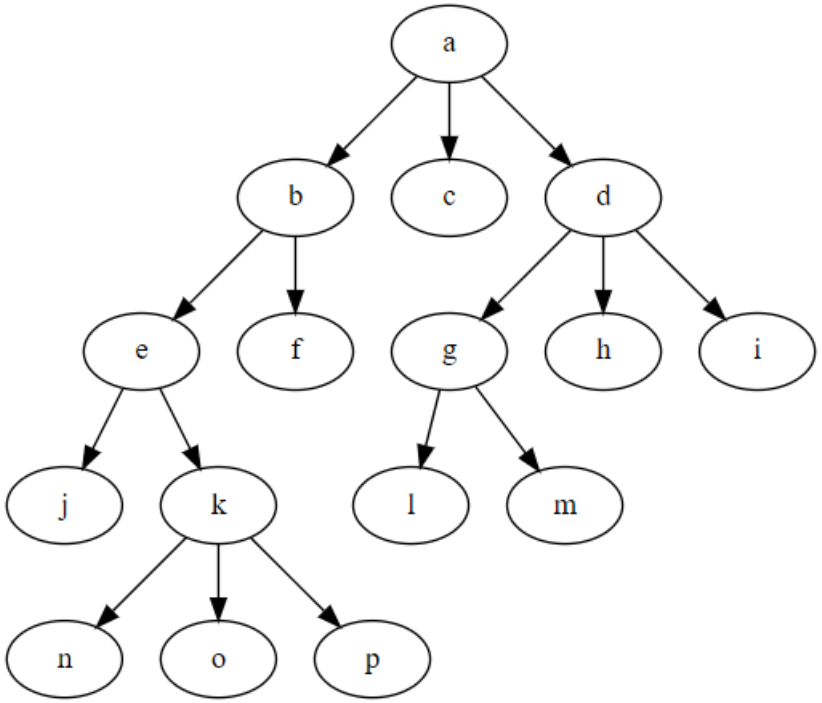
Tree Traversal

Tree Traversal

- Preorder
- Inorder
- Postorder



Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

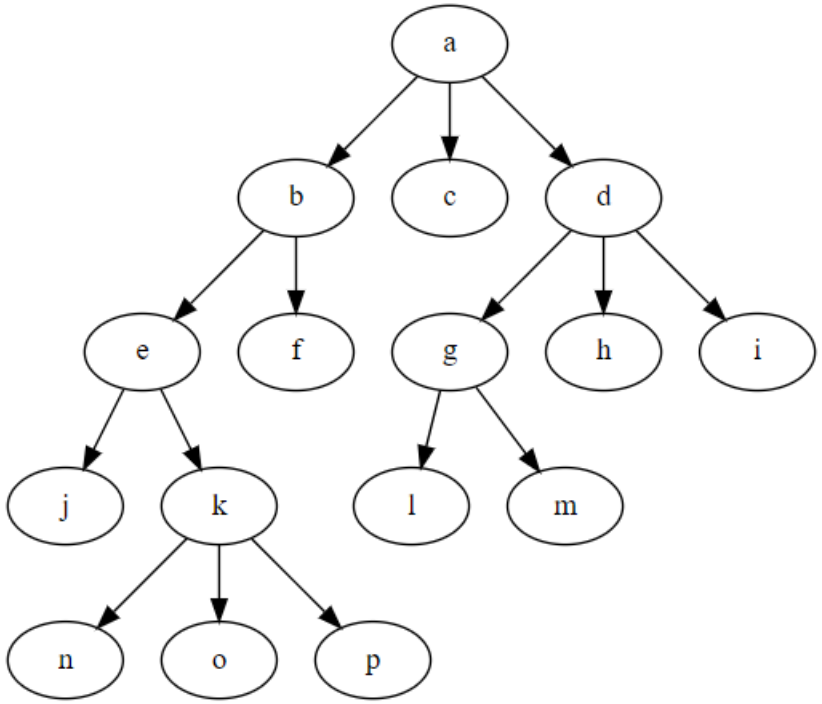
Preorder

If the tree consists only of a root vertex r, then r is the preorder traversal of the tree.

Otherwise, visit r. Then visit each subtree of r starting at the left and do a preorder traversal.

What is the preorder traversal of this tree?

$a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$



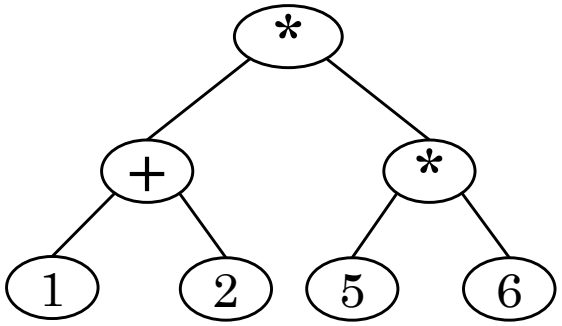
Preorder: Visit root, then visit subtrees left to right

Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

Preorder

Preorder: Visit root, then visit subtrees left to right

What is the preorder traversal of this tree?



* + 1 2 * 5 6

Preorder traversal of a tree representing an expression produces **prefix** or **Polish** notation.

This is also how functions in most programming languages work. For example:

```
add(1, 2)
```

Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

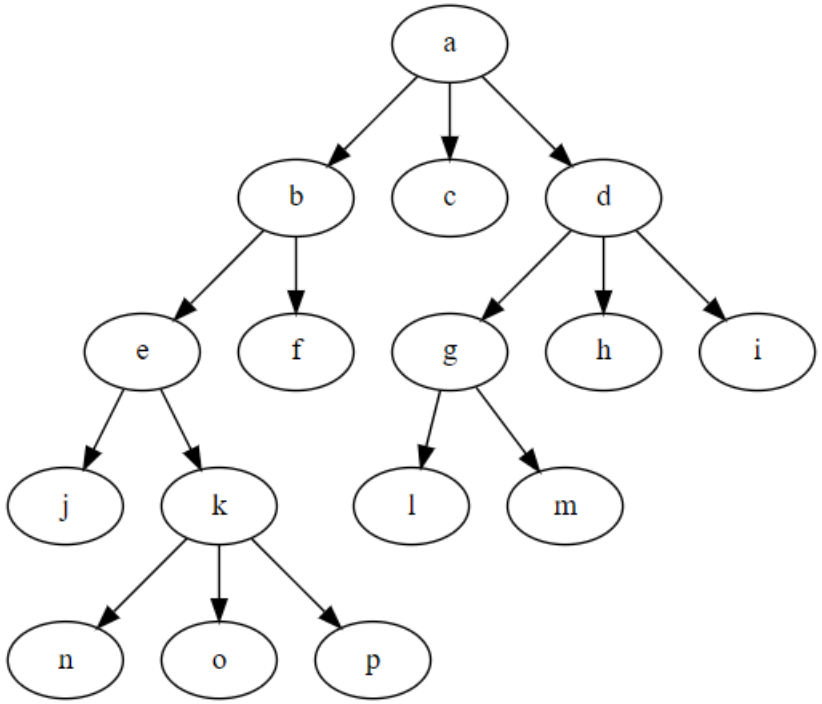
Postorder

If the tree consists only of a root vertex r, then r is the postorder traversal of the tree.

