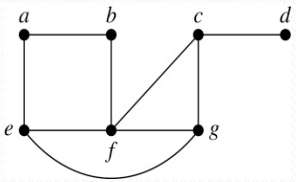
Spanning Trees

Section 11.4

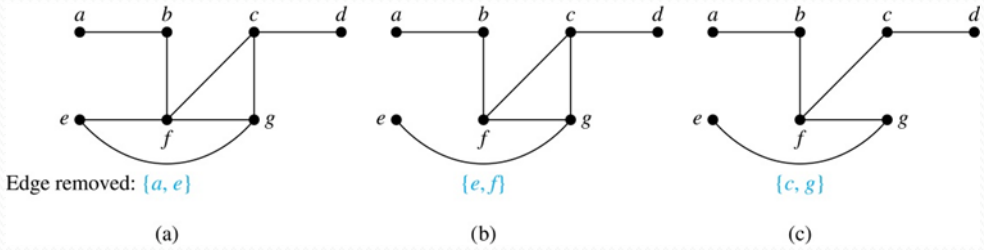
Spanning Trees

Definition: Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.

Example: Find the spanning tree of this graph:



Solution: The graph is connected, but is not a tree because it contains simple circuits. Remove the edge $\{a, e\}$. Now one simple circuit is gone, but the remaining subgraph still has a simple circuit. Remove the edge $\{e, f\}$ and then the edge $\{c, g\}$ to produce a simple graph with no simple circuits. It is a spanning tree, because it contains every vertex of the original graph.



Spanning Trees (*continued*)

Theorem: A simple graph is connected if and only if it has a spanning tree.

Proof: Suppose that a simple graph G has a spanning tree T. T contains every vertex of G and there is a path in T between any two of its vertices. Because T is a subgraph of G, there is a path in G between any two of its vertices. Hence, G is connected.

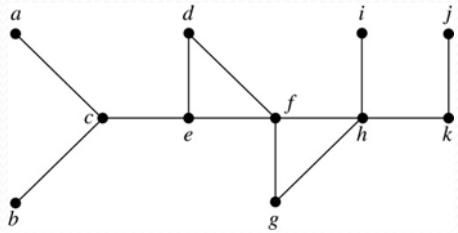
Now suppose that G is connected. If G is not a tree, it contains a simple circuit. Remove an edge from one of the simple circuits. The resulting subgraph is still connected because any vertices connected via a path containing the removed edge are still connected via a path with the remaining part of the simple circuit. Continue in this fashion until there are no more simple circuits. A tree is produced because the graph remains connected as edges are removed. The resulting tree is a spanning tree because it contains every vertex of G.

Depth-First Search

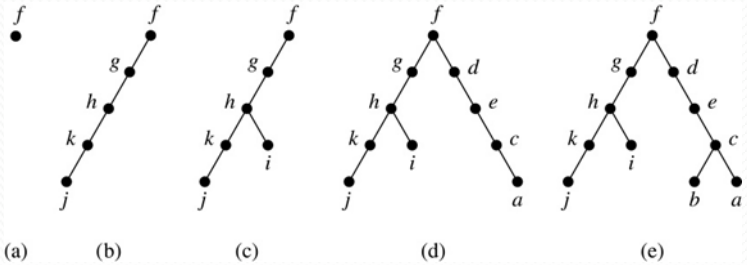
- To use *depth-first search* to build a spanning tree for a connected simple graph first arbitrarily choose a vertex of the graph as the root.
  - Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path. Continue adding vertices and edges to this path as long as possible.
  - If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.
  - Otherwise, move back to the next to the last vertex in the path, and if possible, form a new path starting at this vertex and passing through vertices not already visited. If this cannot be done, move back another vertex in the path.
  - Repeat this procedure until all vertices are included in the spanning tree.

Depth-First Search (*continued*)

Example: Use depth-first search to find a spanning tree of this graph.

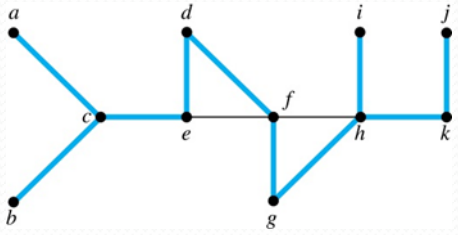


Solution: We start arbitrarily with vertex *f*. We build a path by successively adding an edge that connects the last vertex added to the path and a vertex not already in the path, as long as this is possible. The result is a path that connects *f*, *g*, *h*, *k*, and *j*. Next, we return to *k*, but find no new vertices to add. So, we return to *h* and add the path with one edge that connects *h* and *i*. We next return to *f*, and add the path connecting *f*, *d*, *e*, *c*, and *a*. Finally, we return to *c* and add the path connecting *c* and *b*. We now stop because all vertices have been added.



Depth-First Search (*continued*)

- The edges selected by depth-first search of a graph are called *tree edges*. All other edges of the graph must connect a vertex to an ancestor or descendant of the vertex in the graph. These are called *back edges*.
- In this figure, the tree edges are shown with heavy blue lines. The two thin black edges are back edges.



