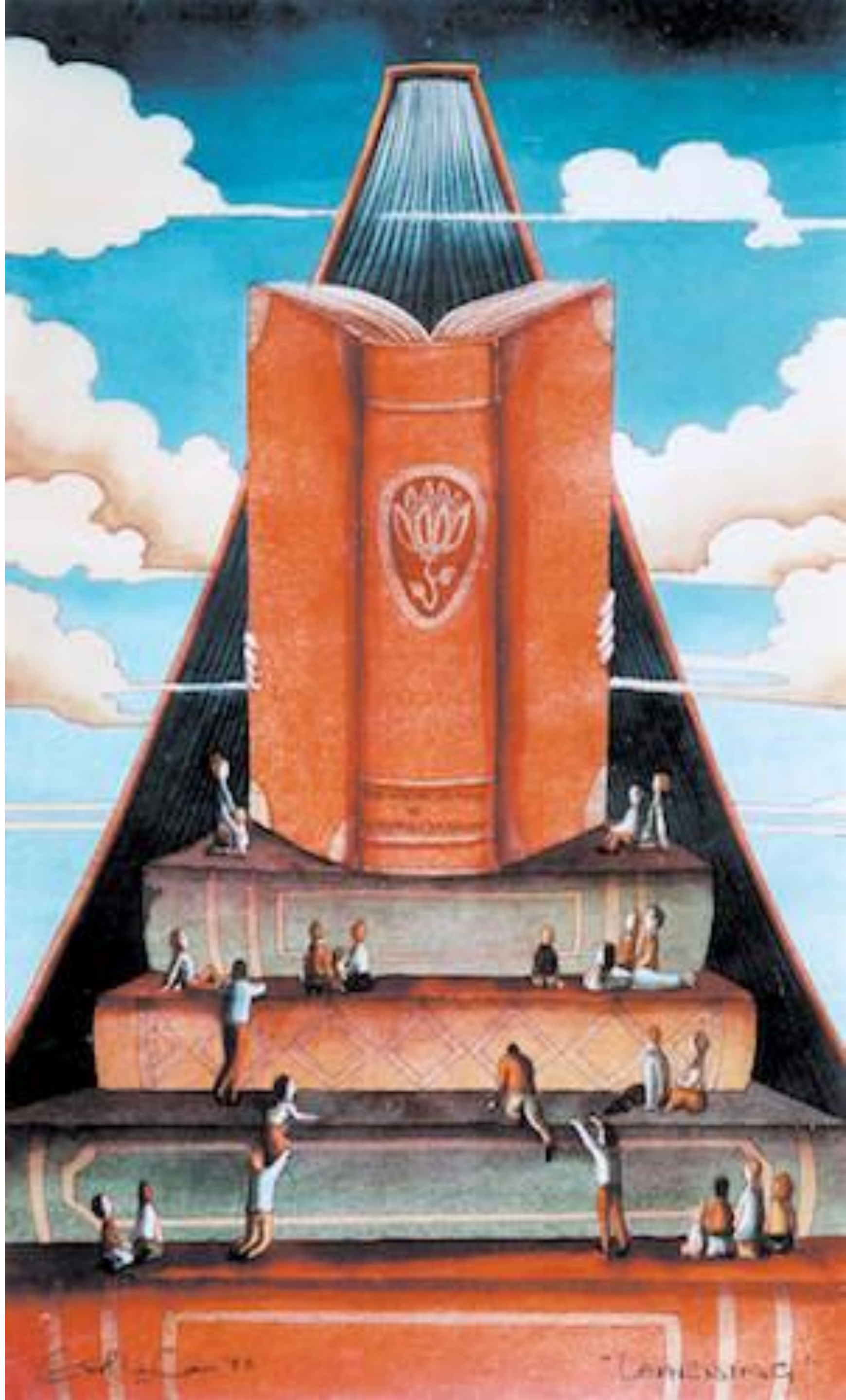
BASIC TRAVERSAL AND SEARCH TECHNIQUES



Outlines

- Introduction
- Binary Tree Traversal Methods
  - Preorder
  - Inorder
  - Postorder
- Graph Search & Traversal Methods
  - Breadth First
  - Depth First

Introduction

- In a traversal of a binary tree, each element of the binary tree is visited exactly once.

- In case of search of a graph (include tree and binary tree), we may not examine all the vertices

- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

Binary Tree Traversal Methods

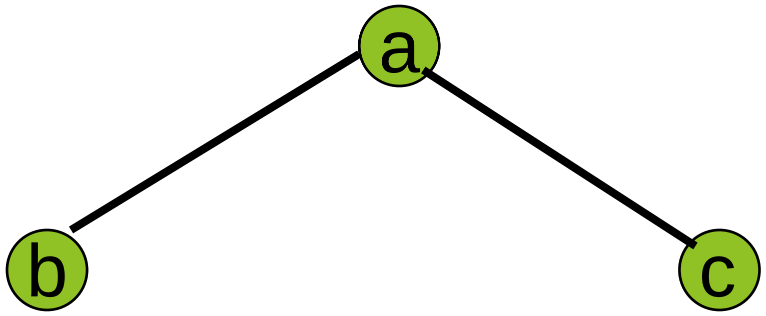
- Preorder
- Inorder
- Postorder

Preorder Traversal

```
treenode = record
{
    Type data; // Type is the data type of data.
    treenode *lchild; treenode *rchild;
}
```

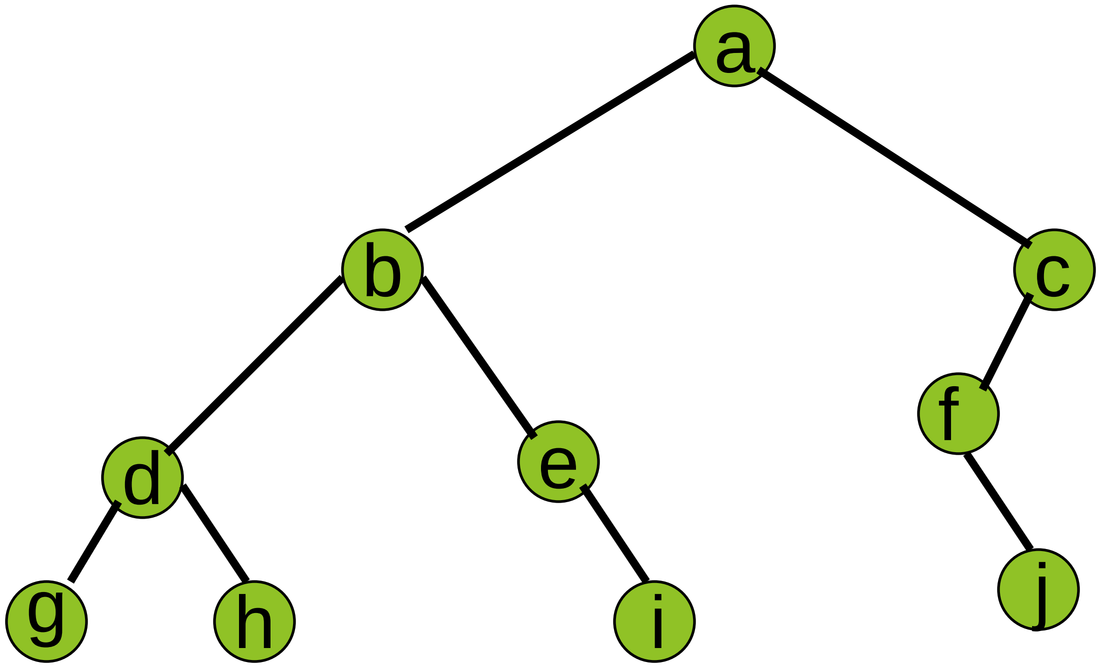
```
Algorithm PreOrder(t)
// t is a binary tree. Each node of t has
// three fields: lchild, data, and rchild.
{
    if t ≠ 0 then
    {
        Visit(t);
        PreOrder(t → lchild);
        PreOrder(t → rchild);
    }
}
```

Preorder Example (visit = print)



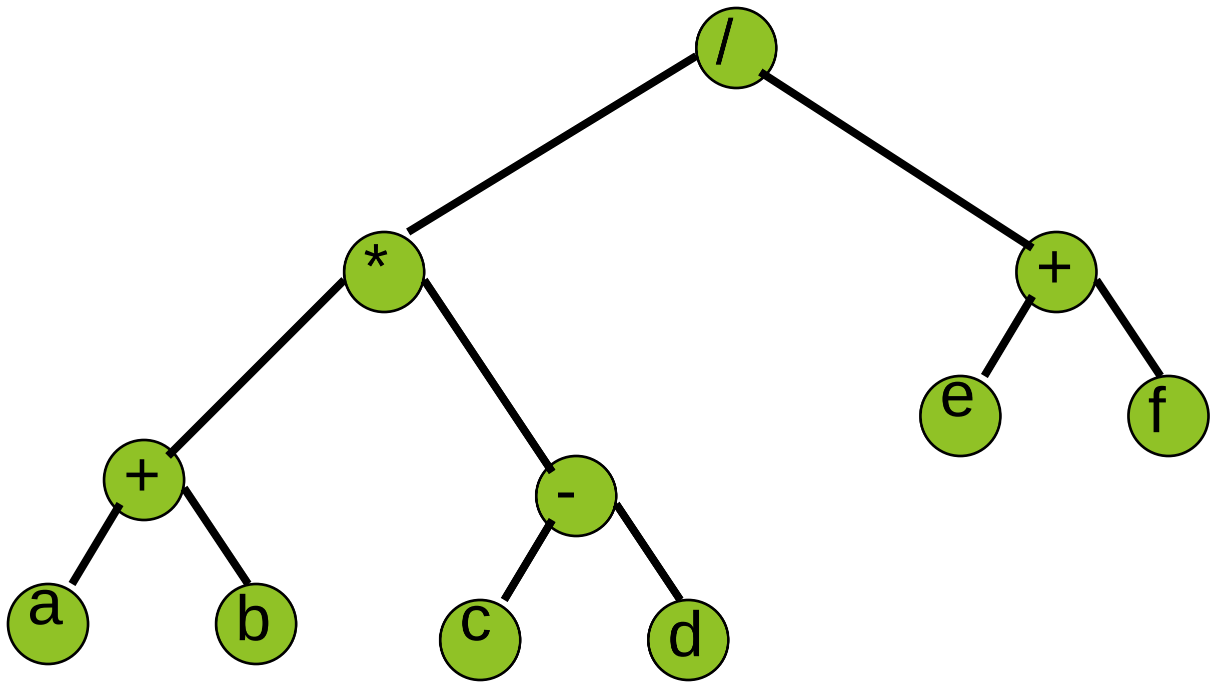
a b c

Preorder Example (visit = print)



a b d g h e i c f j

Preorder Of Expression Tree



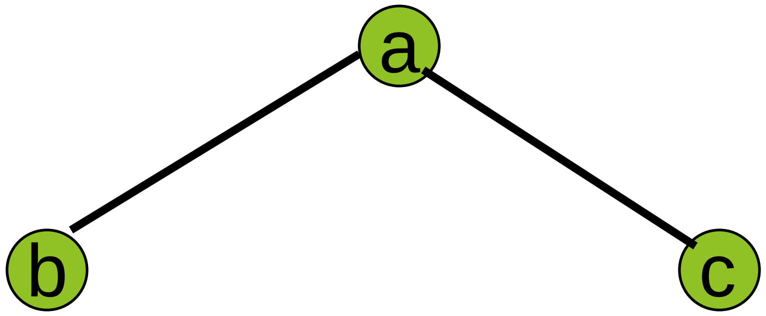
/ * + a b - c d + e f

Gives prefix form of expression!

Inorder Traversal

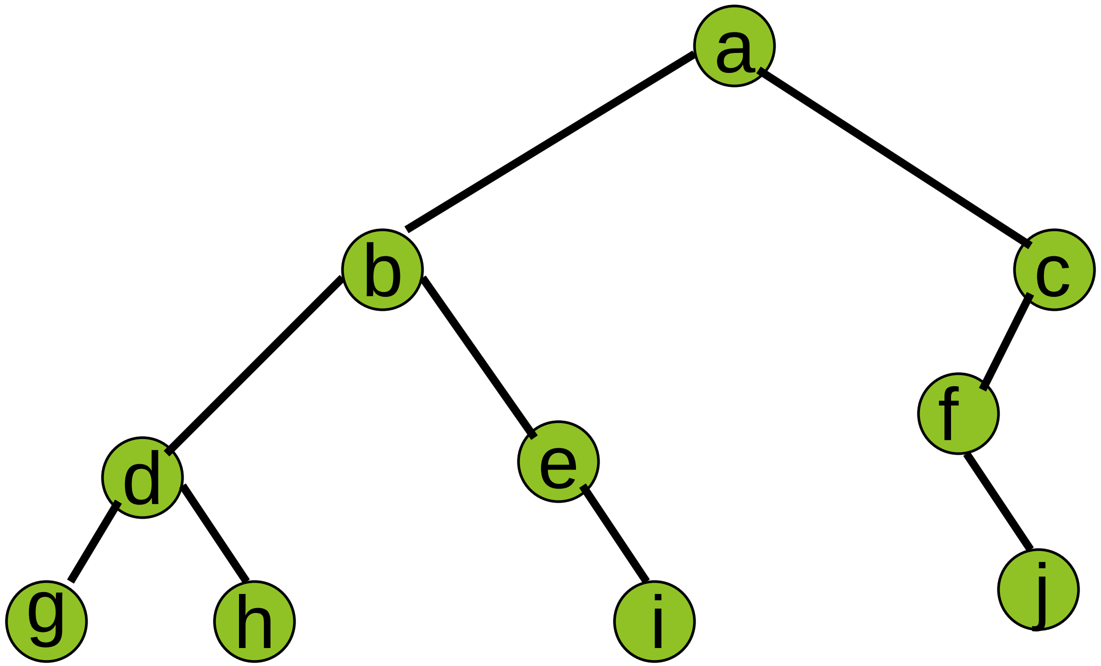
Algorithm InOrder(t)
// t is a binary tree. Each node of t has
// three fields: $lchild$, $data$, and $rchild$.
{
    if $t \neq 0$ **then**
    {
        InOrder($t \rightarrow lchild$);
        Visit(t);
        InOrder($t \rightarrow rchild$);
    }
}

Inorder Example (visit = print)



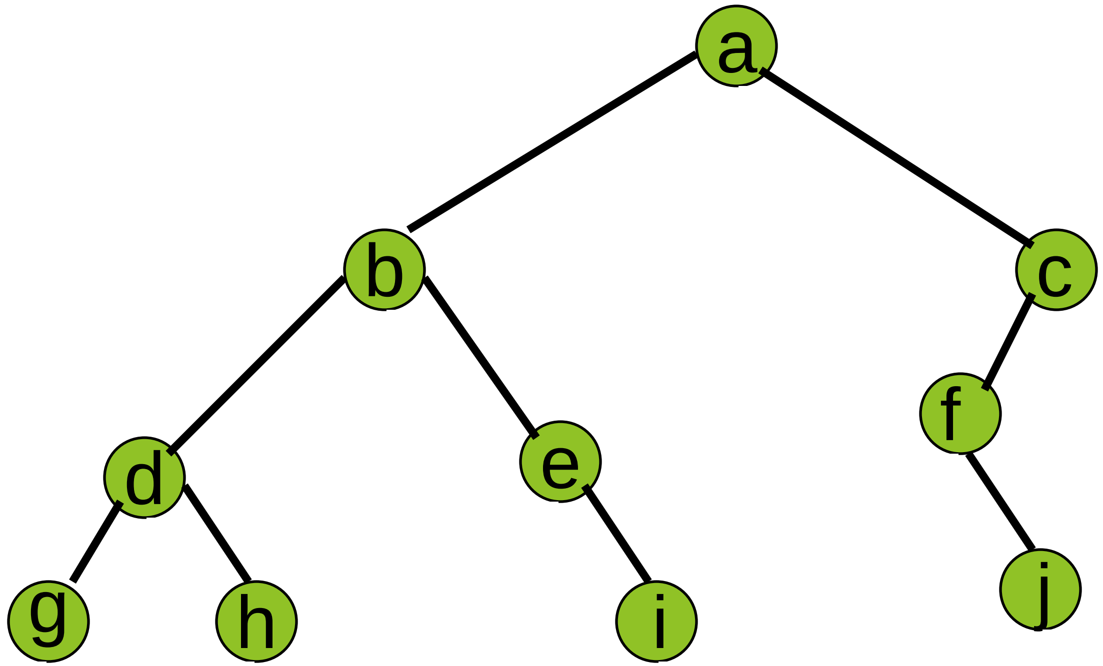
b a c

Inorder Example (visit = print)