Otherwise, traverse the subtrees in order from left to right, **then** visit r.

What is the postorder traversal of this tree?

$j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a$



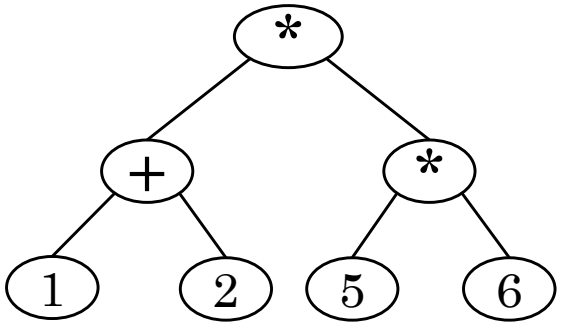
Postorder: Visit subtrees left to right, then visit root

Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

Postorder

Postorder: Visit subtrees left to right, then visit root

What is the postorder traversal of this tree?



1 2 + 5 6 * *

Postorder traversal of a tree representing an expression produces **postfix** or **reverse Polish** notation.

Some calculators use RPN.

Try this online RPN calculator.

Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

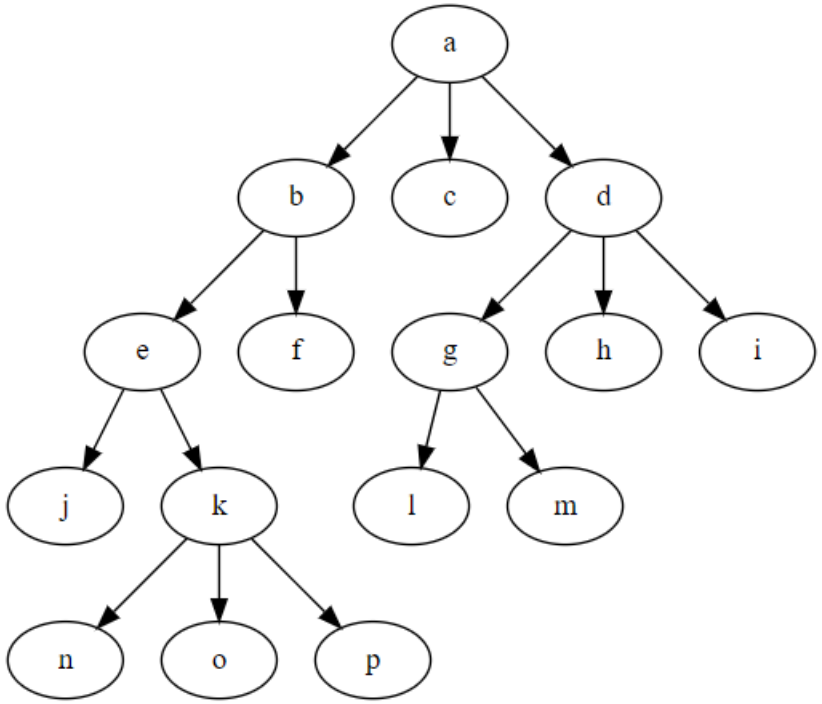
Inorder

If the tree consists only of a root vertex r, then r is the inorder traversal of the tree.

Otherwise, **traverse** the **leftmost** subtree, **then** visit r, then **traverse** the **rest** of the subtrees inorder.

What is the inorder traversal of this tree?

$j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i$



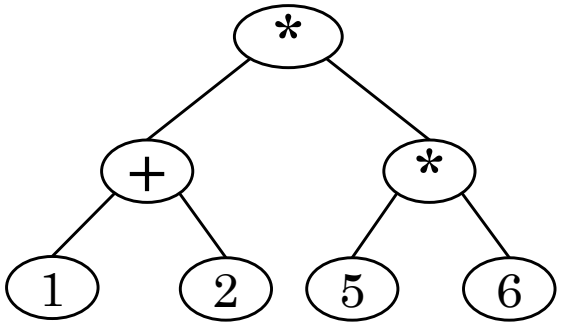
Inorder: Visit leftmost subtree, then visit root, then visit the other subtrees left to right

Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

Inorder

Inorder: Visit leftmost subtree, then visit root, then visit the other subtrees left to right

What is the inorder traversal of this tree?



$1 + 2 * 5 * 6$

Inorder traversal of a tree representing an expression produces **infix** notation.

Note: In order to make inorder traversal expressions unambiguous, we need to include parenthesis whenever we encounter an operation as we traverse the tree inorder.

$((1 + 2) * (5 * 6))$

Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

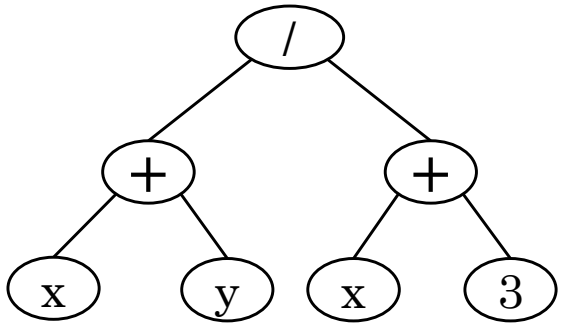
Inorder

These trees would all produce the same expression without parenthesis.

