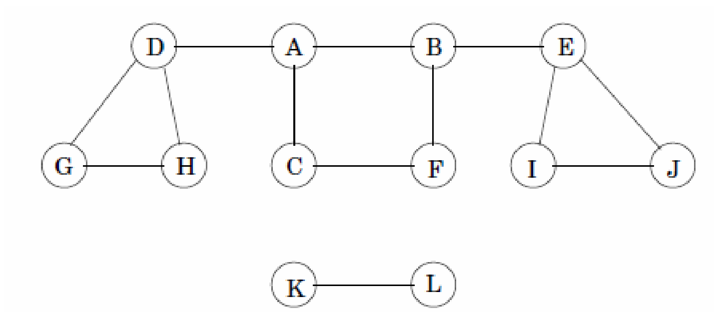
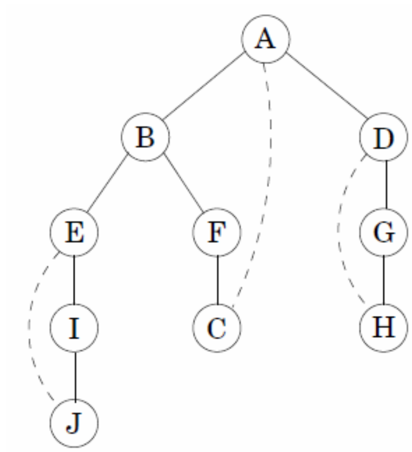
Figure 3.3 Finding all nodes reachable from a particular node.

```
procedure explore(G, v)
Input:    G = (V, E) is a graph; v ∈ V
Output:   visited(u) is set to true for all nodes u reachable from v

visited(v) = true
previsit(v)
for each edge (v, u) ∈ E:
   if not visited(u):  explore(u)
postvisit(v)
```





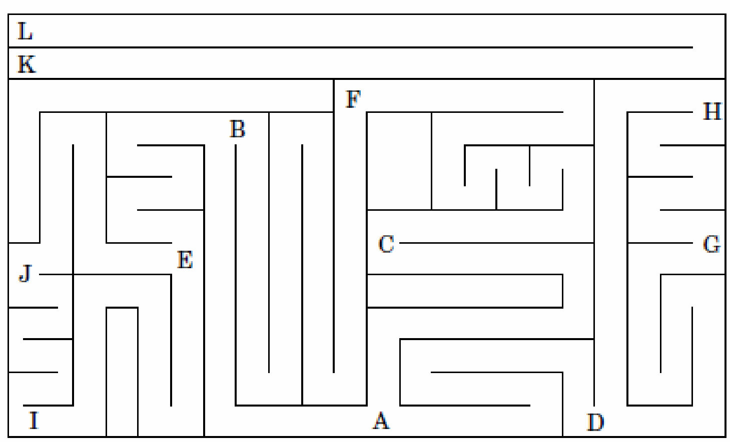
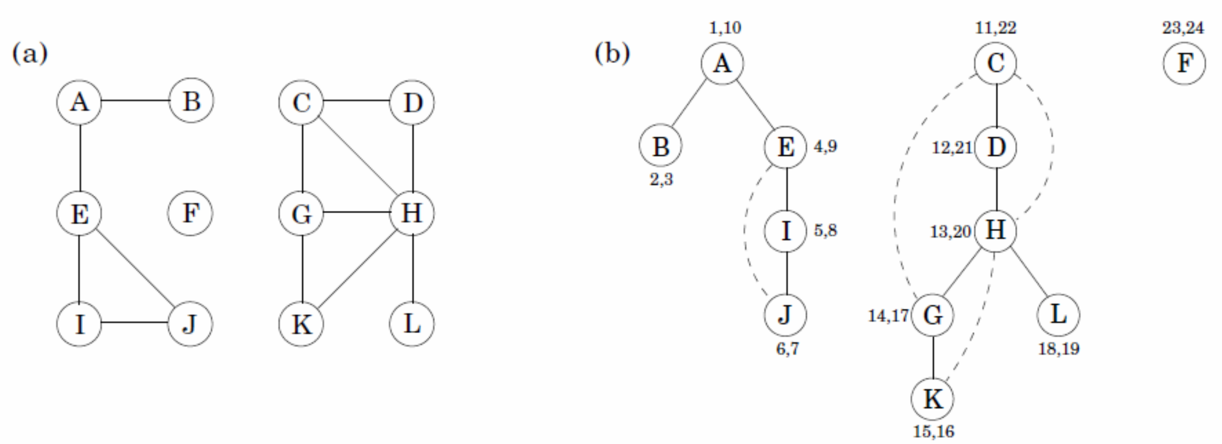