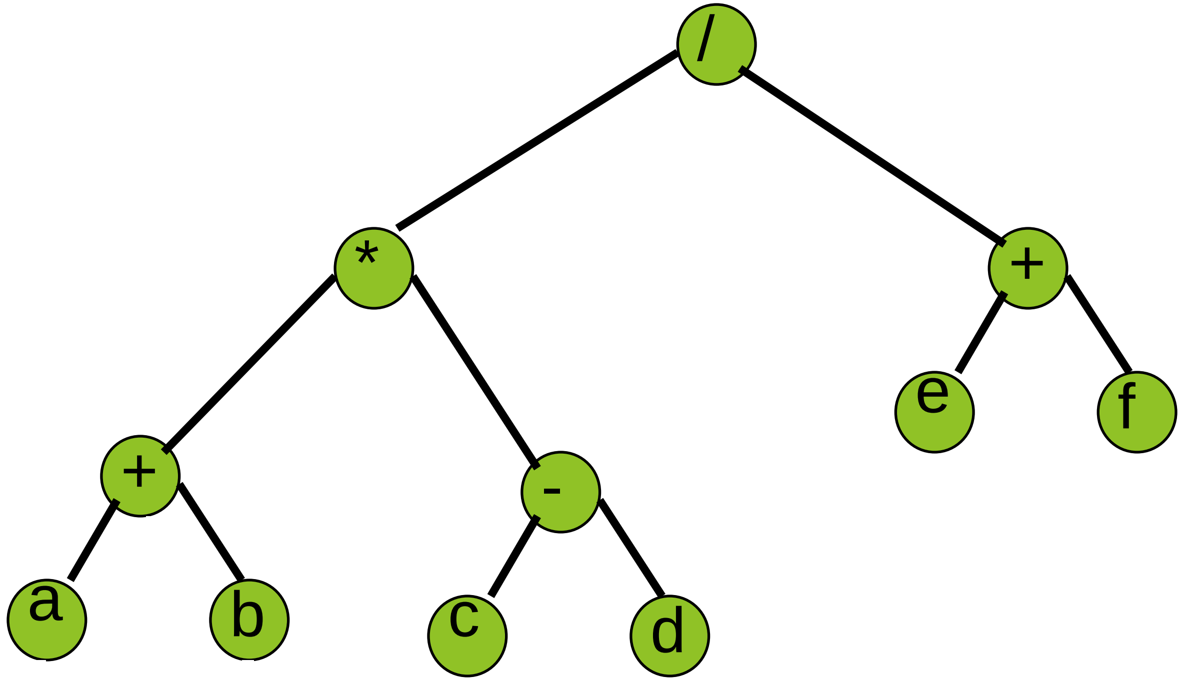
g d h b e i a f j c

Inorder By Projection (Squishing)



g d h b e i a f j c

Inorder Of Expression Tree



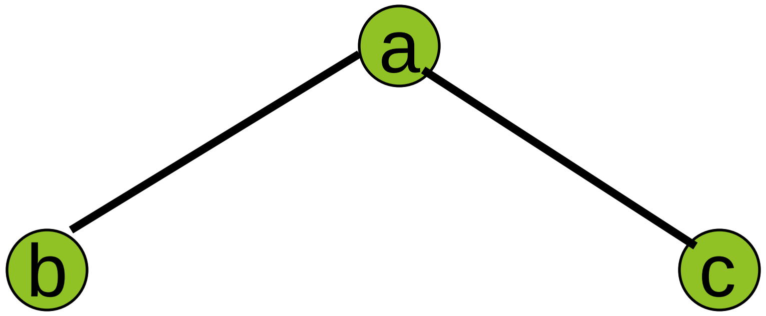
a + b * c - d / e + f

Gives infix form of expression (sans parentheses)!

Postorder Traversal

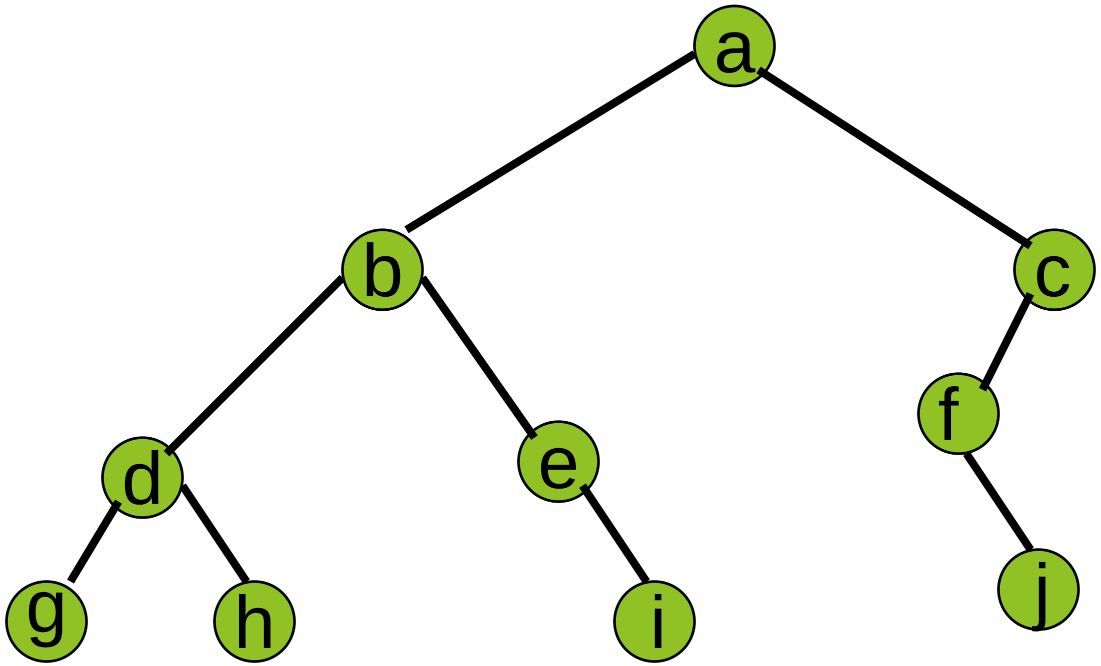
```
Algorithm PostOrder(t)
// t is a binary tree. Each node of t has
// three fields: lchild, data, and rchild.
{
    if t ≠ 0 then
    {
        PostOrder(t → lchild);
        PostOrder(t → rchild);
        Visit(t);
    }
}
```

Postorder Example (visit = print)



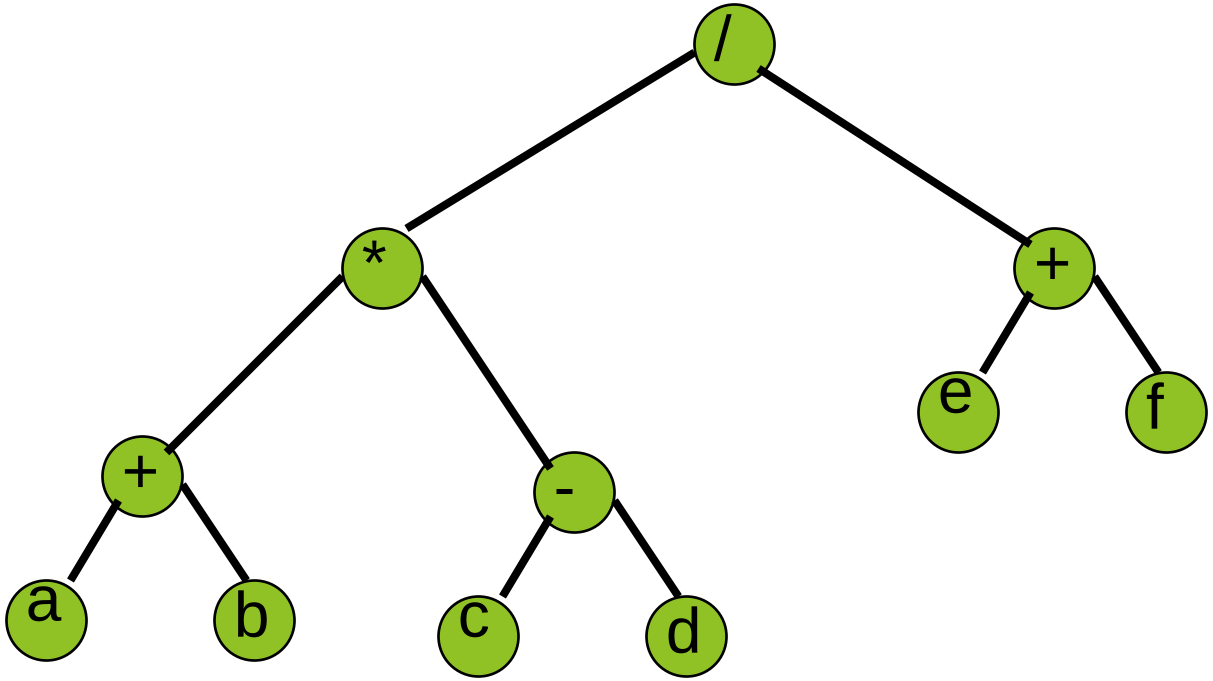
b c a

Postorder Example (visit = print)



g h d i e b j f c a

Postorder Of Expression Tree



a b + c d - * e f + /

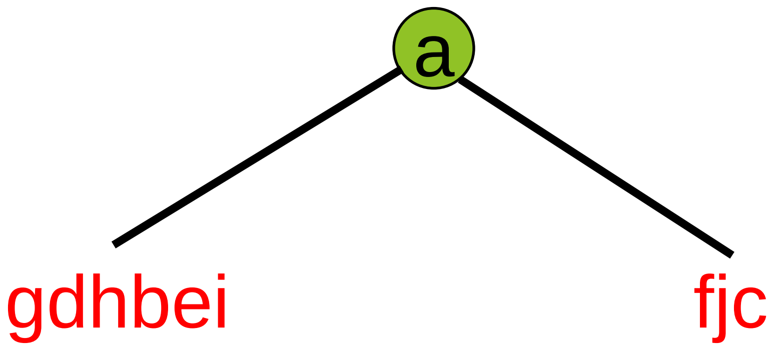
Gives postfix form of expression!

Binary Tree Construction

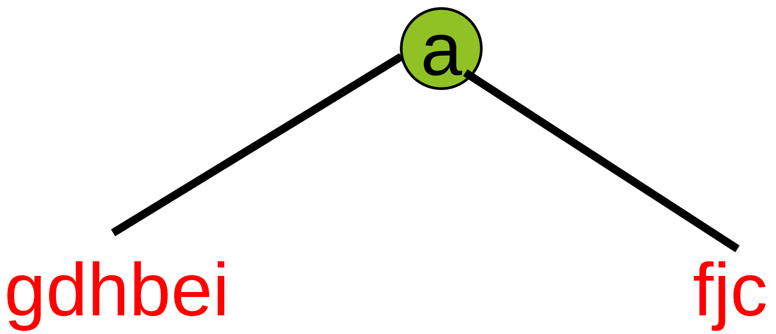
- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Inorder And Preorder

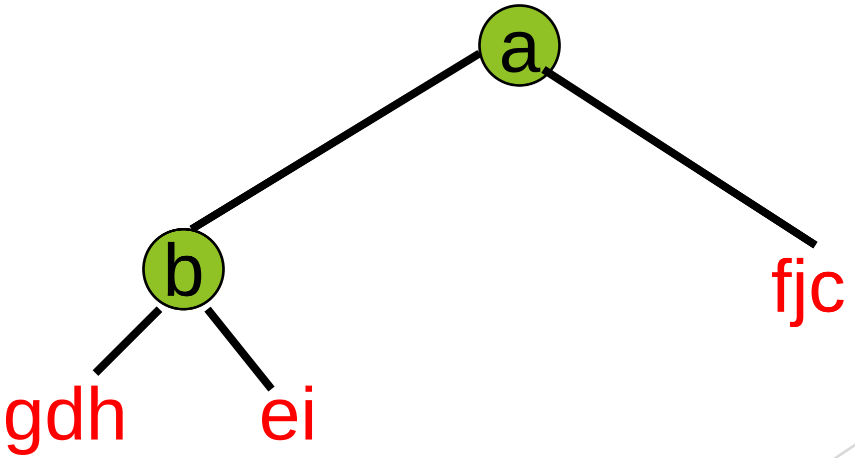
- Inorder = g d h b e i a f j c
- Preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



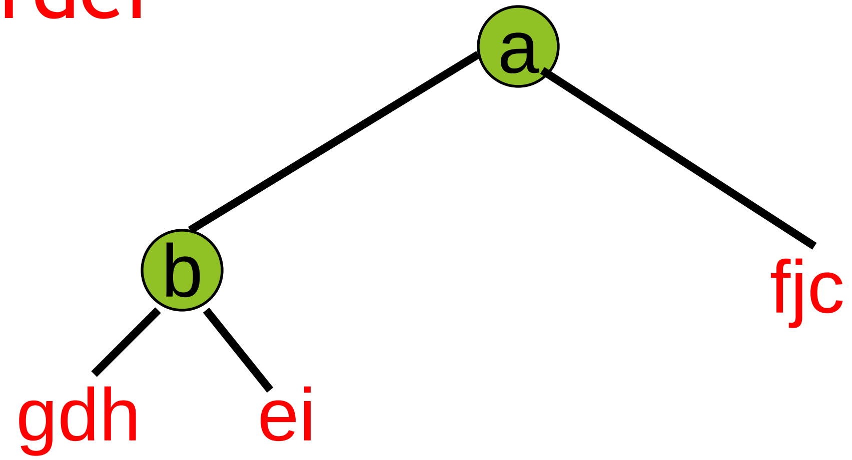
Inorder And Preorder



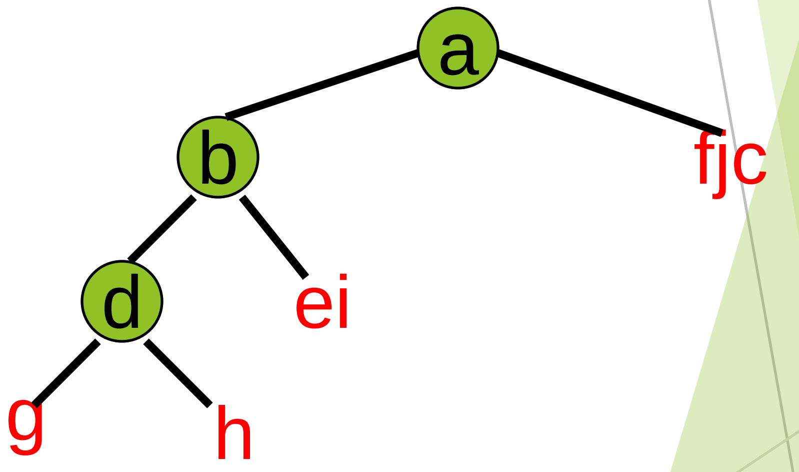
- preorder = a b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



Inorder And Preorder

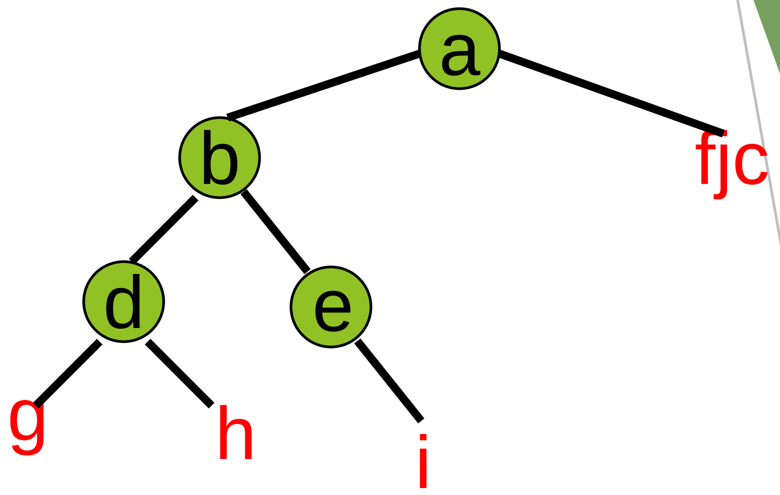


- preorder = d g h e i c f j
- d is the next root; g is in the left subtree; h is in the right subtree.

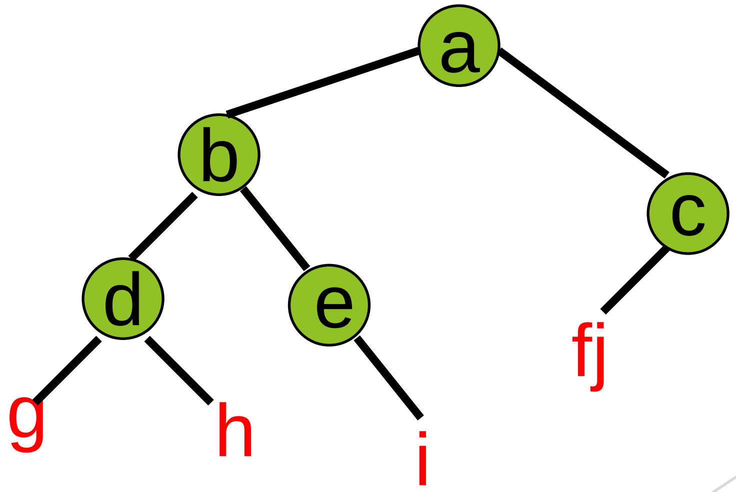


Inorder And Preorder

- preorder = e i c f j
- e is the next root; i is in the right subtree.