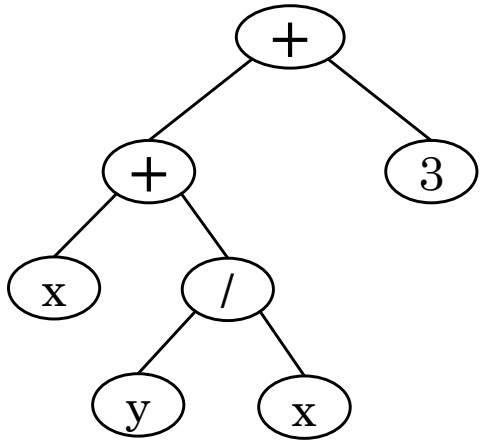
$x + y/x + 3$

Add parenthesis for each operator as we traverse the tree to get:

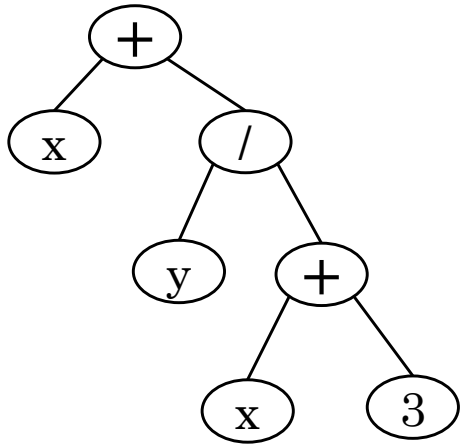
$((x + y) / (x + 3))$



$((x + (y/x)) + 3)$



$(x + (y/(x + 3)))$



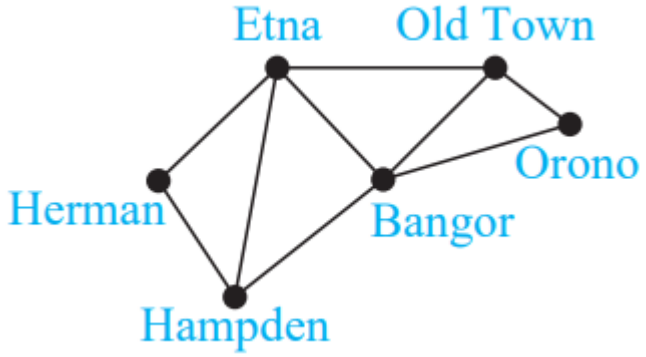
Suggested Additional Exercises:

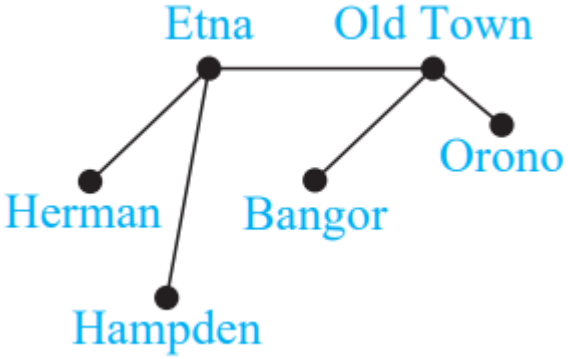
14.4.1

14.4.2

Spanning Trees

- Remember, a tree is a connected graph with no cycles
- If G is a simple graph, a spanning tree of G is a subgraph of G that is a **tree** containing **every vertex of G**
- Finding a spanning tree gives us a simple way to traverse or travel through all the vertices
- **Every connected graph has a spanning tree**

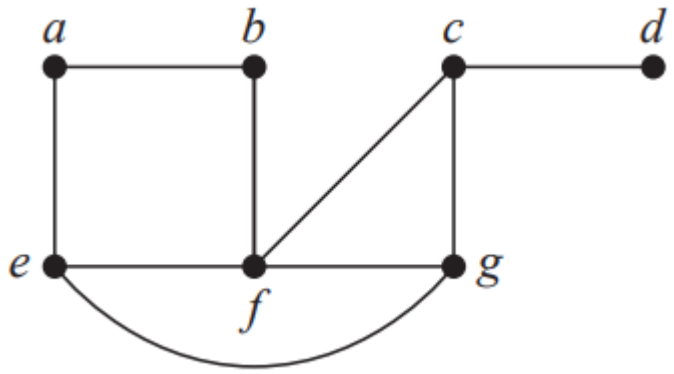


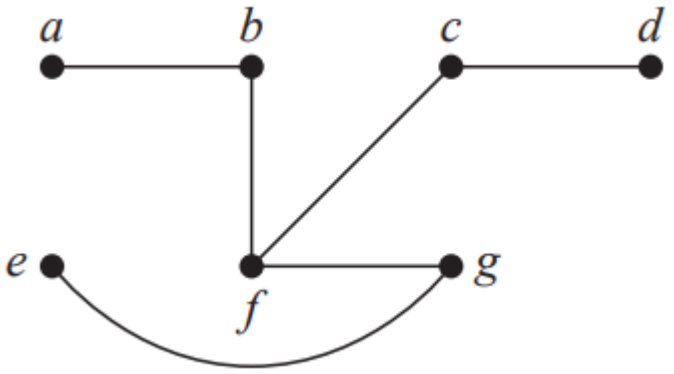


Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

Finding a Spanning Tree

- Remove edges that create a cycle



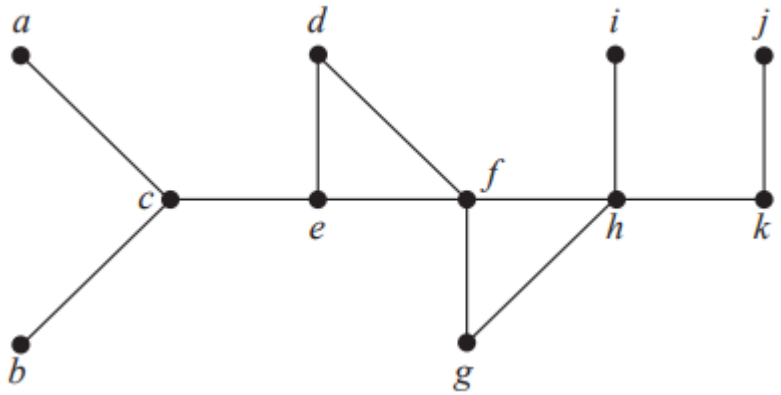


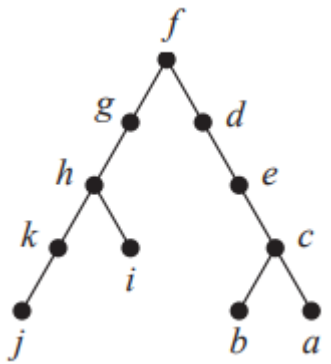
Rosen, Kenneth. *Discrete Mathematics and Its Applications, 7th Edition*. McGraw-Hill, 2012

Depth-First Search

Start with arbitrary vertex. Build a path by successively visiting each edge incident with that vertex that has not already been visited. Continue until no longer possible. Backtrack to the previous vertex and repeat process.