Figure 3.5 Depth-first search.

```
procedure dfs(G)

for all v ∈ V:
   visited(v) = false

for all v ∈ V:
   if not visited(v):  explore(v)
```

Figure 3.6 (a) A 12-node graph. (b) DFS search forest.




```
procedure previsit(v)
pre[v] = clock
clock = clock + 1
```

```
procedure postvisit(v)
post[v] = clock
clock = clock + 1
```

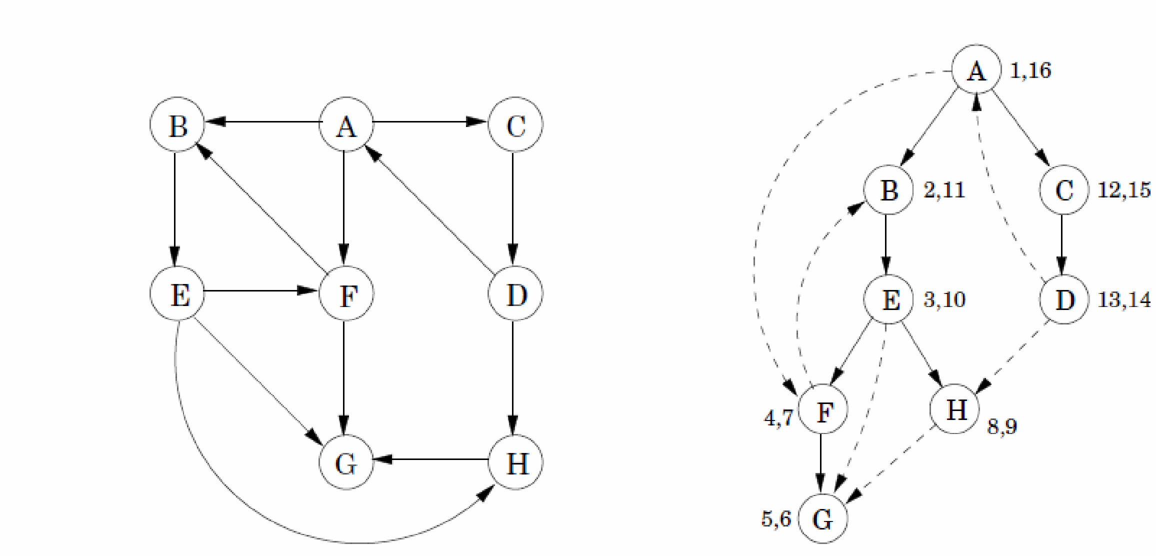
Property *For any nodes* u *and* v*, the two intervals* $[\text{pre}(u), \text{post}(u)]$ *and* $[\text{pre}(v), \text{post}(v)]$ *are either disjoint or one is contained within the other.*

Why? Because $[\text{pre}(u), \text{post}(u)]$ is essentially the time during which vertex u was on the stack. The last-in, first-out behavior of a stack explains the rest.

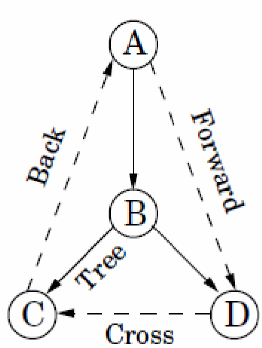
Figure 3.5 Depth-first search.

```
procedure dfs(G)

for all v ∈ V:
   visited(v) = false

for all v ∈ V:
   if not visited(v):   explore(v)
```

Figure 3.7 DFS on a directed graph.



DFS tree



pre/post *ordering for* (u, v)	*Edge type*
$[_u \; [_v \;]_v \;]_u$	Tree/forward
$[_v \; [_u \;]_u \;]_v$	Back
$[_v \;]_v \; [_u \;]_u$	Cross

Fig. 3.7 has two forward edges, two back edges, and two cross edges. Because of DFS exploration strategy, vertex u is an **ancestor** of vertex v when u is discovered first and v is discovered during *explore*(u), i.e., pre(u) < pre(v) < post(v) < post(u).