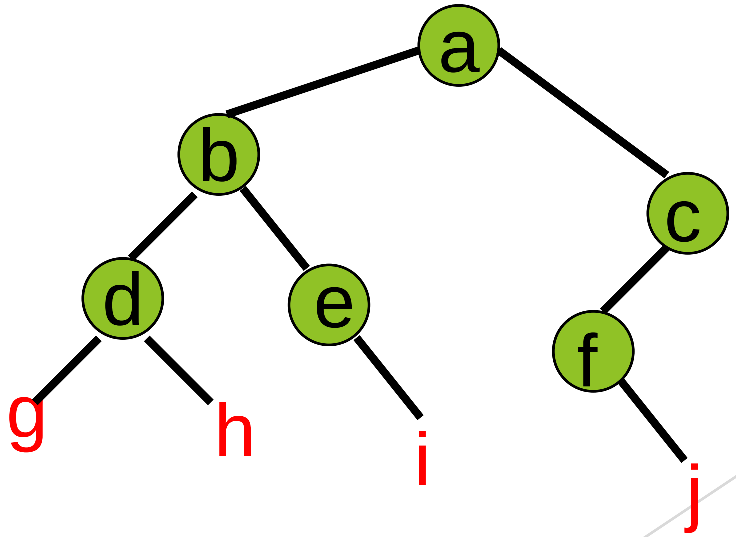


- preorder = c f j
- c is the next root; fj is in the left subtree.



Inorder And Preorder

- preorder = f j
- f is the next root; j is in the right subtree.

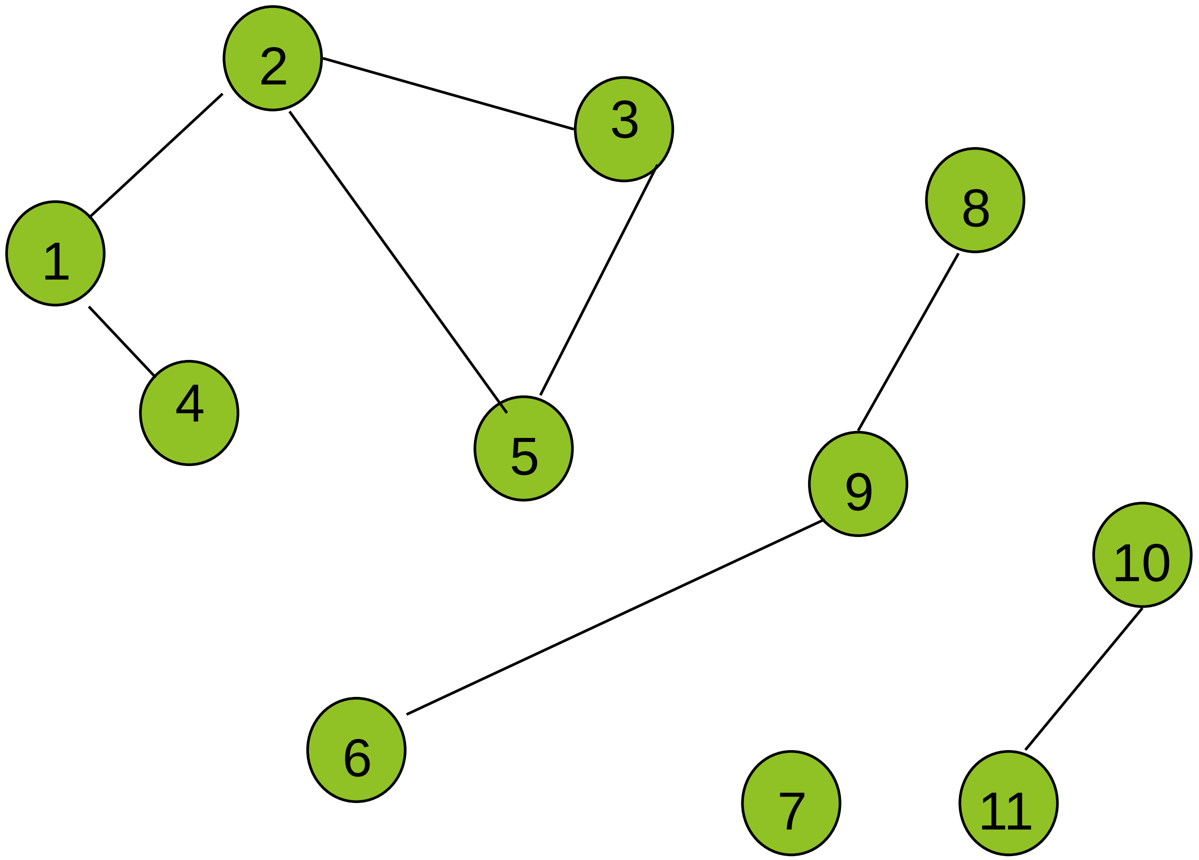


Inorder And Postorder

- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Graph Search Methods

- A vertex u is reachable from vertex v iff there is a path from v to u.



Graph Search Methods

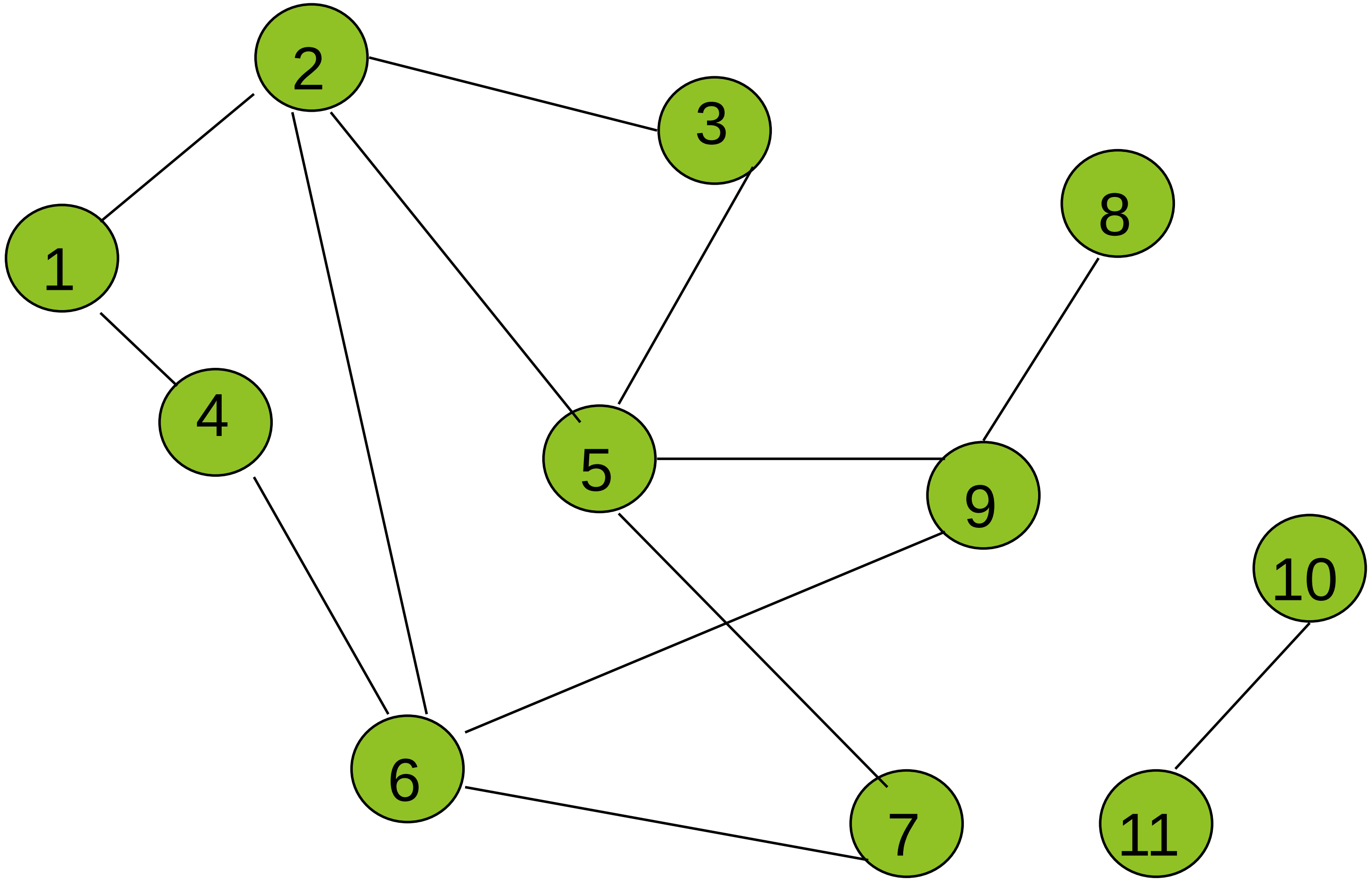
- Many graph problems solved using a search method.
  - Path from one vertex to another.
  - Is the graph connected?
  - Find a spanning tree.
  - Etc.
- Commonly used search methods:
  - Breadth-first search.
  - Depth-first search.

Breadth-First Search

- Visit start vertex and put into a FIFO queue.
- Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue.

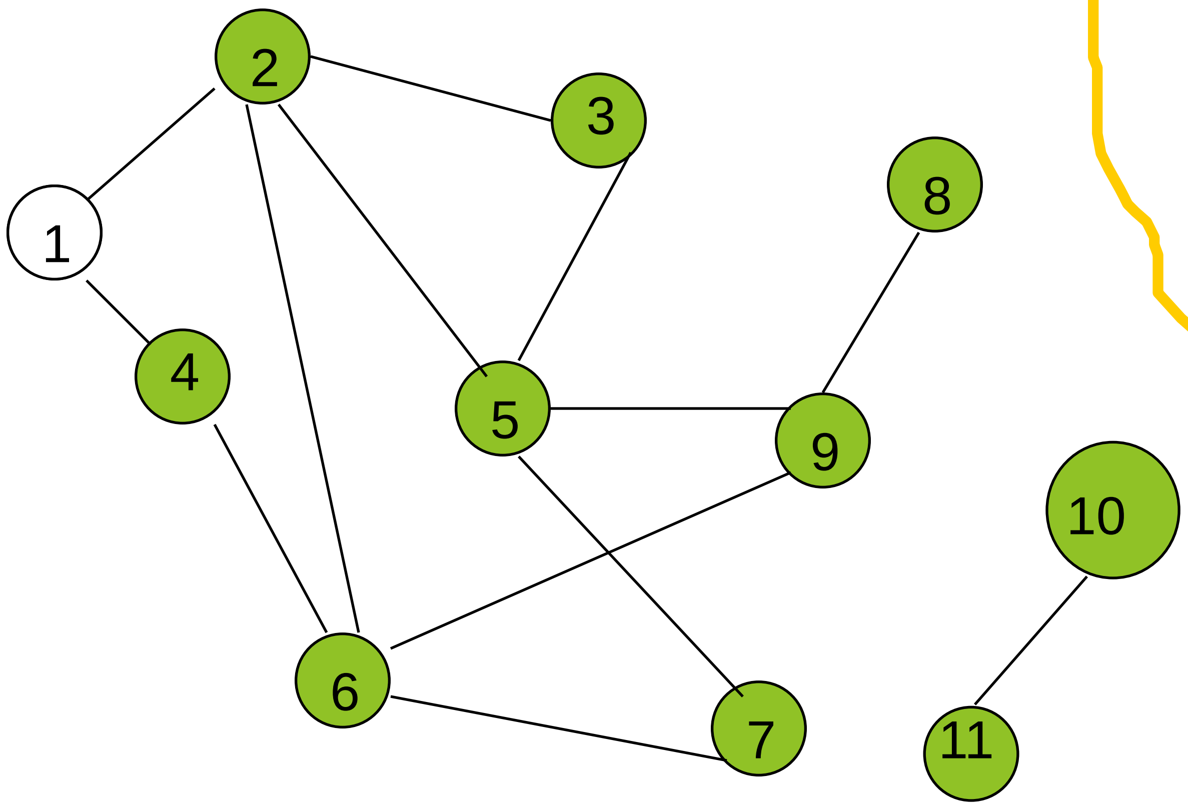
```
Algorithm BFS(v)
// A breadth first search of G is carried out beginning
// at vertex v. For any node i, visited[i] = 1 if i has
// already been visited. The graph G and array visited[ ]
// are global; visited[ ] is initialized to zero.
{
    u := v; // q is a queue of unexplored vertices.
    visited[v] := 1;
    repeat
    {
        for all vertices w adjacent from u do
        {
            if (visited[w] = 0) then
            {
                Add w to q; // w is unexplored.
                visited[w] := 1;
            }
        }
        if q is empty then return; // No unexplored vertex.
        Delete u from q; // Get first unexplored vertex.
    } until(false);
}
```

Breadth-First Search Example



Start search at vertex 1.

Breadth-First Search Example

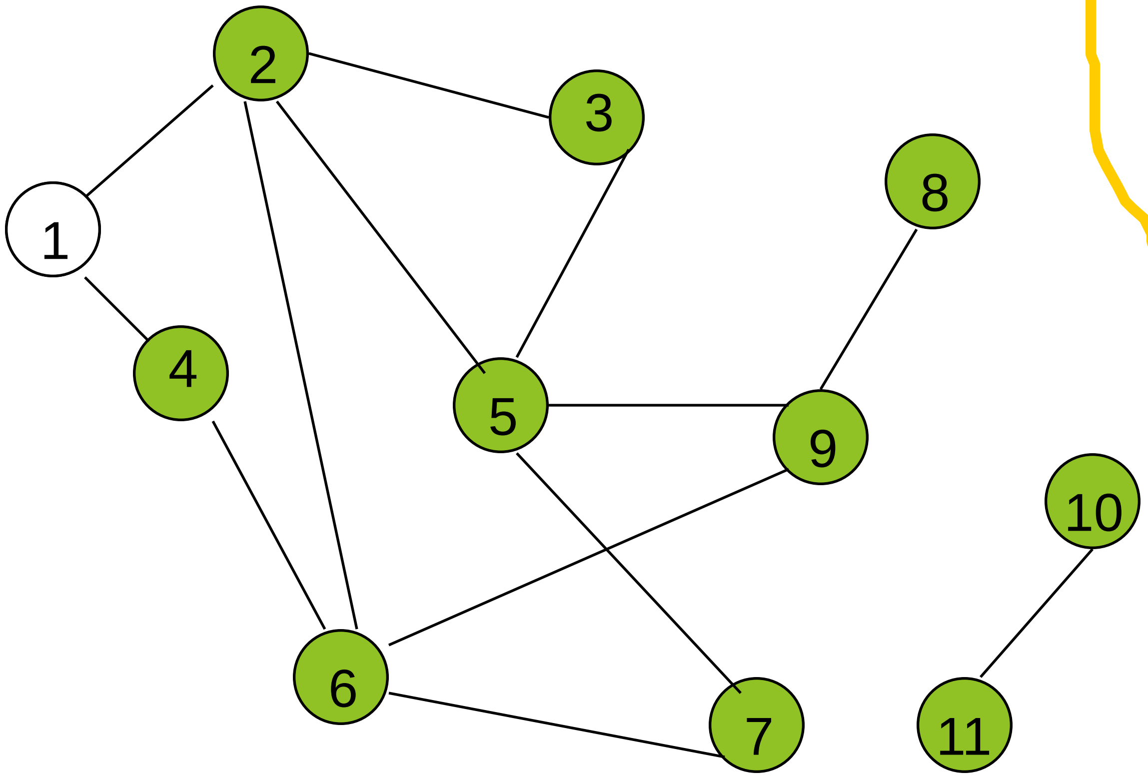


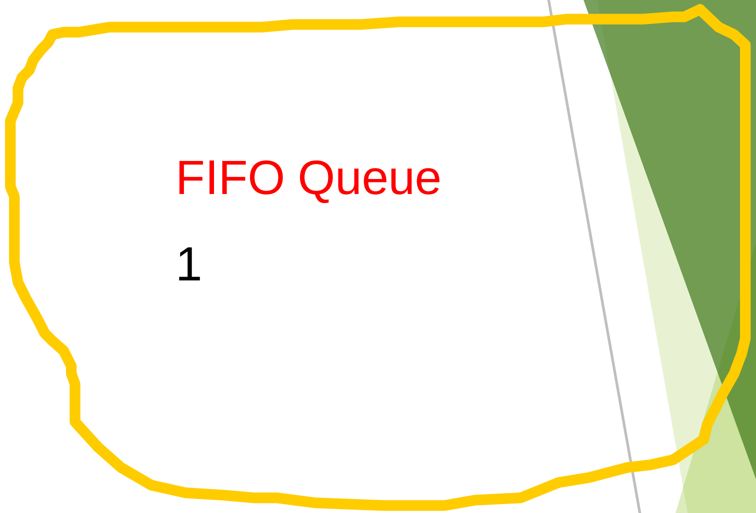
FIFO Queue

1

Visit/mark/label start vertex and put in a FIFO queue.