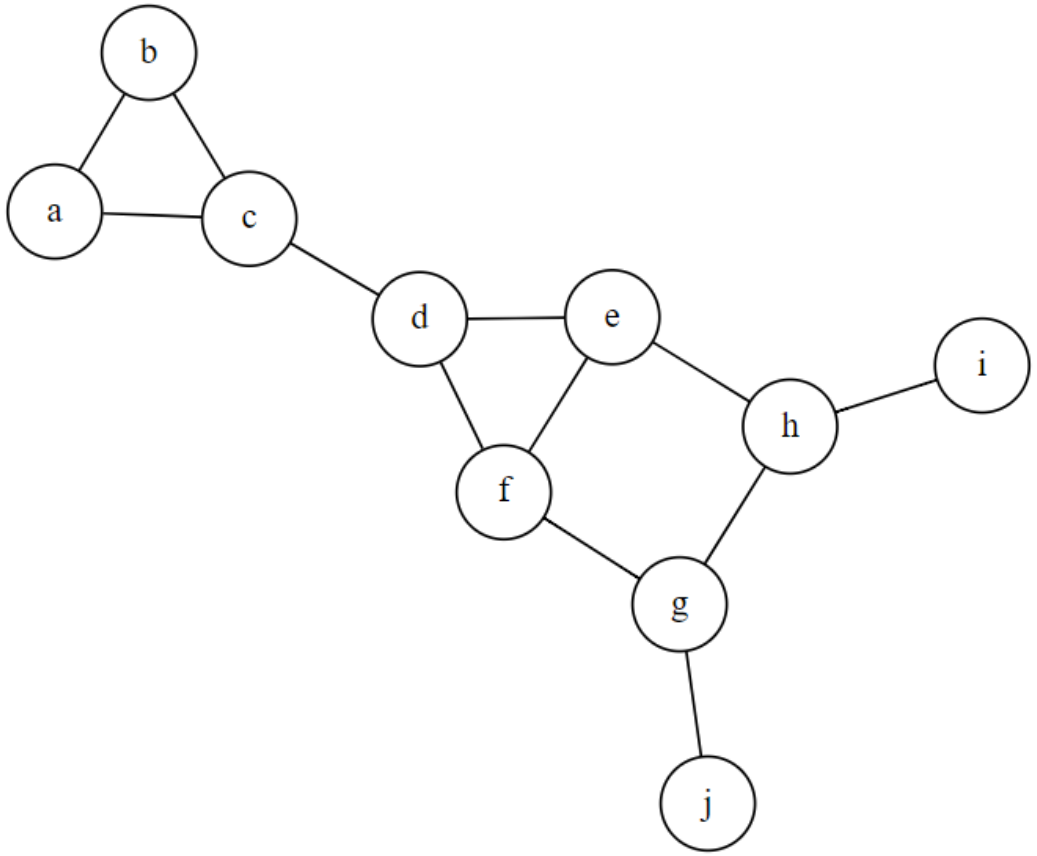
Let's start at f:

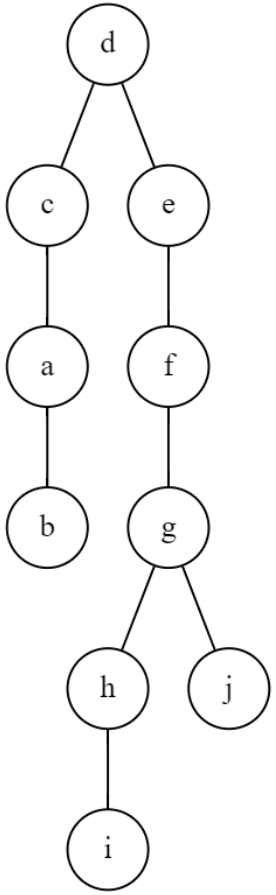




Practice:

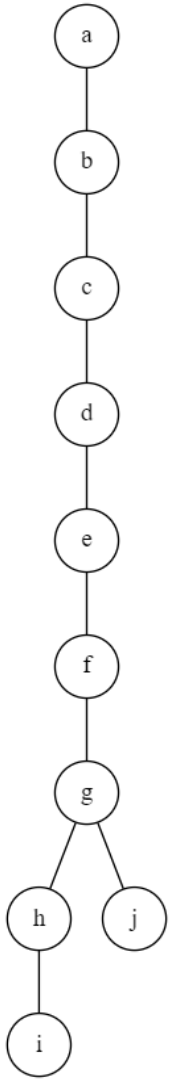
Use depth-first search to find a spanning tree for this graph, starting at "d"





Order starting at "d":

d,c,a,b,e,f,g,h,i,j



Order starting at "a":

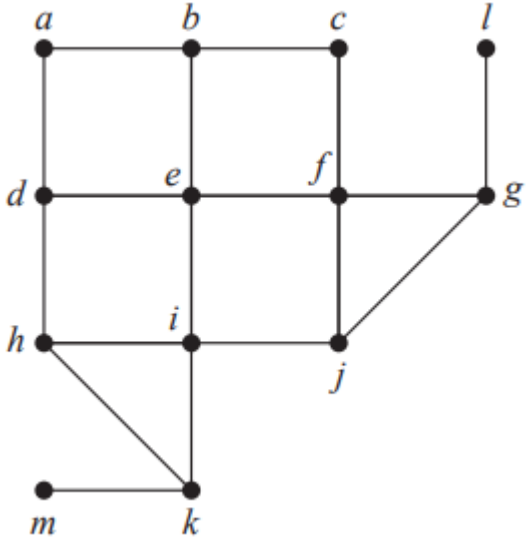
a,b,c,d,e,f,g,h,i,j

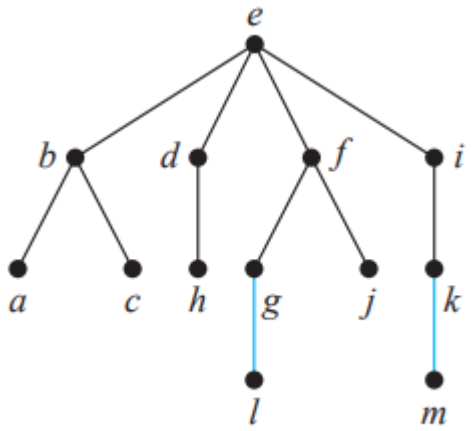
Breadth-first Search

Choose a vertex and add it to the new tree.

Add all edges incident with that vertex. Then for each of those, add each edge incident. Continue until all edges have been added.