Depth-First Search Algorithm

- We now use pseudocode to specify depth-first search. In this recursive algorithm, after adding an edge connecting a vertex v to the vertex w, we finish exploring w before we return to v to continue exploring from v.

procedure *DFS*(G: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of the vertex v_1
visit(v_1)

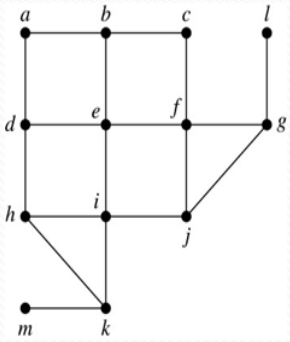
procedure *visit*(v: vertex of G)
for each vertex w adjacent to v and not yet in T
  add vertex w and edge $\{v,w\}$ to T
  visit(w)

Breadth-First Search

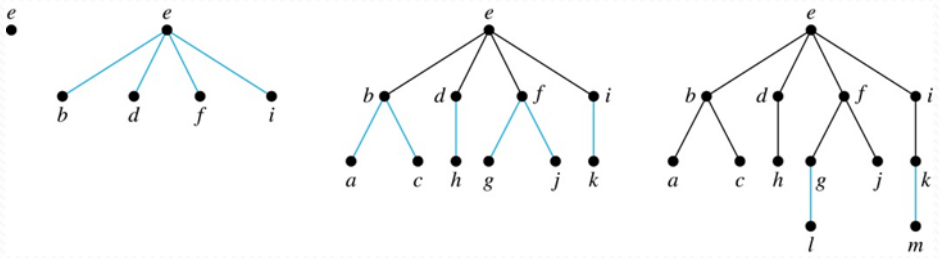
- We can construct a spanning tree using *breadth-first search*. We first arbitrarily choose a root from the vertices of the graph.
  - Then we add all of the edges incident to this vertex and the other endpoint of each of these edges. We say that these are the vertices at level 1.
  - For each vertex added at the previous level, we add each edge incident to this vertex, as long as it does not produce a simple circuit. The new vertices we find are the vertices at the next level.
  - We continue in this manner until all the vertices have been added and we have a spanning tree.

Breadth-First Search (*continued*)

Example: Use breadth-first search to find a spanning tree for this graph.



Solution: We arbitrarily choose vertex *e* as the root. We then add the edges from *e* to *b*, *d*, *f*, and *i*. These four vertices make up level 1 in the tree. Next, we add the edges from *b* to *a* and *c*, the edges from *d* to *h*, the edges from *f* to *j* and *g*, and the edge from *i* to *k*. The endpoints of these edges not at level 1 are at level 2. Next, add edges from these vertices to adjacent vertices not already in the graph. So, we add edges from *g* to *l* and from *k* to *m*. We see that level 3 is made up of the vertices *l* and *m*. This is the last level because there are no new vertices to find.



Breadth-First Search Algorithm

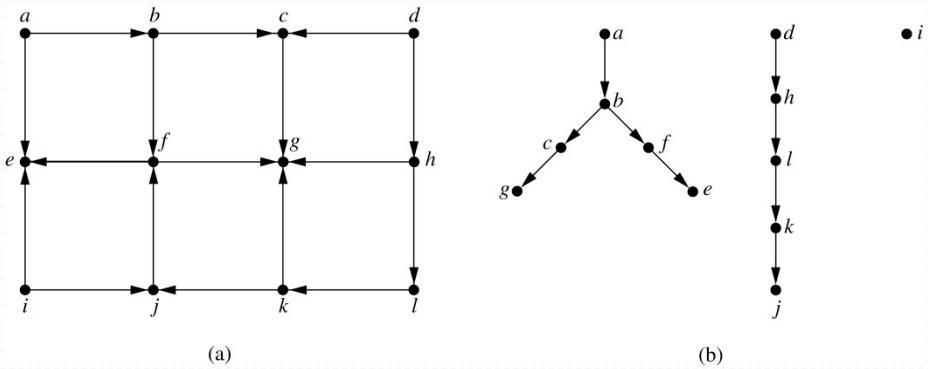
- We now use pseudocode to describe breadth-first search.

procedure *BFS*(*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of the vertex v_1
L := empty list *visit*(v_1)
put v_1 in the list *L* of unprocessed vertices
while *L* is not empty
    remove the first vertex, *v*, from *L*
    for each neighbor *w* of *v*
        if *w* is not in *L* and not in *T* **then**
            add *w* to the end of the list *L*
            add *w* and edge {*v*,*w*} to *T*

Depth-First Search in Directed Graphs

- Both depth-first search and breadth-first search can be easily modified to run on a directed graph. But the result is not necessarily a spanning tree, but rather a spanning forest.

Example: For the graph in (a), if we begin at vertex *a*, depth-first search adds the path connecting *a*, *b*, *c*, and *g*. At *g*, we are blocked, so we return to *b*. Next, we add the path connecting b, *f* and *e*. Next, we return to *a* and find that we cannot add a new path. So, we begin another tree with *d* as its root. We find that this new tree consists of the path connecting the vertices *d*, *h*, *l*, *k*, and *j*. Finally, we add a new tree, which only contains *i*, its root.



- To index websites, search engines such as Google systematically explore the web starting at known sites. The programs that do this exploration are known as *Web spiders*. They may use both breath-first search or depth-first search to explore the Web graph.