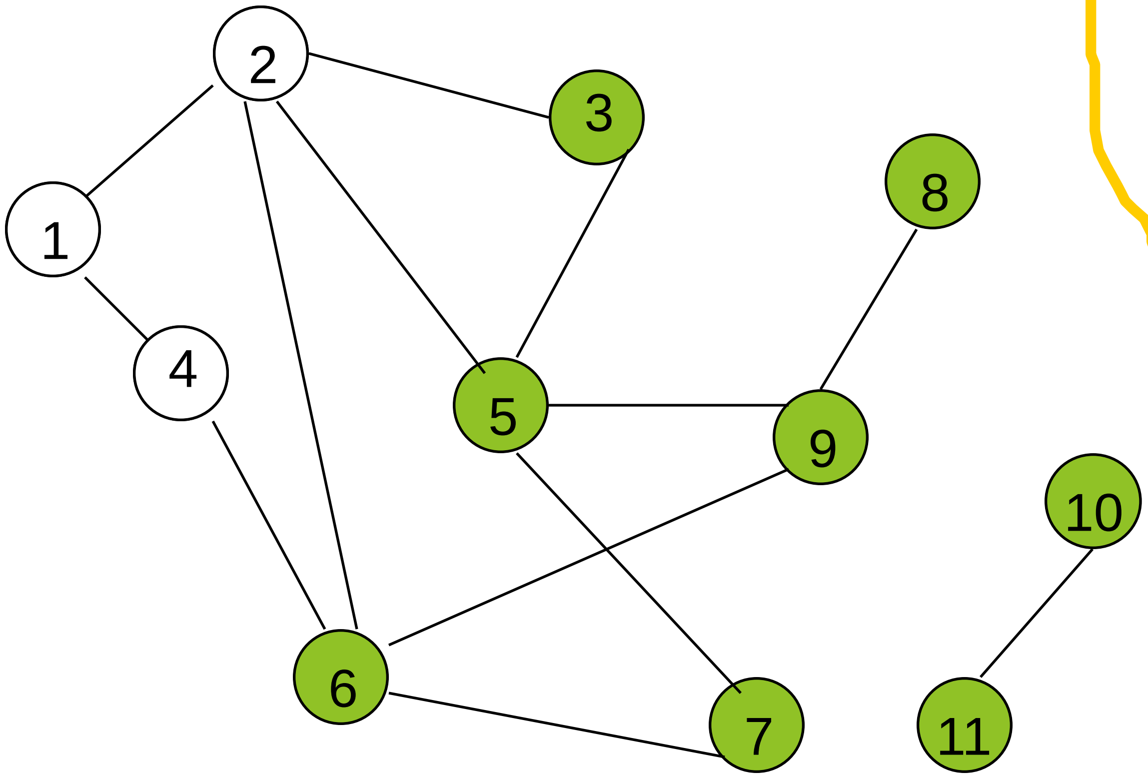
Breadth-First Search Example





Remove 1 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

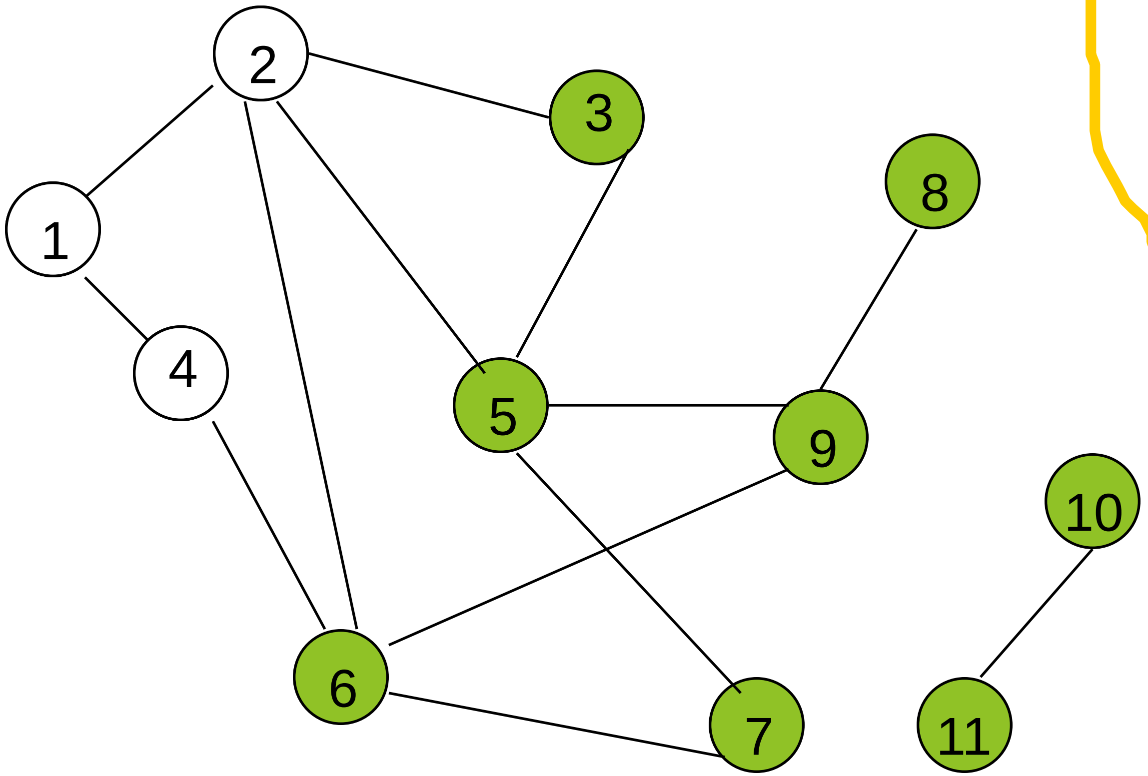


FIFO Queue

2 4

Remove 1 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

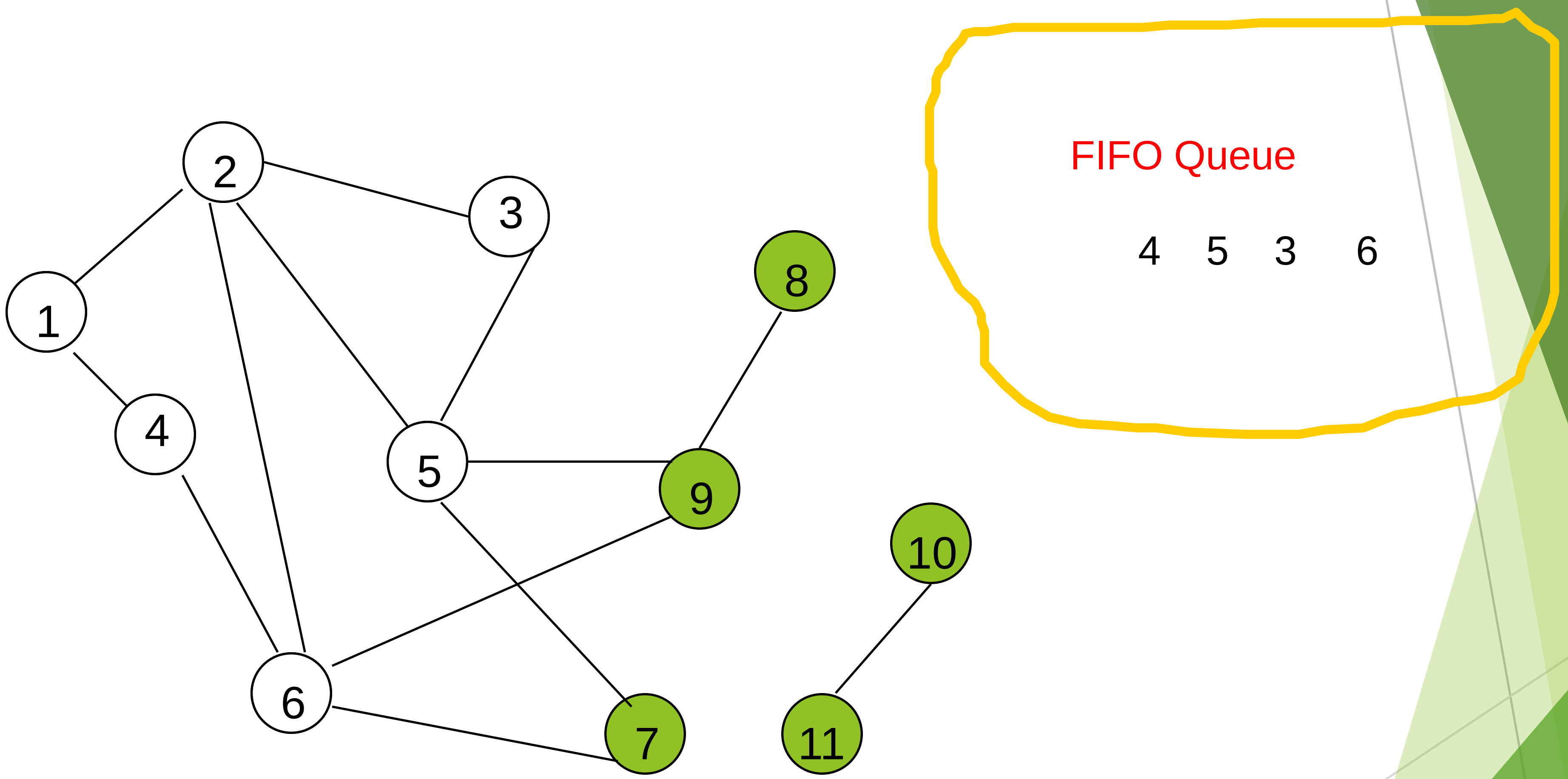


FIFO Queue

2 4

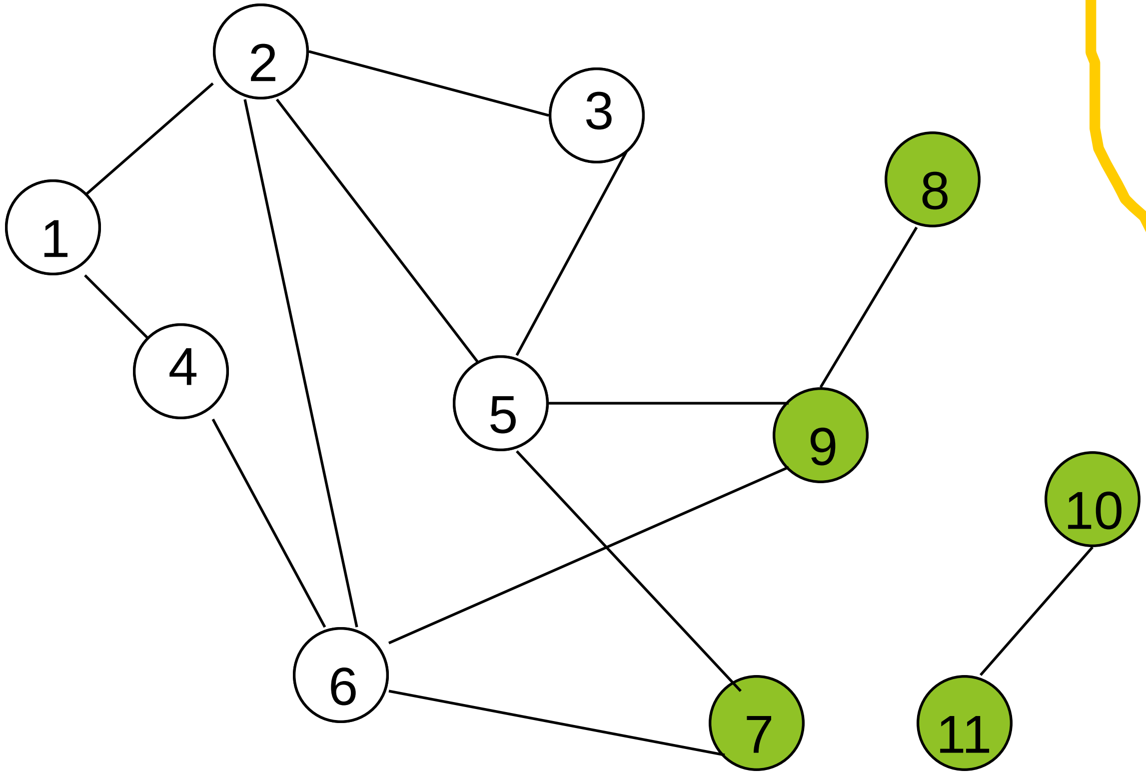
Remove 2 from Q; visit adjacent unvisited vertices; put in Q.

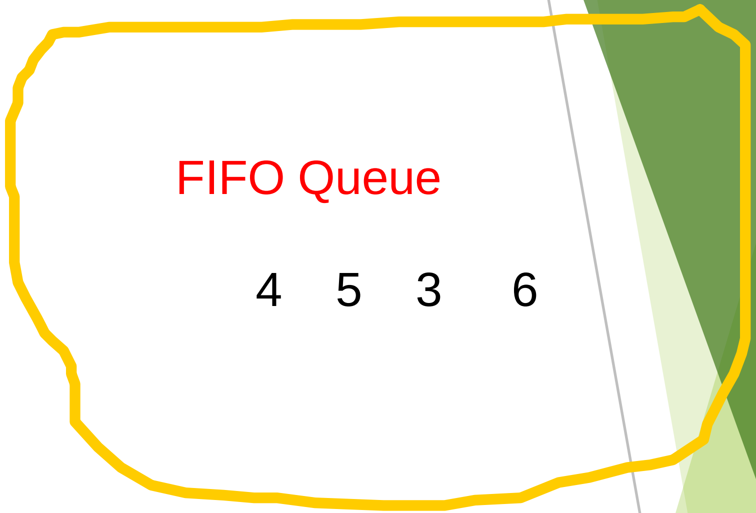
Breadth-First Search Example



Remove 2 from Q; visit adjacent unvisited vertices; put in Q.