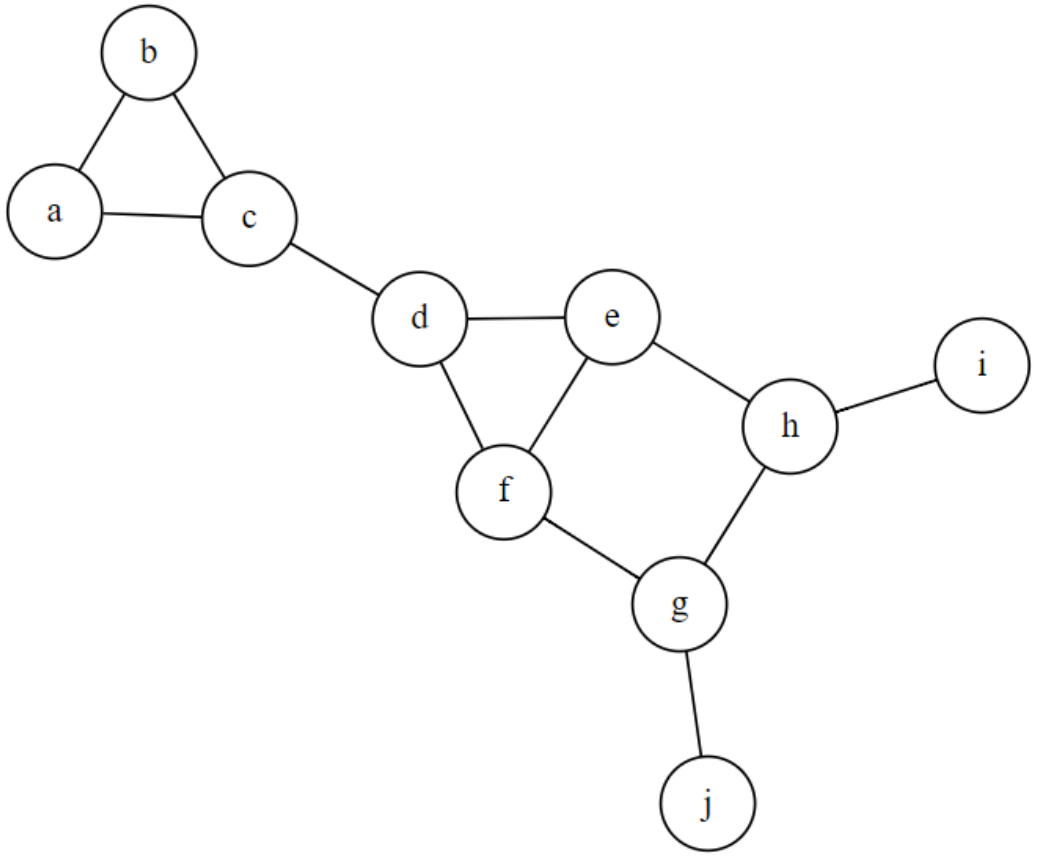
Let's start at *e*:

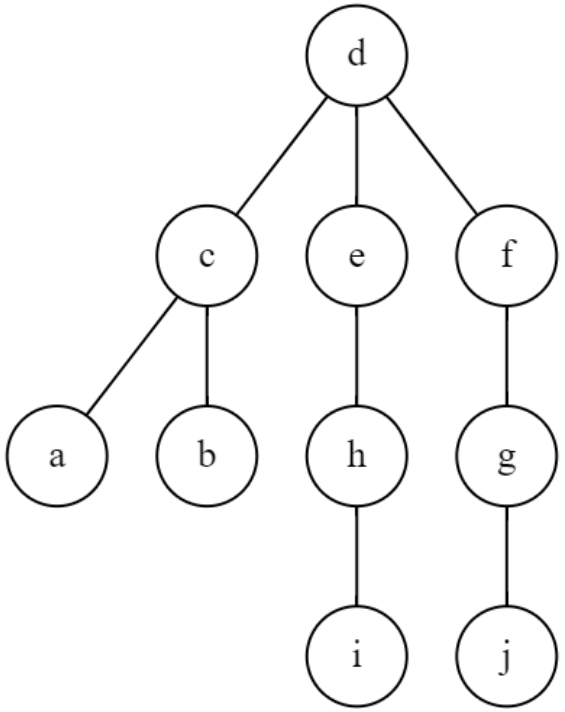




Practice:

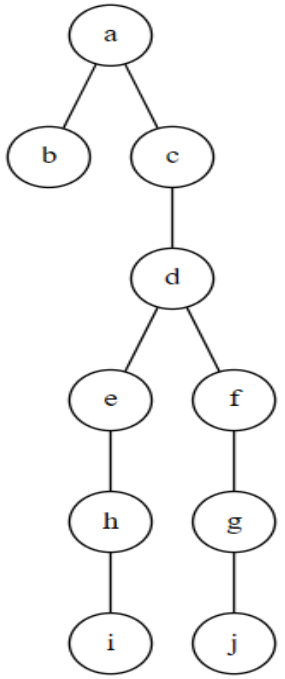
Use breadth-first search to find a spanning tree for this graph, starting at "d"





Order starting at "d":

d,c,e,f,a,b,h,g,i,j



Order starting at "a":

a,b,c,d,e,f,h,g,i,j

Applications for BFS & DFS

What would BFS & DFS look like for these scenarios?

- Family "tree"
  - Finding names
  - Learning about ancestors
- Finding a path through a maze
- Web crawler
- Can you think of any other applications?

Suggested Additional Exercises:

14.5.1

Minimum Spanning Tree

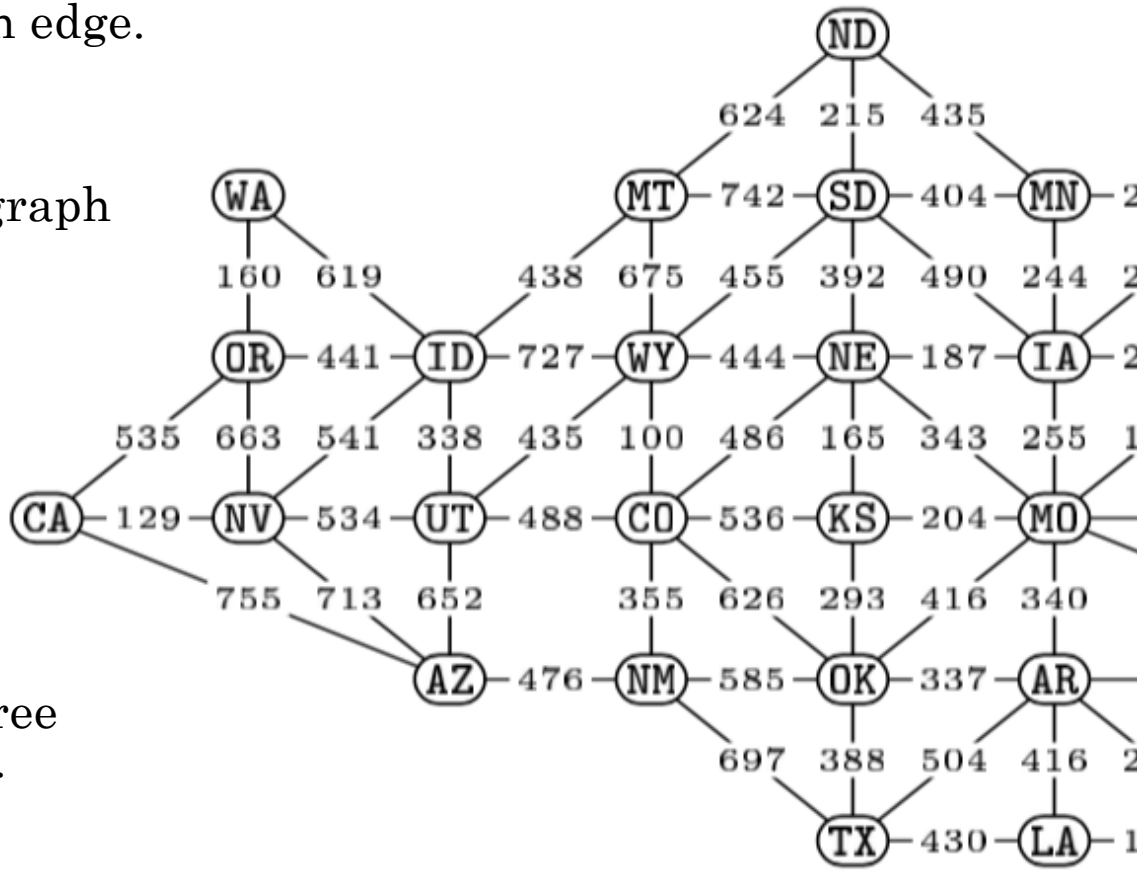
Some graphs are represented with a **cost** or **weight** for each edge.

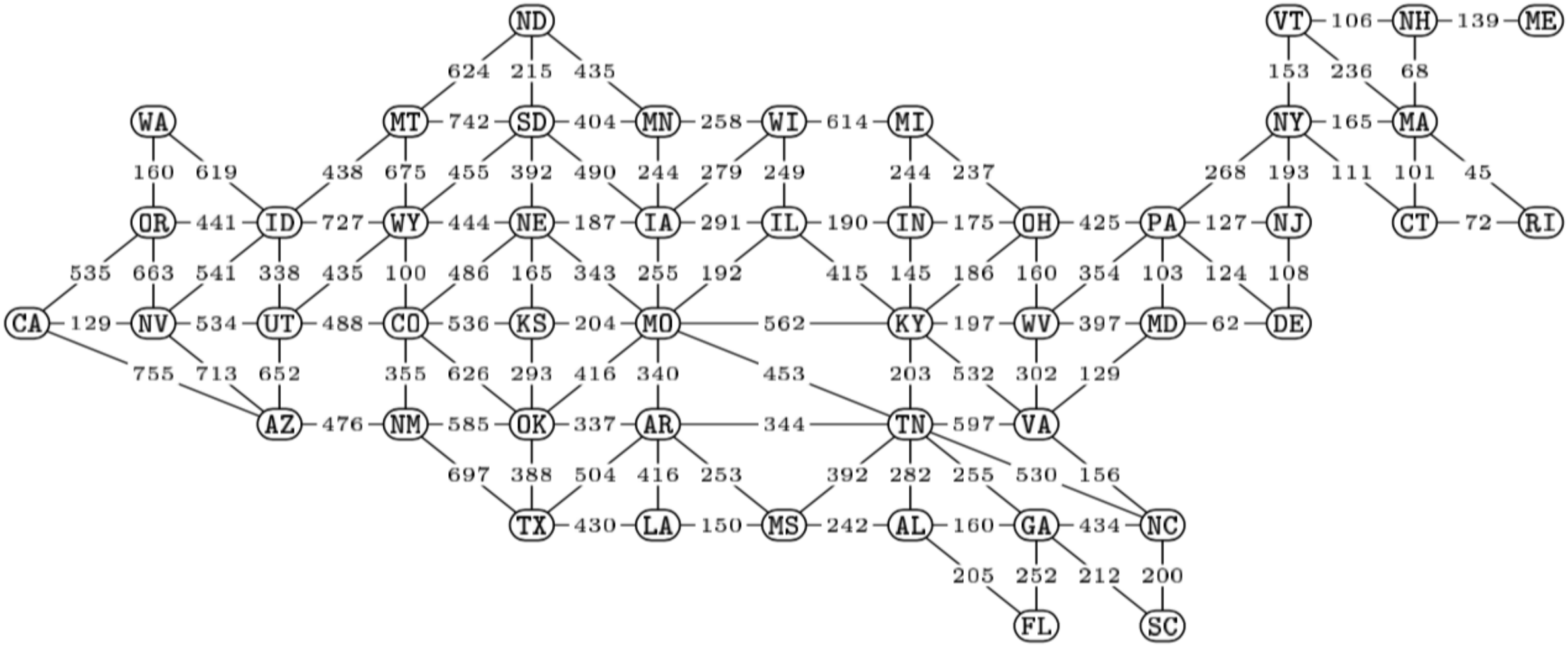
We would like to figure out how to visit every vertex in the graph while minimizing the total cost.

What are some problems this would help to solve?

The **minimum spanning tree** of a graph G is a spanning tree whose weight is no larger than any other spanning tree of G.

There can be more than one minimum spanning tree for a given graph.




