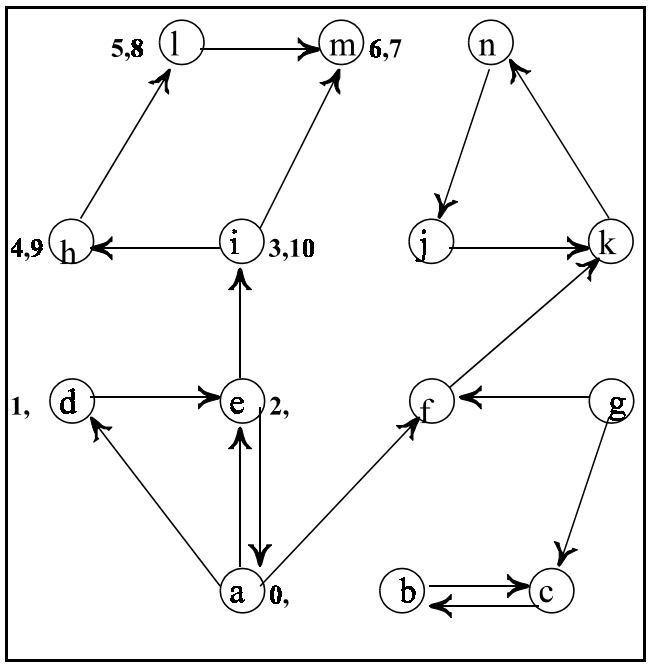


Figure 8 Three Iterations of Backward Search

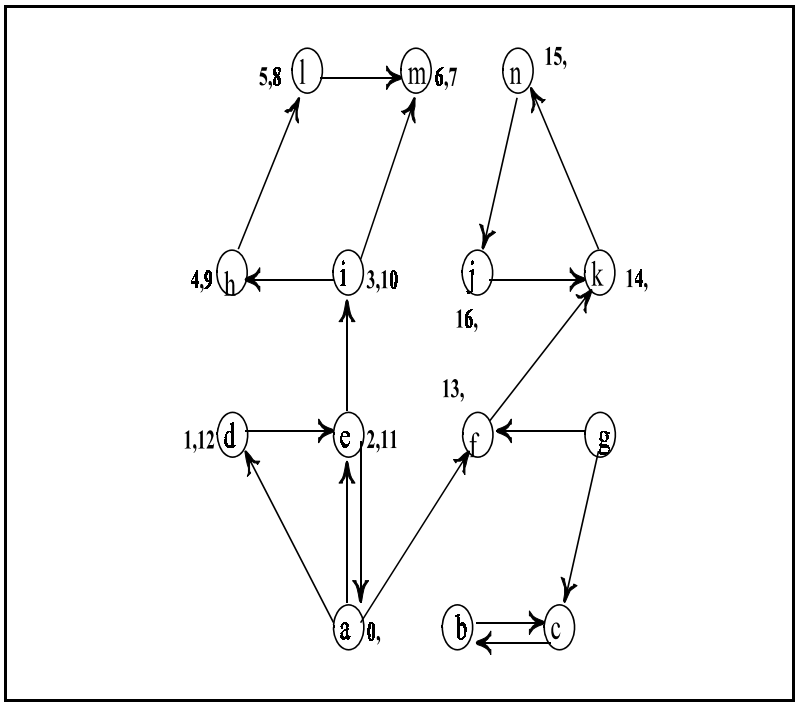


Figure 9 Back to the Root

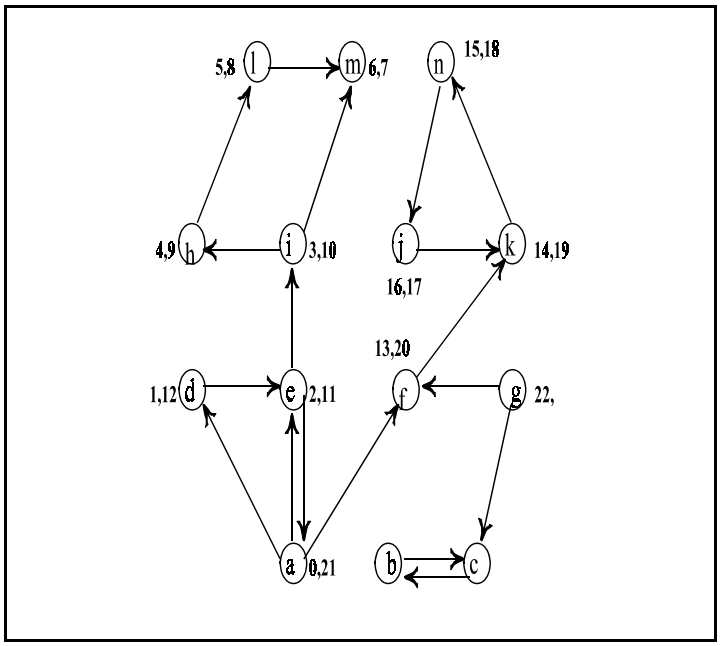


Figure 10 The First Tree is Finished

☐ **Exercise 1** Do BFS on this graph:

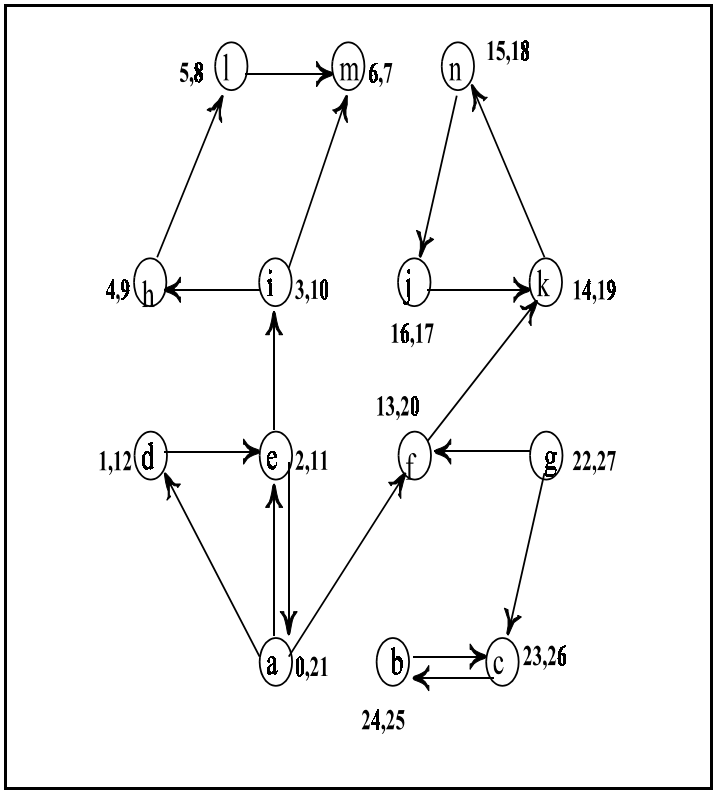


Figure 11 The Finished Search: Two Trees

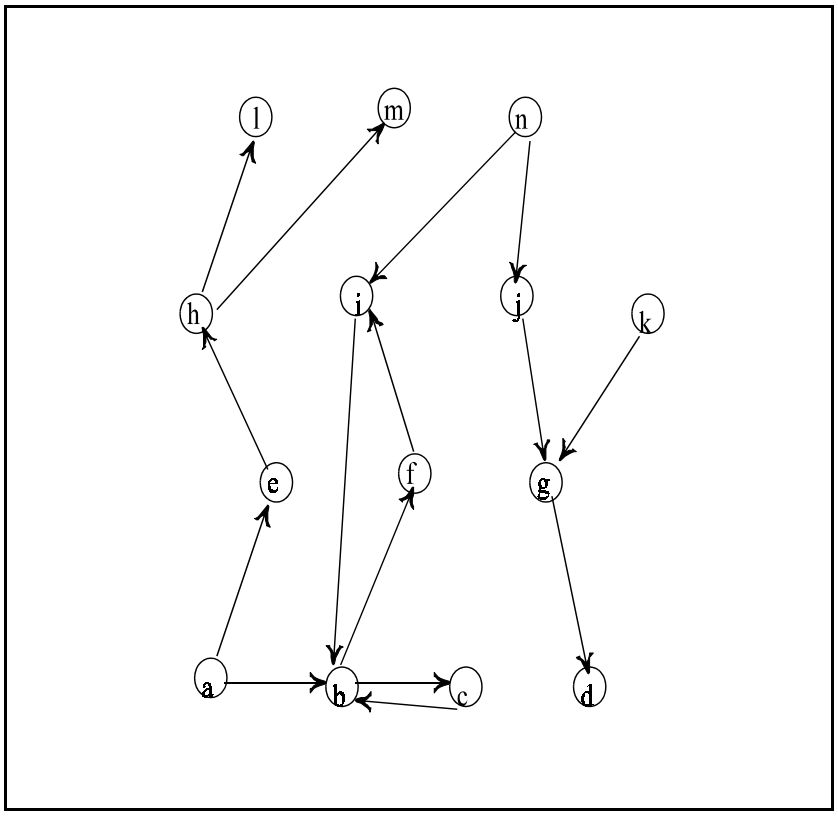
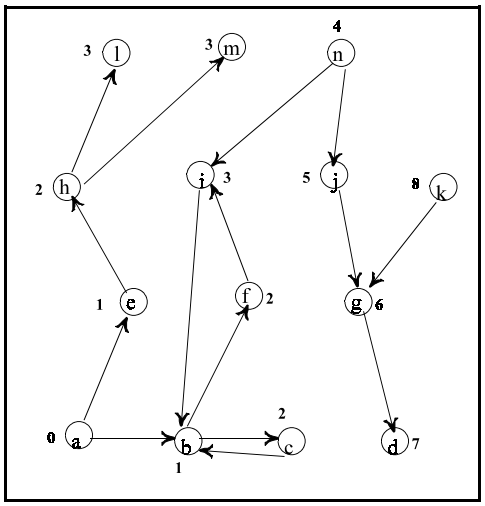


Figure 12 Do BFS and DFS

☐ **Exercise 2** Do DFS on the graph in the previous exercise.

1. Note there is more than one correct answer.



2. Note there is more than one correct answer.