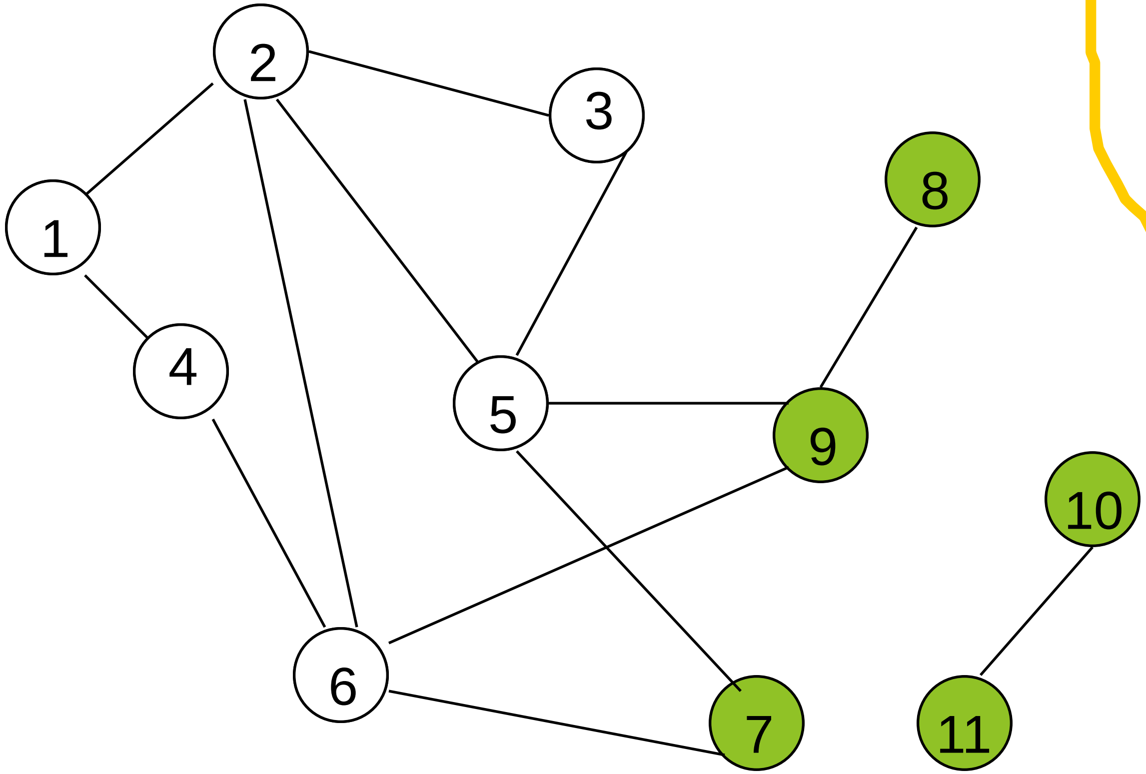
Breadth-First Search Example





Remove 4 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

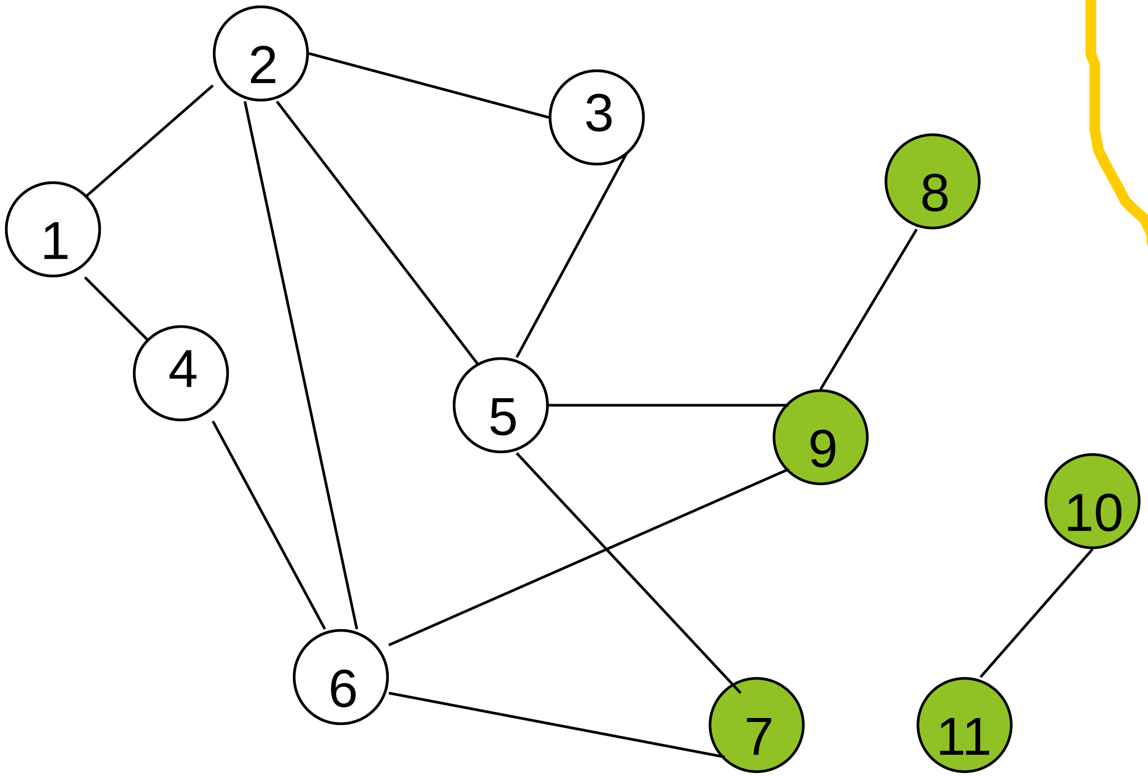


FIFO Queue

5 3 6

Remove 4 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

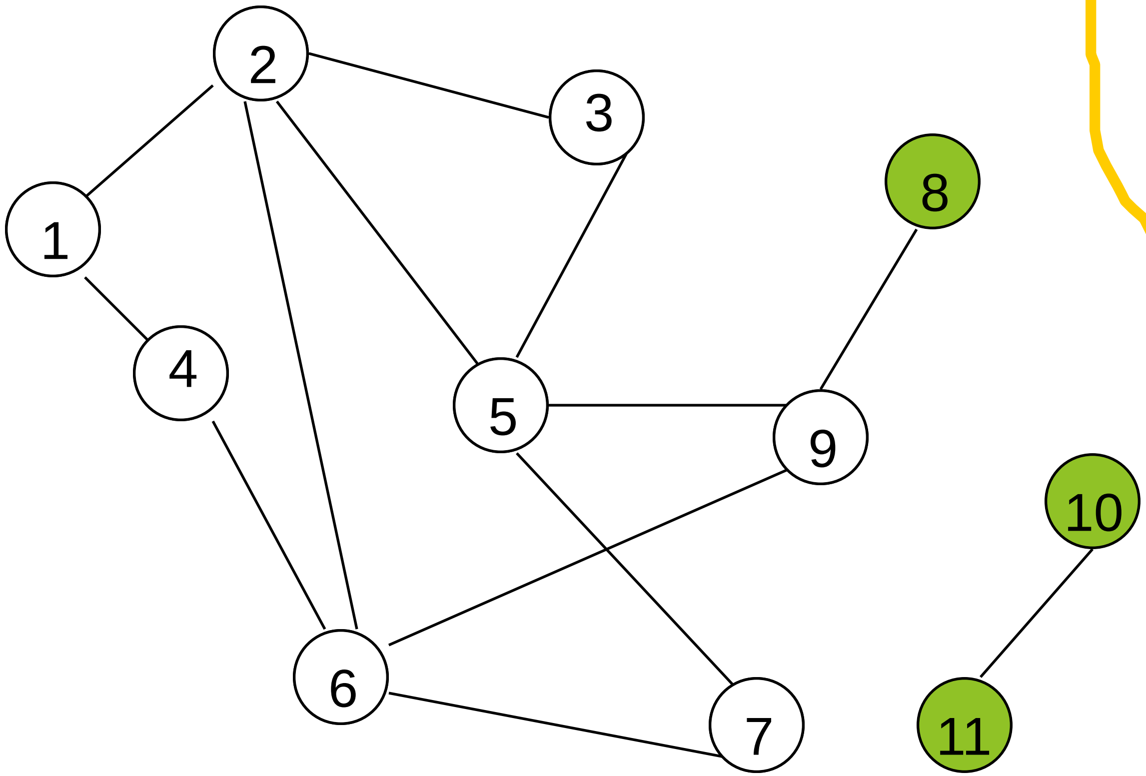


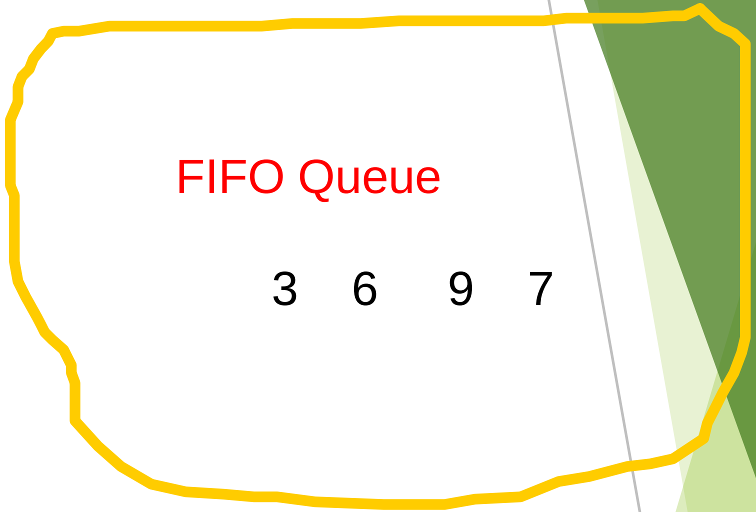
FIFO Queue

5 3 6

Remove 5 from Q; visit adjacent unvisited vertices; put in Q.

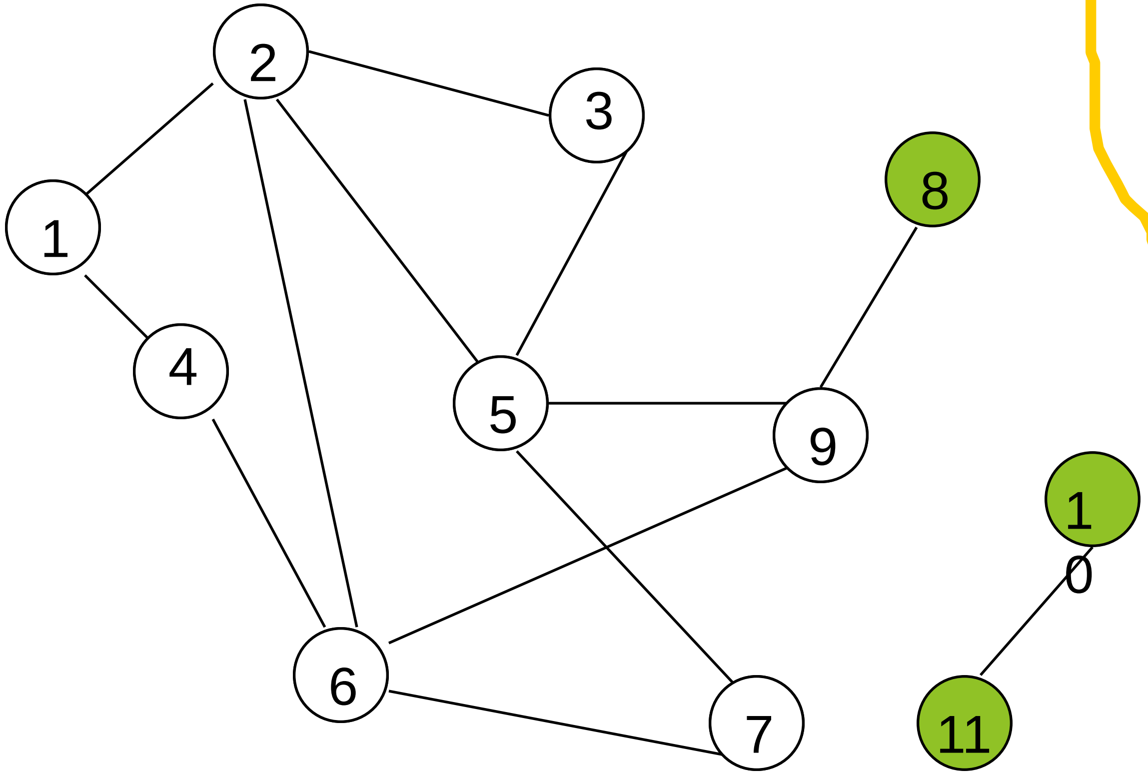
Breadth-First Search Example





Remove 5 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

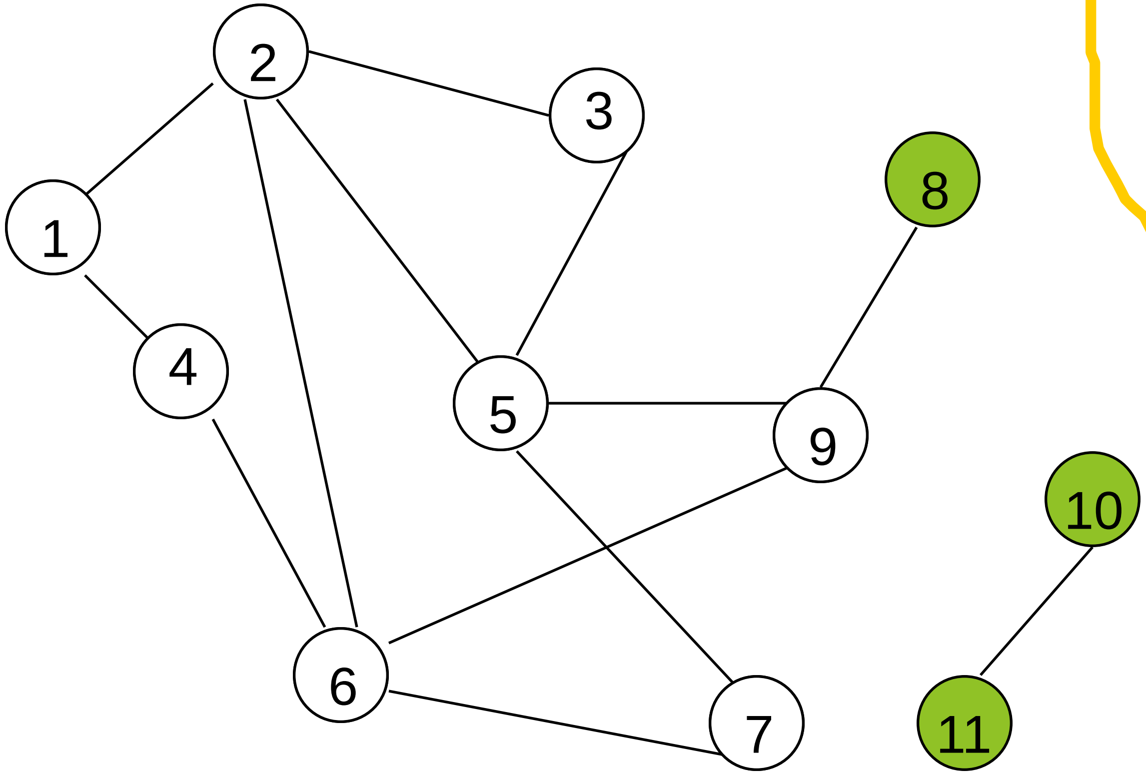


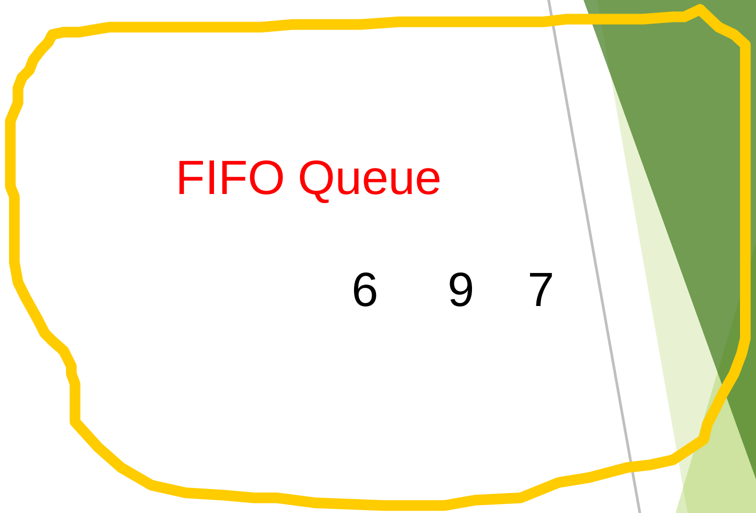
FIFO Queue

3 6 9 7

Remove 3 from Q; visit adjacent unvisited vertices; put in Q.

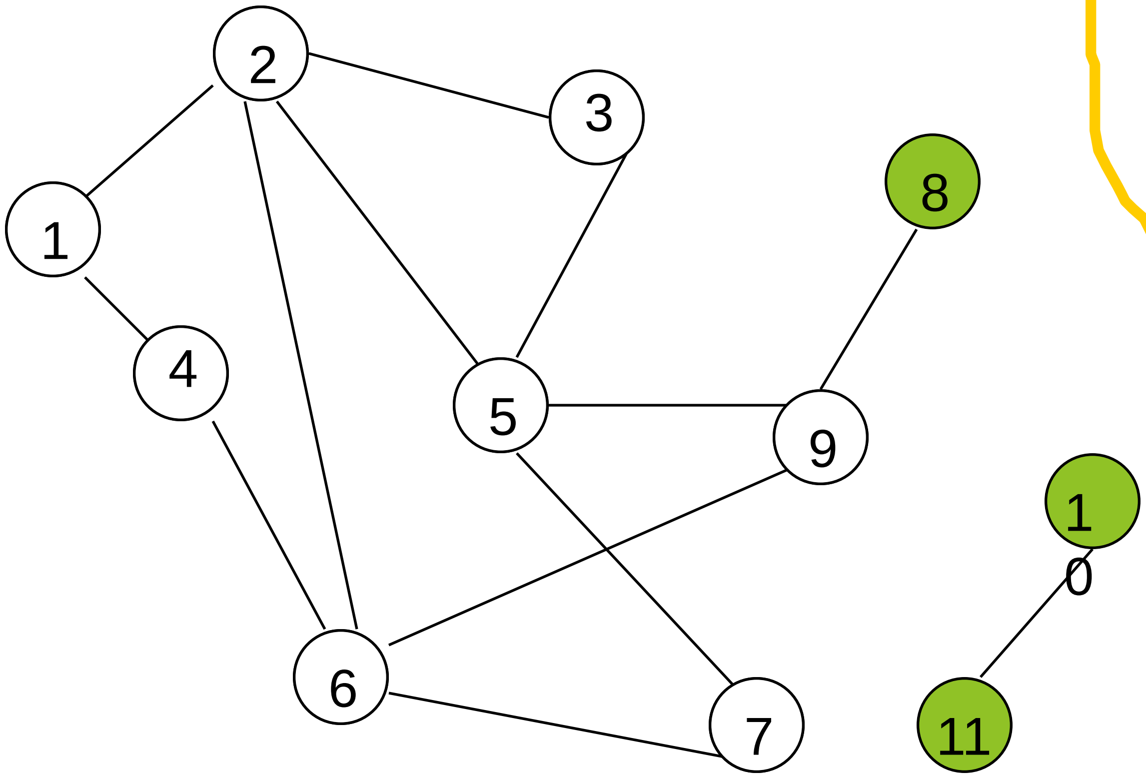
Breadth-First Search Example

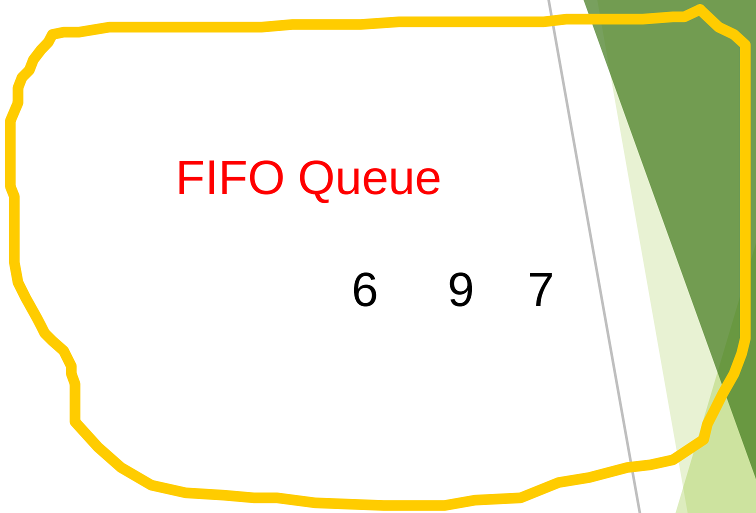




Remove 3 from Q; visit adjacent unvisited vertices; put in Q.