ND
624 215 435
VT 106 NH 139 ME
153 236 68
WA MT 742 SD 404 MN 258 WI 614 MI NY 165 MA
160 619 438 675 455 392 490 244 279 249 244 237 268 193 111 101 45
OR 441 ID 727 WY 444 NE 187 IA 291 IL 190 IN 175 OH 425 PA 127 NJ CT 72 RI
535 663 541 338 435 100 486 165 343 255 192 415 145 186 160 354 103 124 108
CA 129 NV 534 UT 488 CO 536 KS 204 MO 562 KY 197 WV 397 MD 62 DE
755 713 652 355 626 293 416 340 453 203 532 302 129
AZ 476 NM 585 OK 337 AR 344 TN 597 VA
697 388 504 416 253 392 282 255 530 156
TX 430 LA 150 MS 242 AL 160 GA 434 NC
205 252 212 200
FL SC

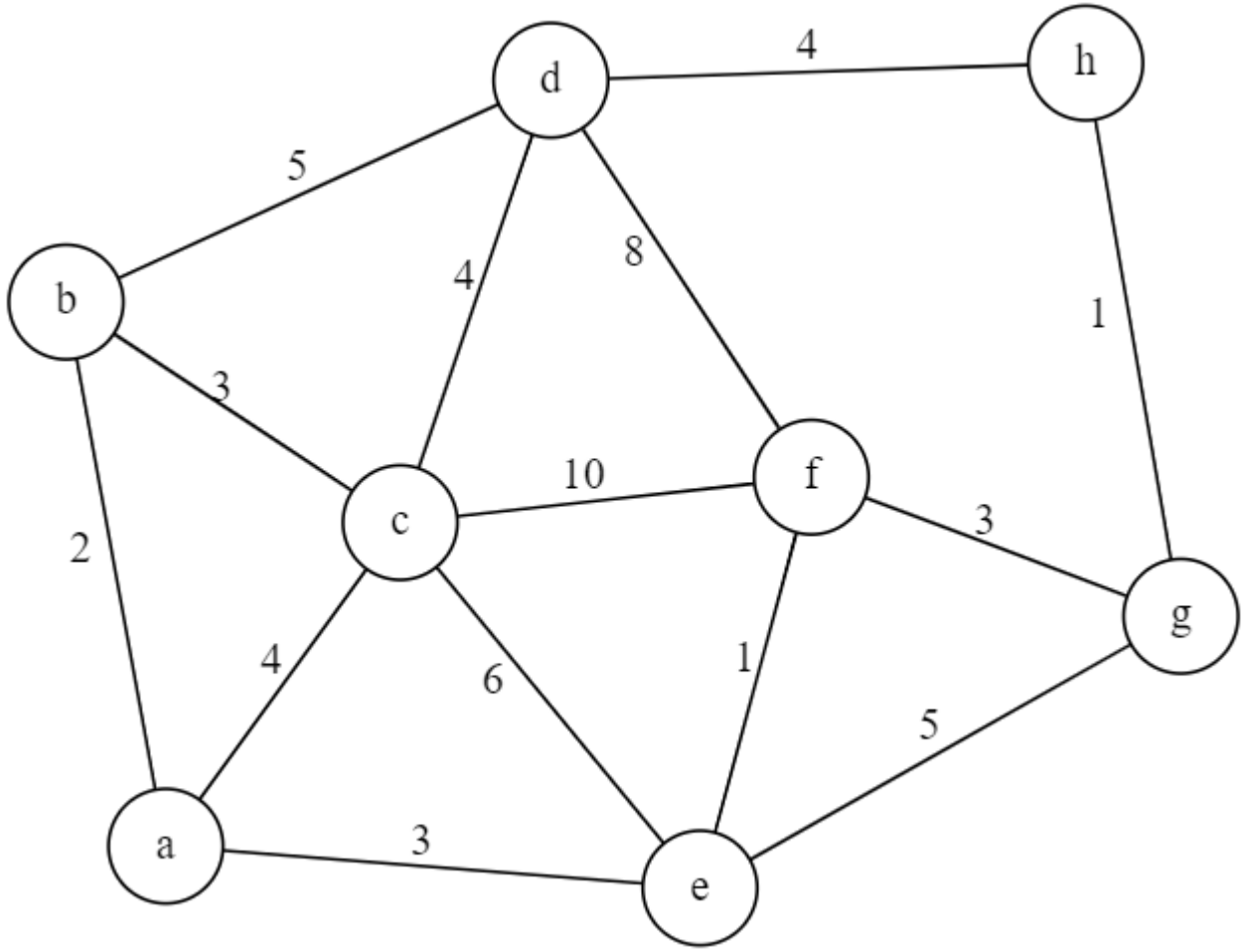
Prim's Algorithm

Created in 1957 by Robert Prim.

1. Pick a starting vertex in the graph and add it to the tree.

2. For each vertex in the tree, find an edge in the graph with the smallest cost that leads to a vertex not yet in the tree. Add the new vertex to the tree.