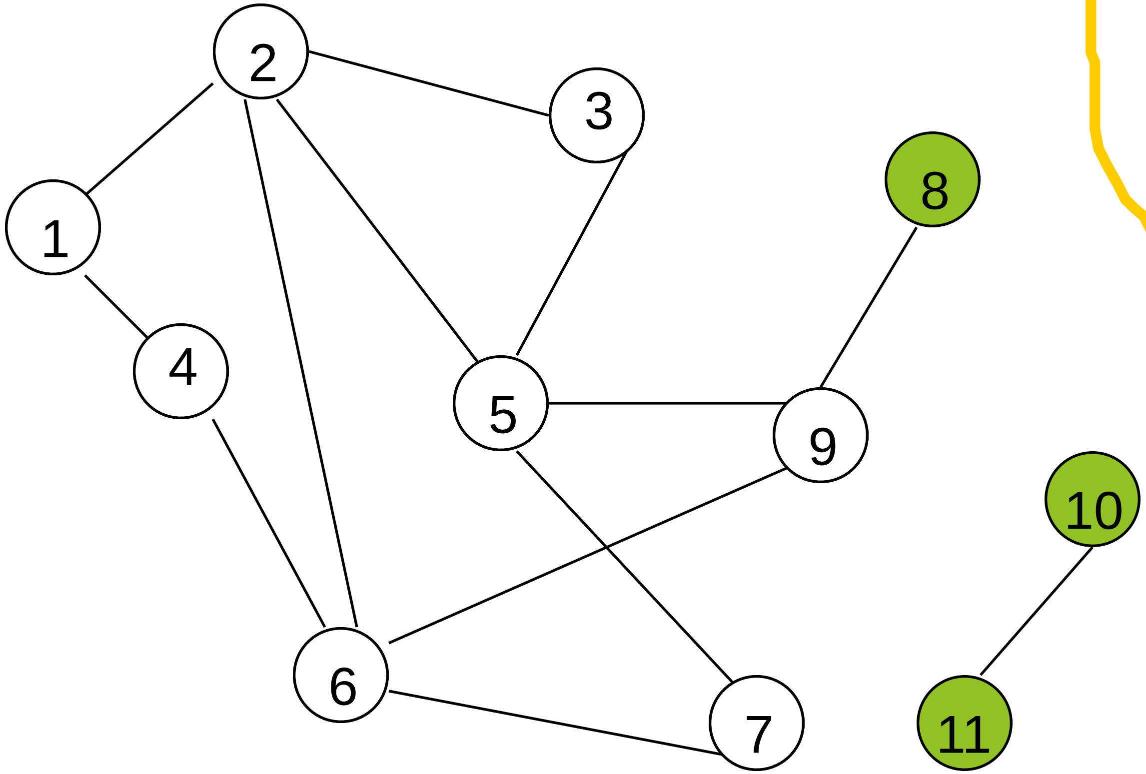
Breadth-First Search Example





Remove 6 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

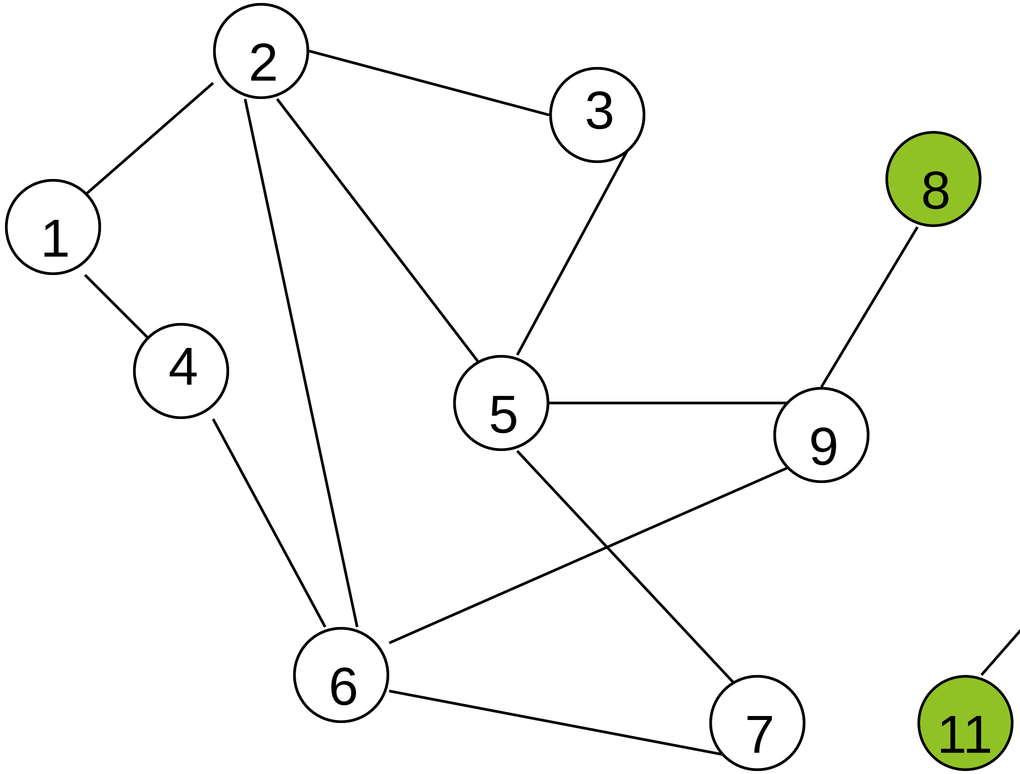


FIFO Queue

9 7

Remove 6 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

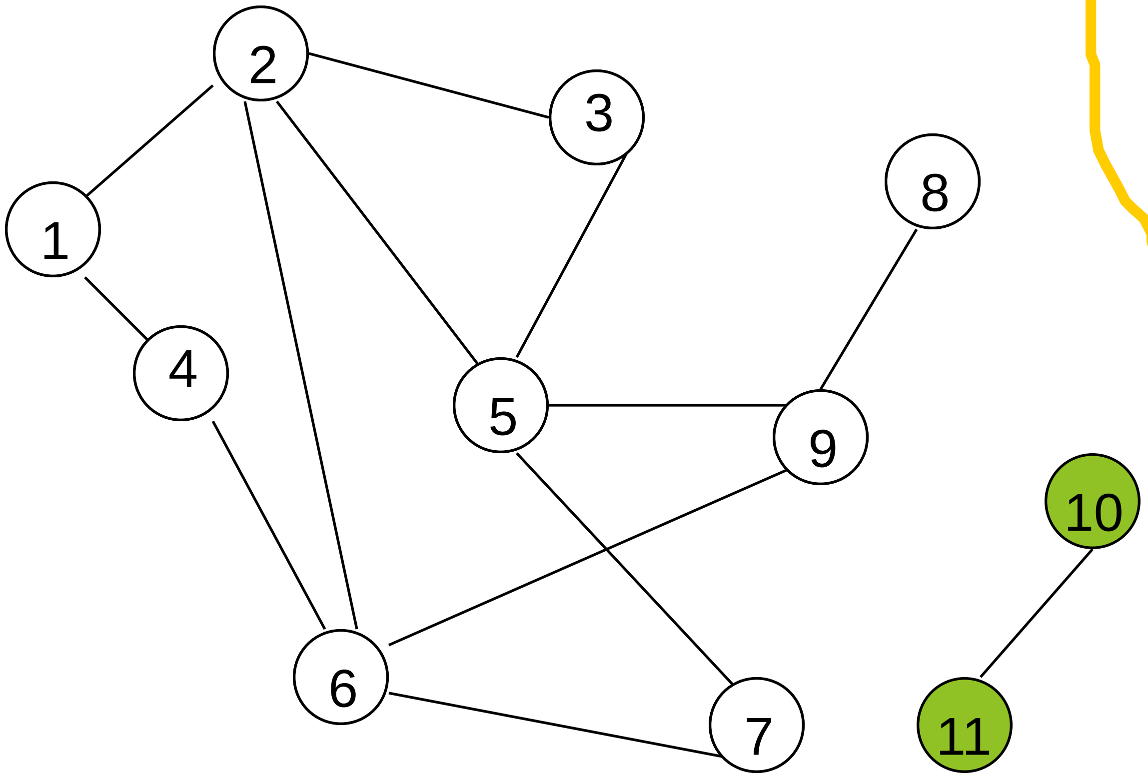


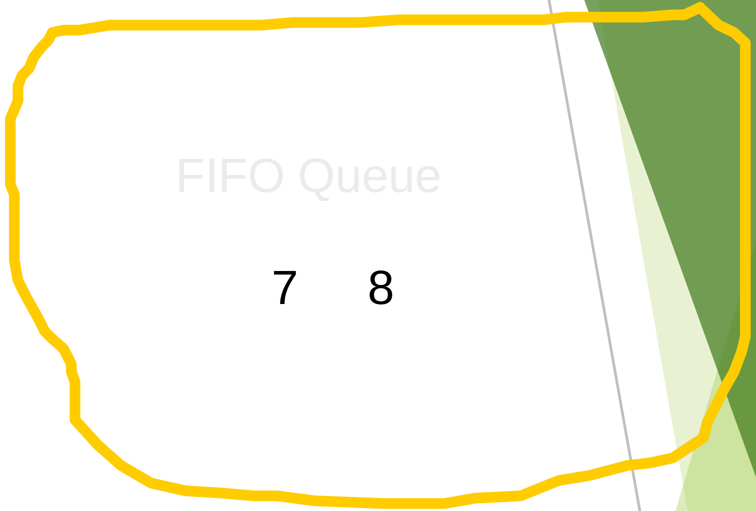
FIFO Queue

9 7

Remove 9 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example

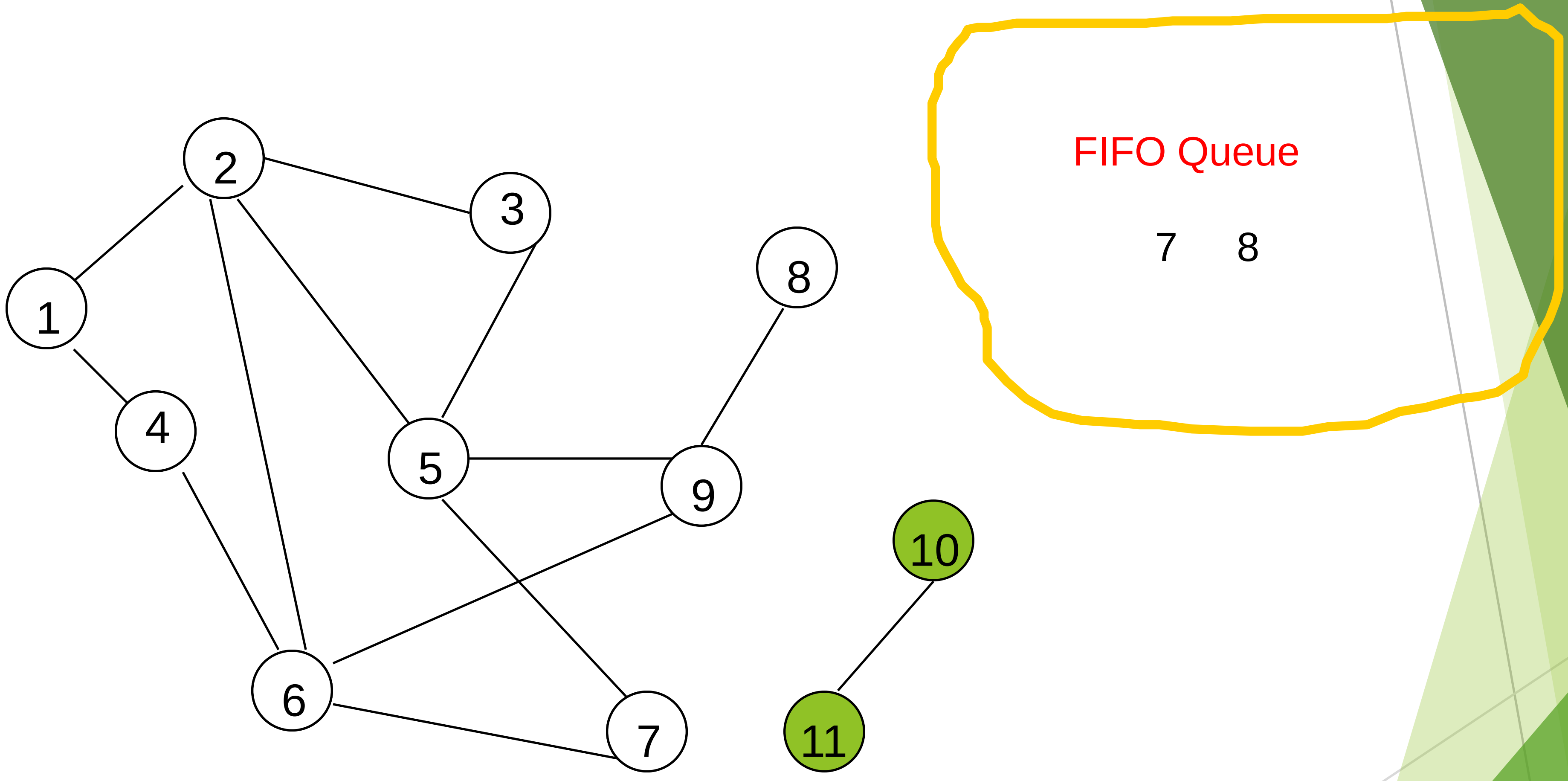




Remove 9 from Q; visit adjacent unvisited vertices;

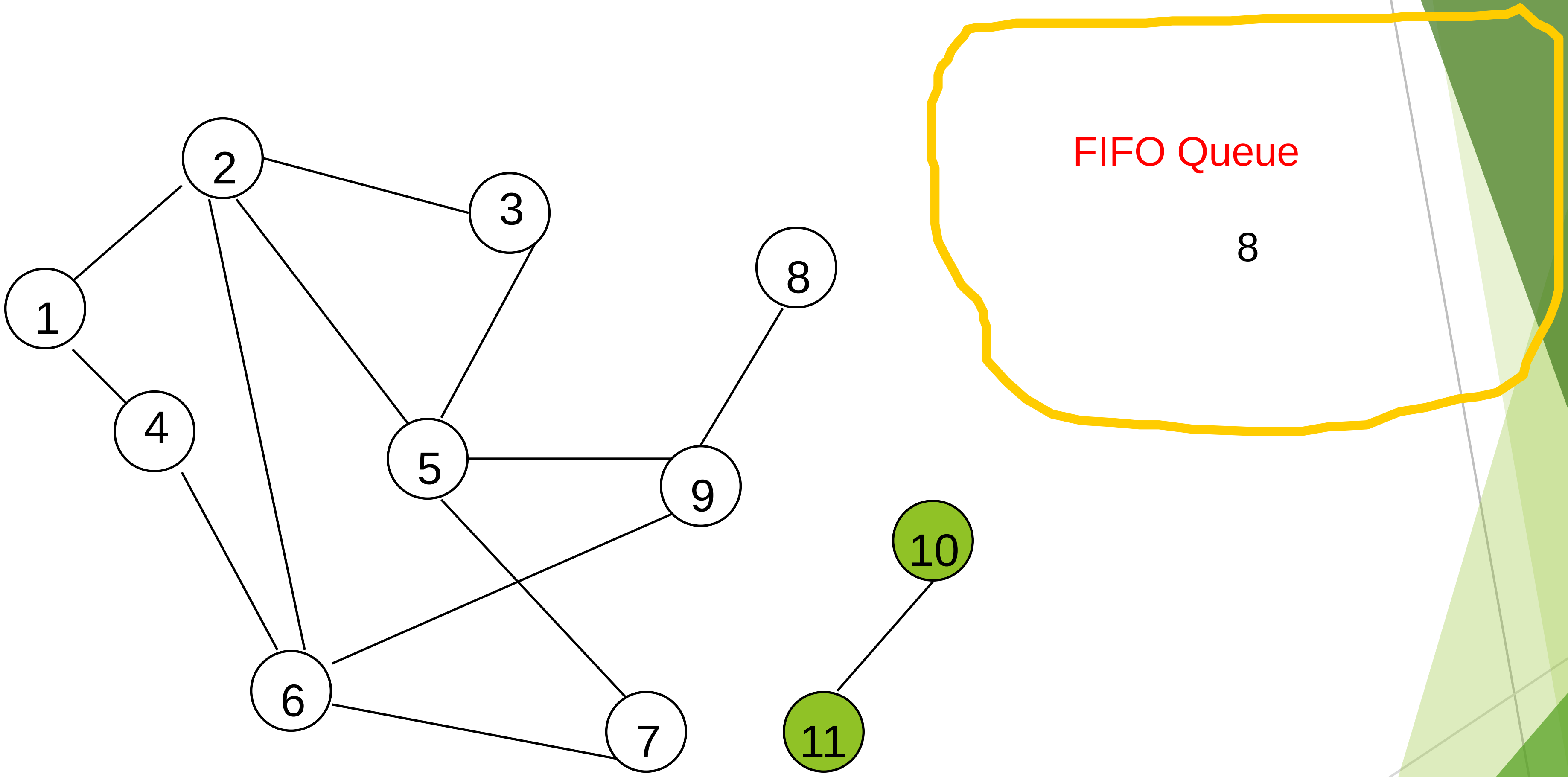
put in Q.

Breadth-First Search Example



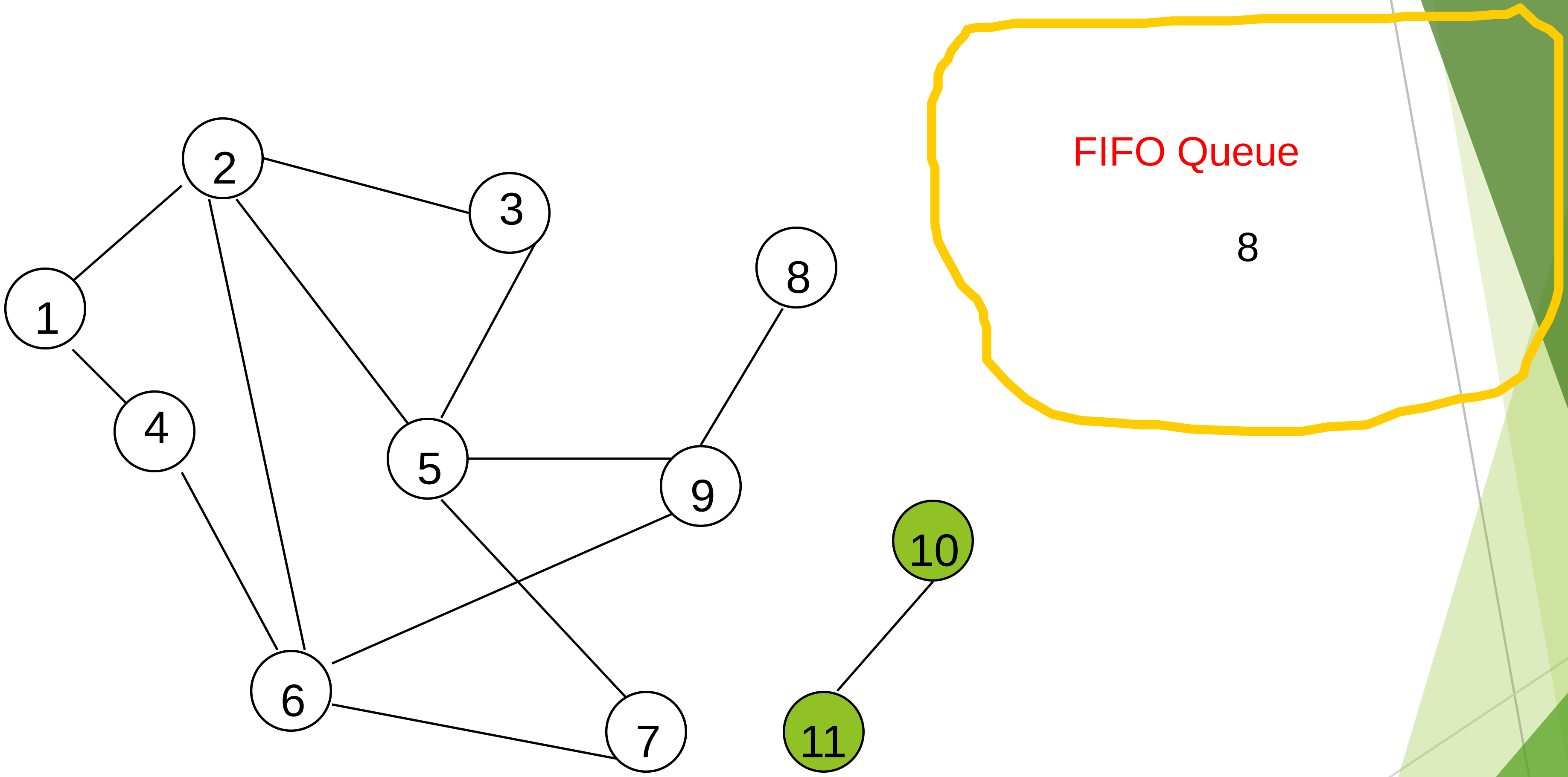
Remove 7 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example



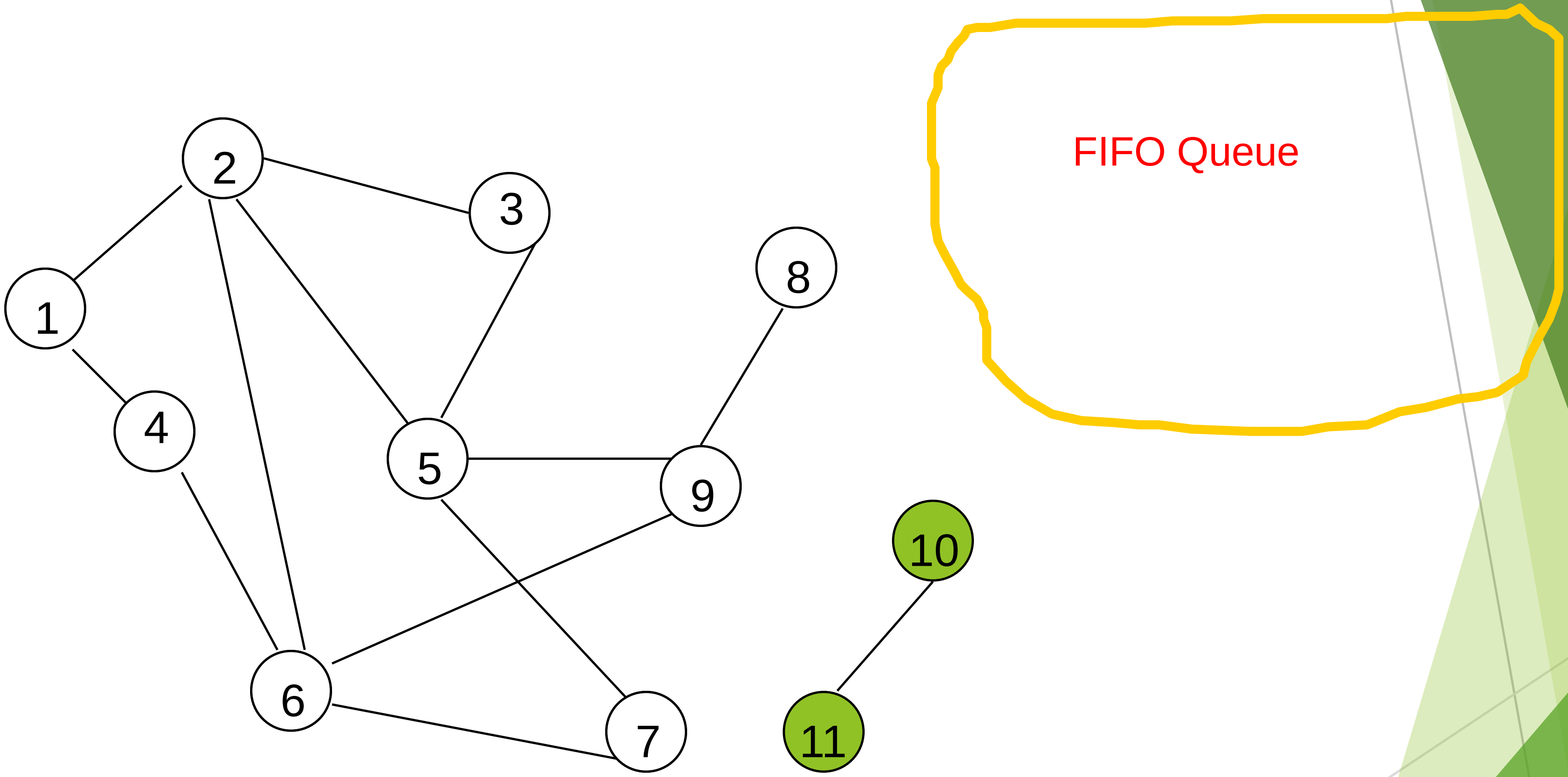
Remove 7 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example



Remove 8 from Q; visit adjacent unvisited vertices; put in Q.

Breadth-First Search Example



- Queue is empty. Search terminates.
- All vertices reachable from the start vertex (including the start vertex) are visited.

Path From Vertex v To Vertex u

- Start a breadth-first search at vertex v.
- Terminate when vertex u is visited or when Q becomes empty (whichever occurs first).
- Time
  - $O(n^2)$ when adjacency matrix used
  - $O(n+e)$ when adjacency lists used (e is number of edges)

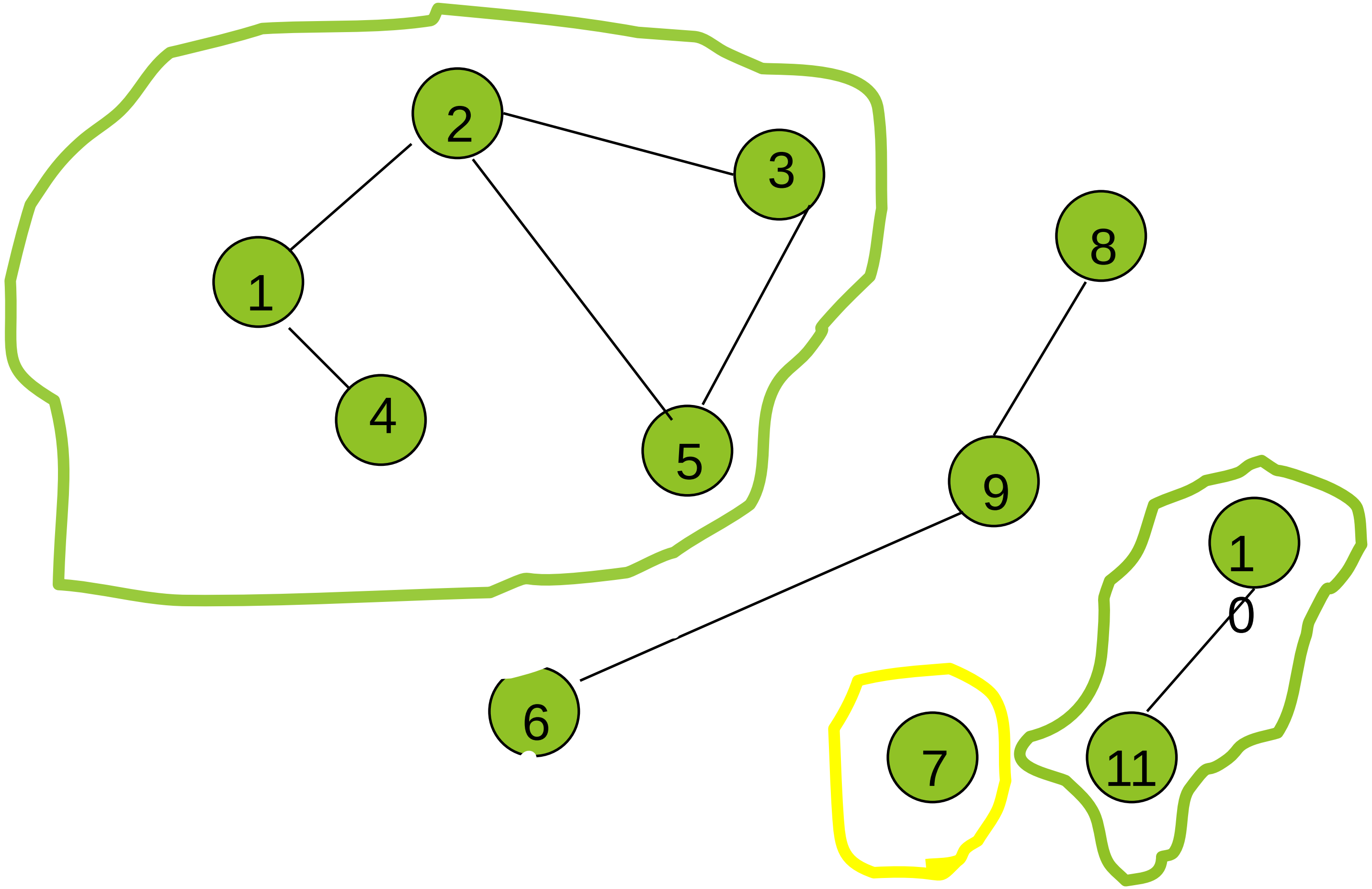
Is The Graph Connected?

- Start a breadth-first search at any vertex of the graph.
- Graph is connected iff all n vertices get visited.
- Time
  - $O(n^2)$ when adjacency matrix used
  - $O(n+e)$ when adjacency lists used (e is number of edges)

Connected Components

- Start a breadth-first search at any as yet unvisited vertex of the graph.
- Newly visited vertices (plus edges between them) define a component.
- Repeat until all vertices are visited.