3. Continue with step 2 until every vertex in the graph has been added to the tree.

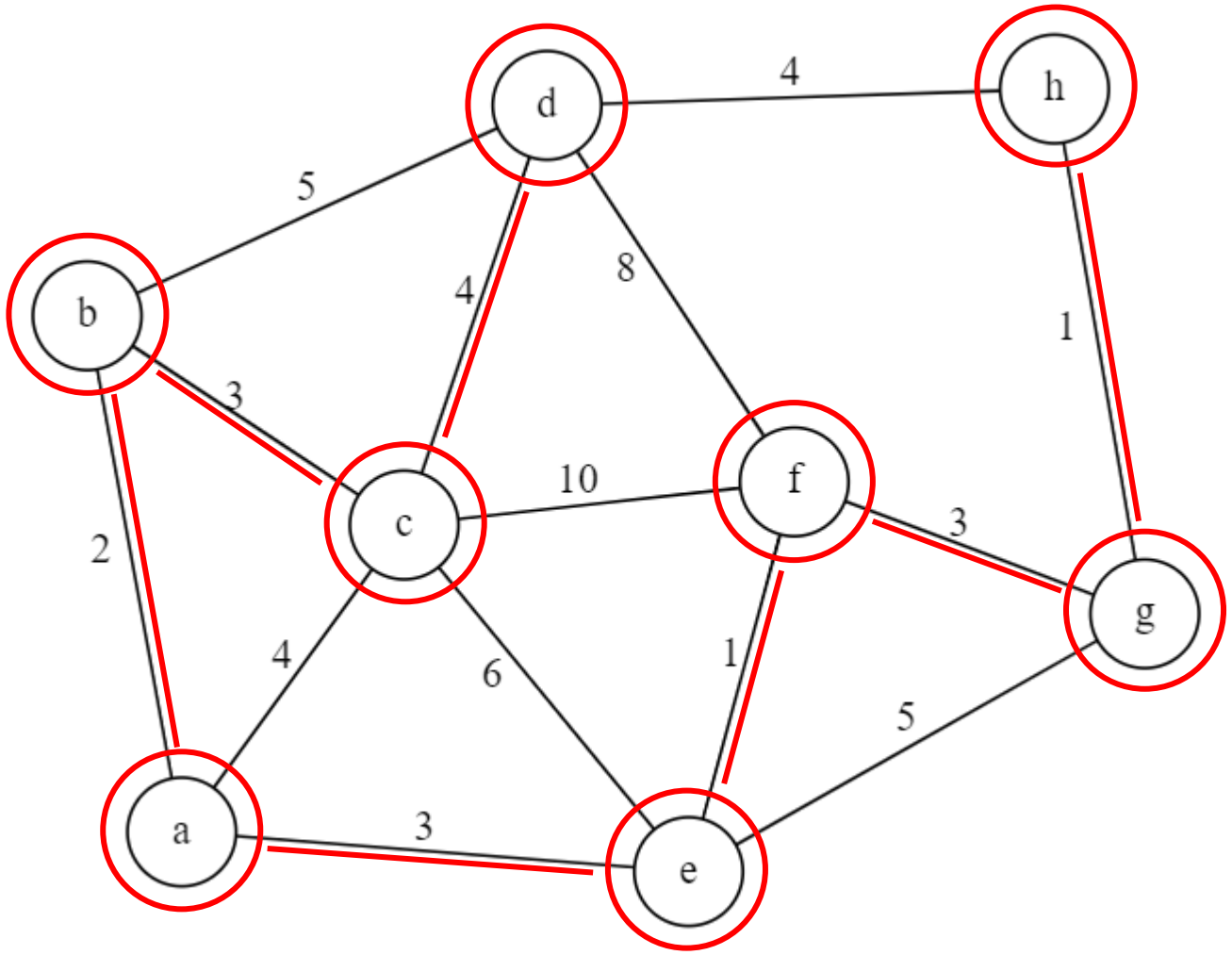


Prim's Algorithm

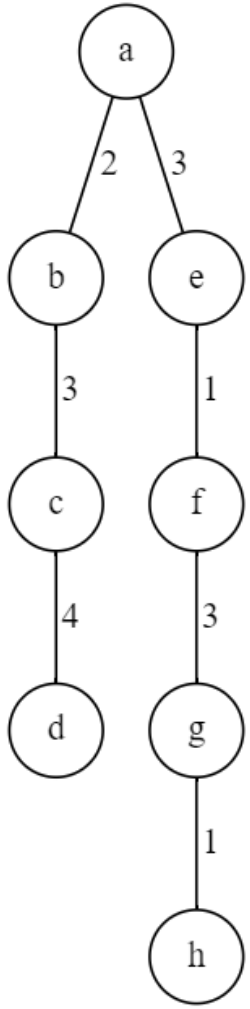
1. Pick a starting vertex in the graph and add it to the tree.

2. For each vertex in the tree, find an edge in the graph with the smallest cost that leads to a vertex not yet in the tree. Add the new vertex to the tree.

3. Continue with step 2 until every vertex in the graph has been added to the tree.

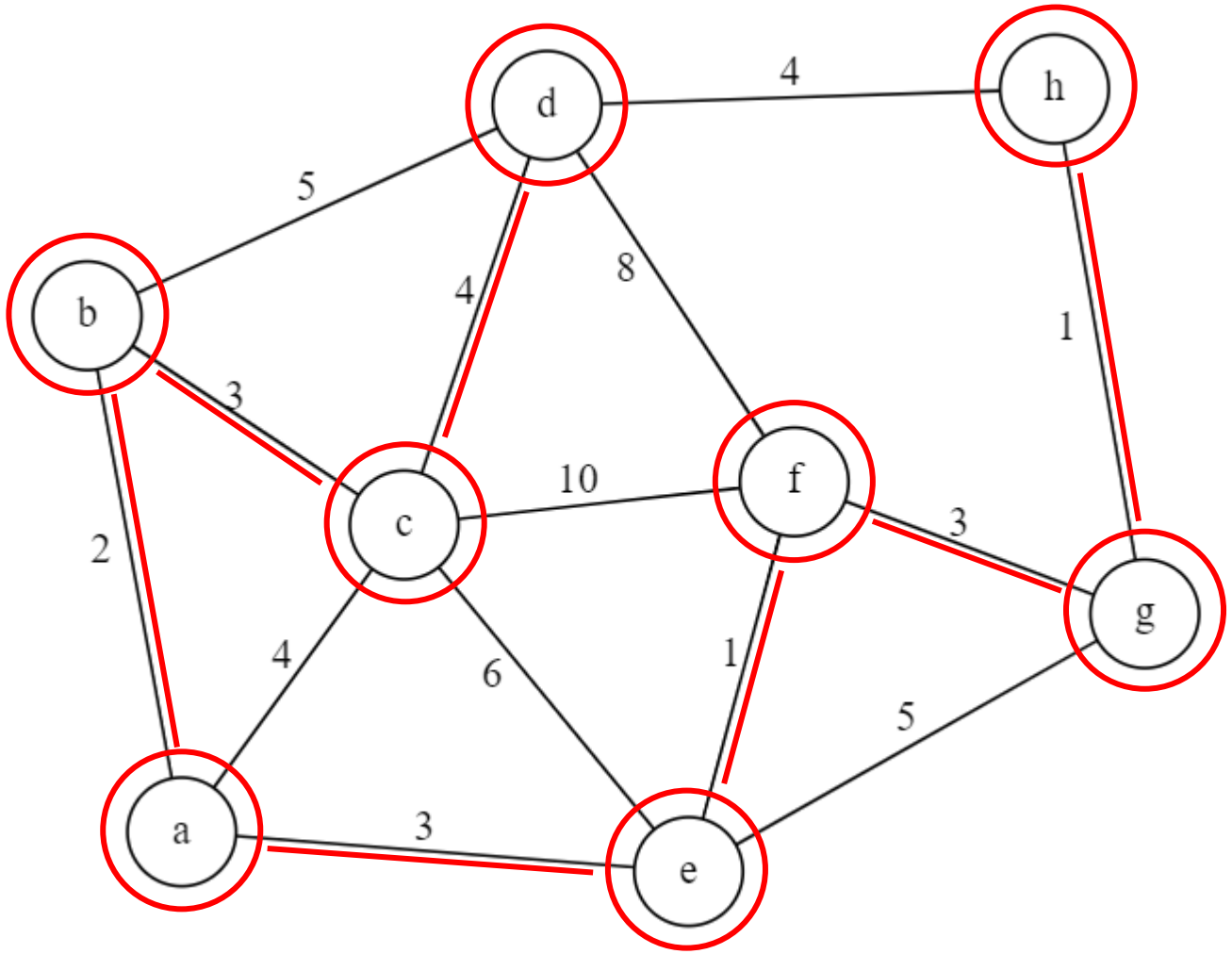


Prim's Algorithm



Total cost: 17

Order:
(a,b), (b,c), (a,e), (e,f), (f,g), (g,h), (c,d)



Suggested Additional Exercises:

14.6.1

14.6.2