Connected Components



Spanning Tree

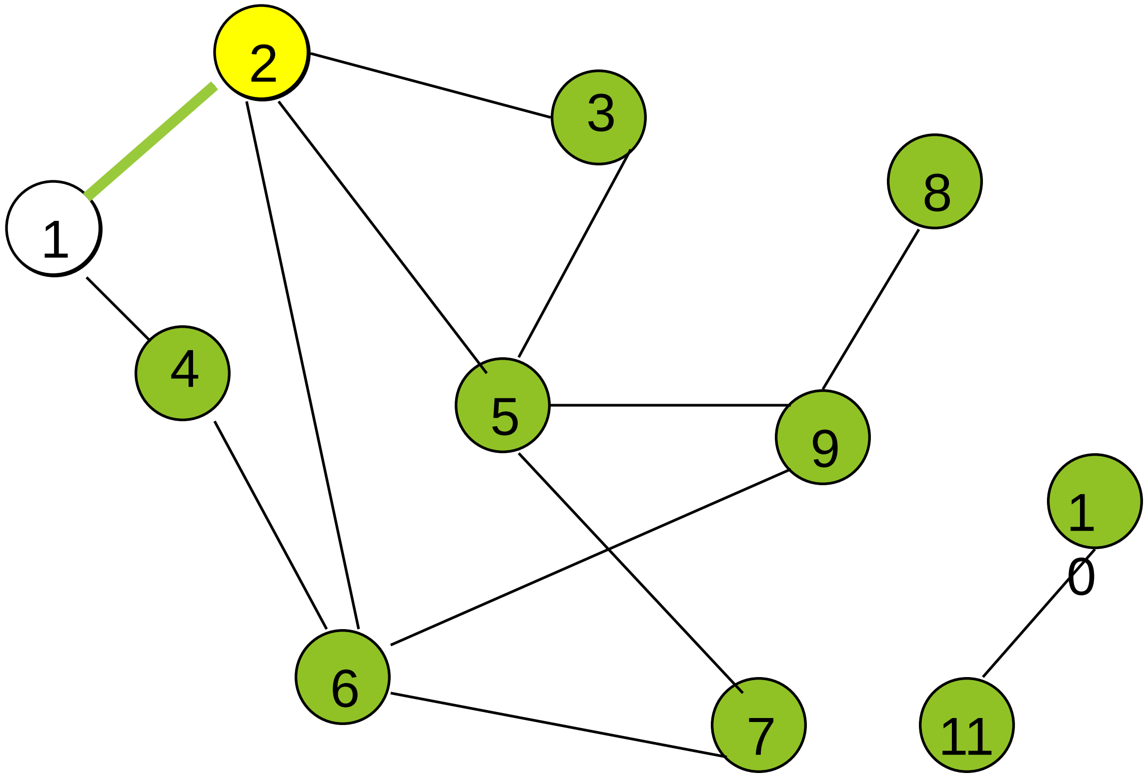
- Start a breadth-first search at any vertex of the graph.
- If graph is connected, the n-1 edges used to get to unvisited vertices define a spanning tree (breadth-first spanning tree).
- Time
  - $O(n^2)$ when adjacency matrix used
  - $O(n+e)$ when adjacency lists used (e is number of edges)

Depth-First Search

- Note that vertices adjacent from v are examined one at a time.
- As soon as an unreached adjacent vertex w is found, a DFS(w) is done.
- Remaining vertices adjacent from v are examined after DFS(w) completes.

```
Algorithm DFS(v)
// Given an undirected (directed) graph G = (V, E) with
// n vertices and an array visited[ ] initially set
// to zero, this algorithm visits all vertices
// reachable from v. G and visited[ ] are global.
{
    visited[v] := 1;
    for each vertex w adjacent from v do
    {
        if (visited[w] = 0) then DFS(w);
    }
}
```

Depth-First Search Example

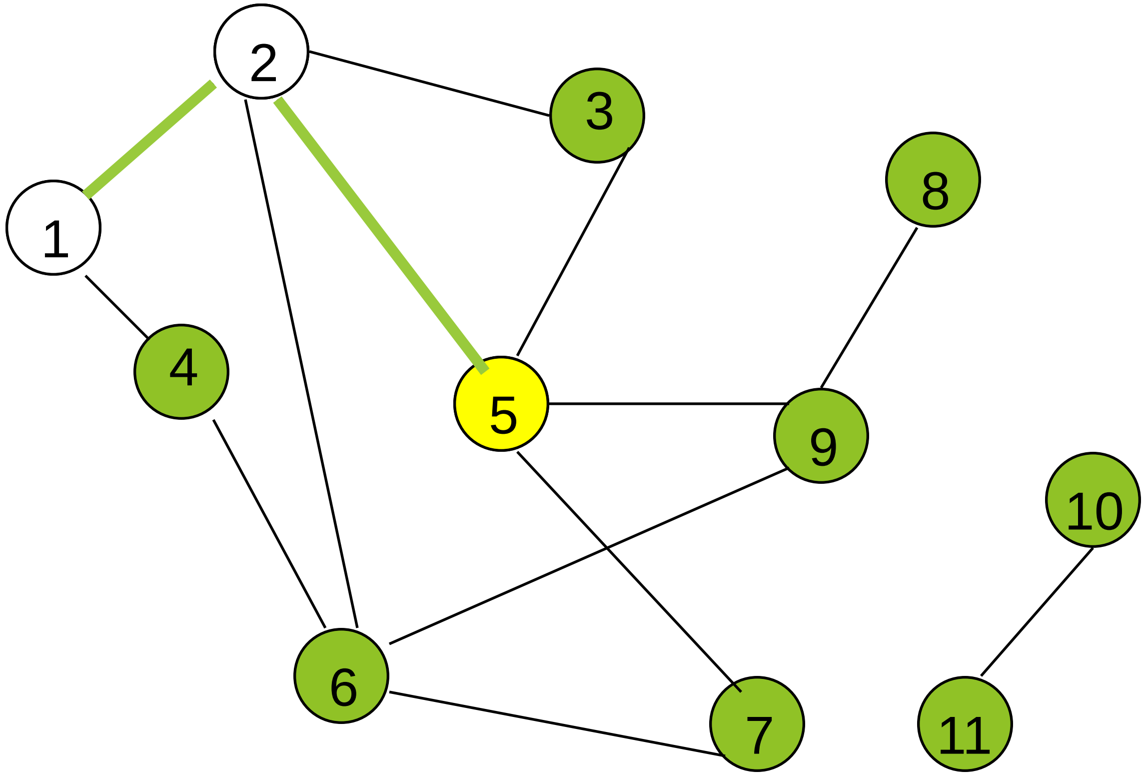


Start search at vertex 1.

Label vertex 1 and do a depth first search from either 2 or 4.

Suppose that vertex 2 is selected.

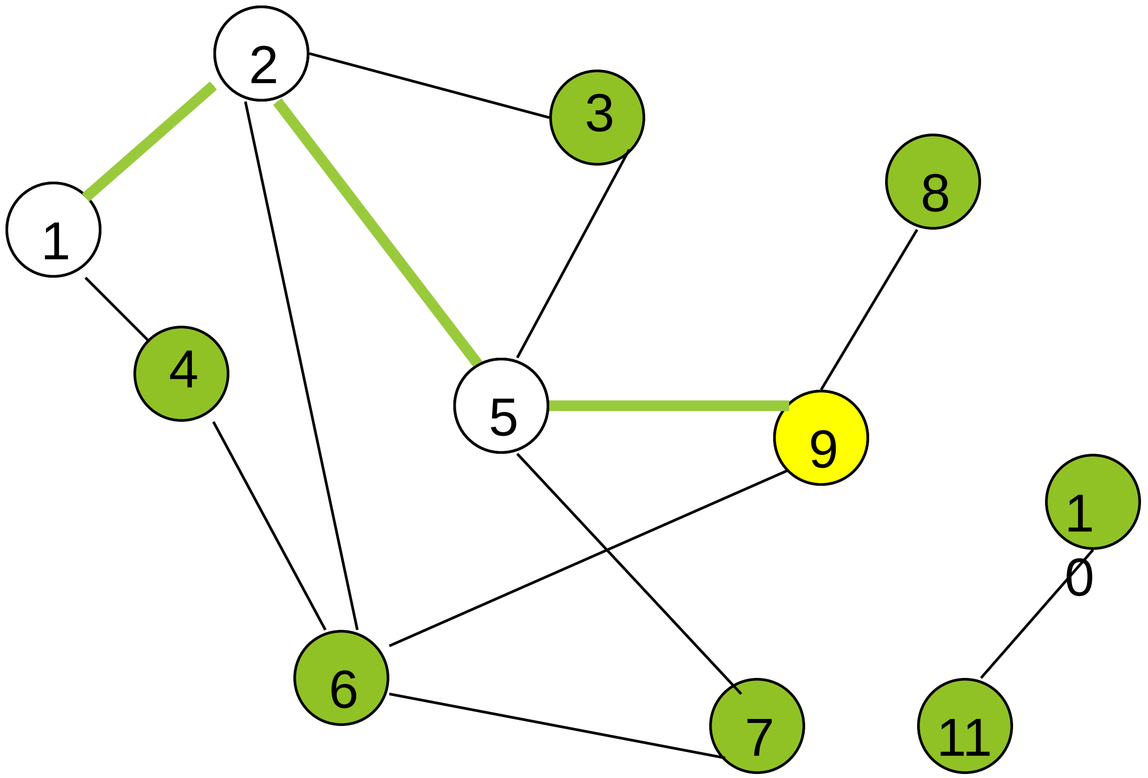
Depth-First Search Example



Label vertex 2 and do a depth first search from either 3, 5, or 6.

Suppose that vertex 5 is selected.

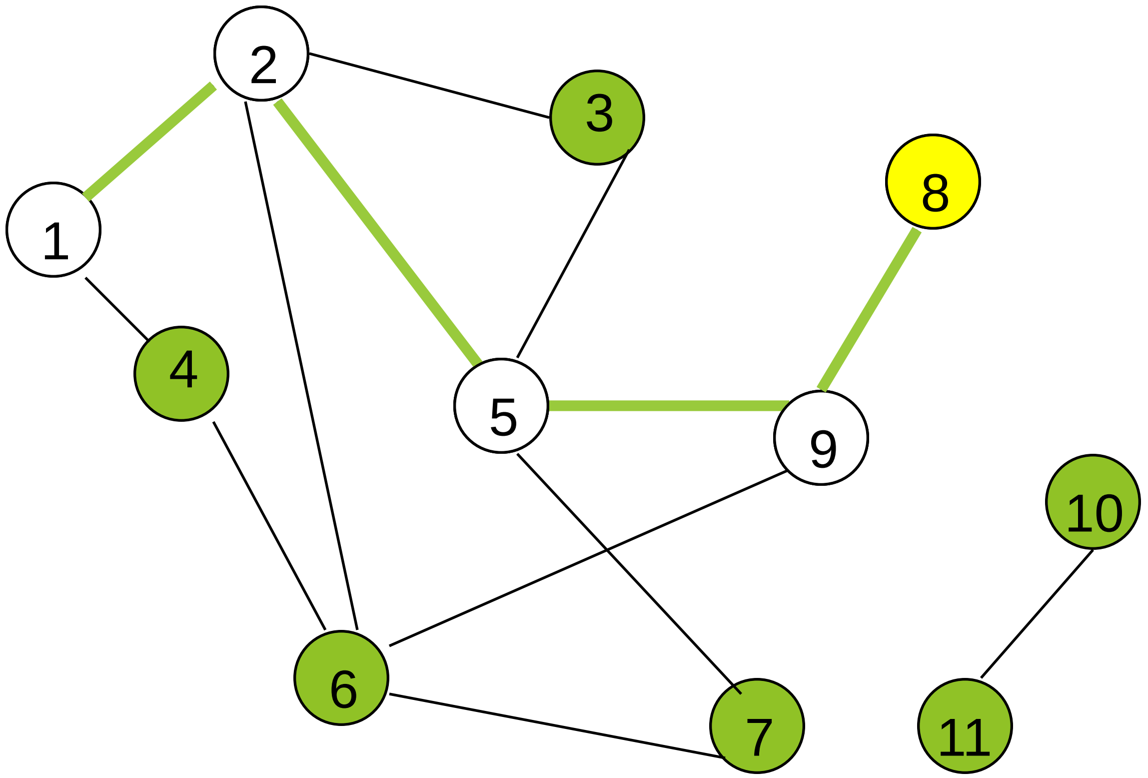
Depth-First Search Example



Label vertex 5 and do a depth first search from either 3, 7, or 9.

Suppose that vertex 9 is selected.

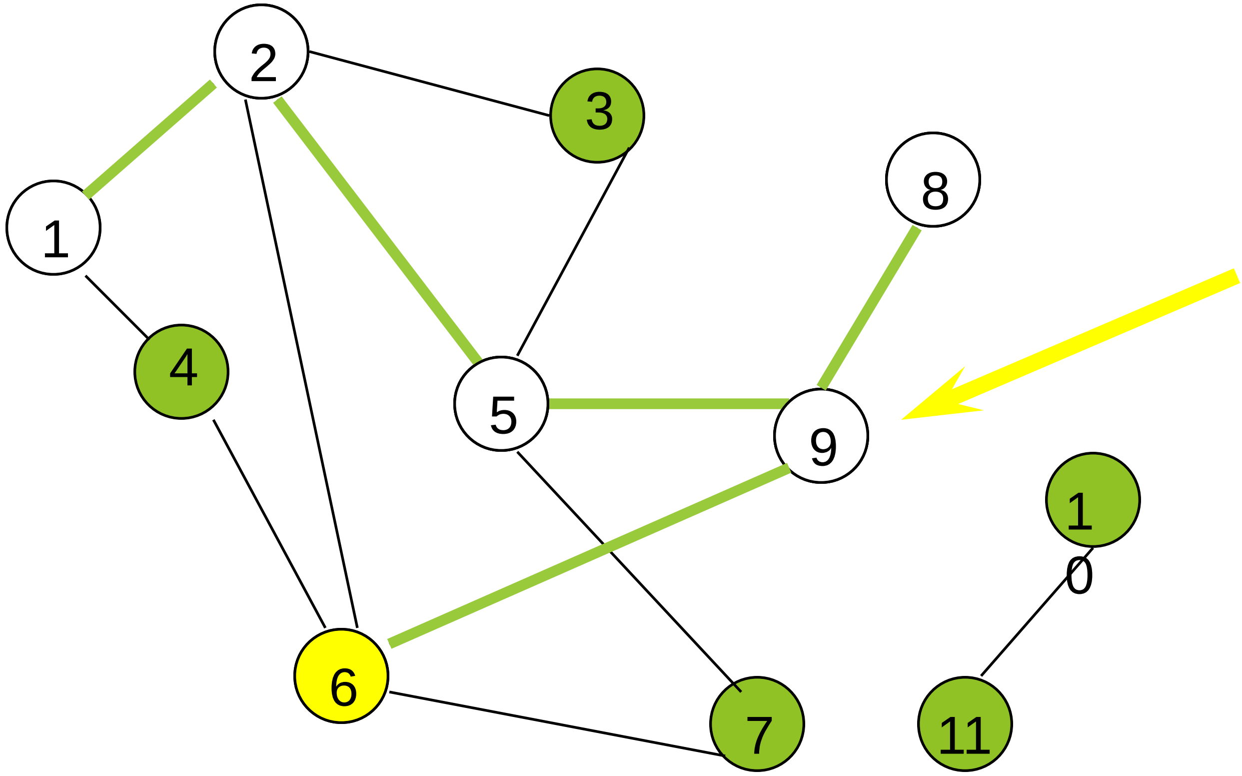
Depth-First Search Example



Label vertex 9 and do a depth first search from either 6 or 8.

Suppose that vertex 8 is selected.

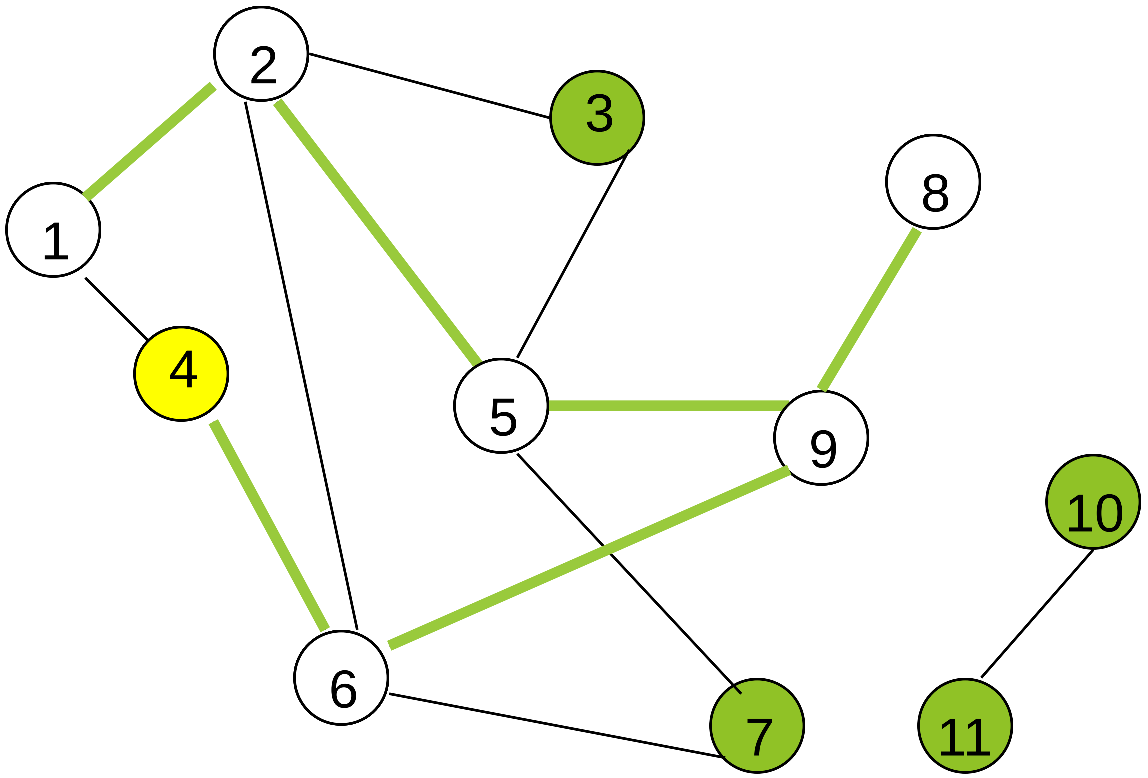
Depth-First Search Example



Label vertex 8 and return to vertex 9.

From vertex 9 do a DFS(6).

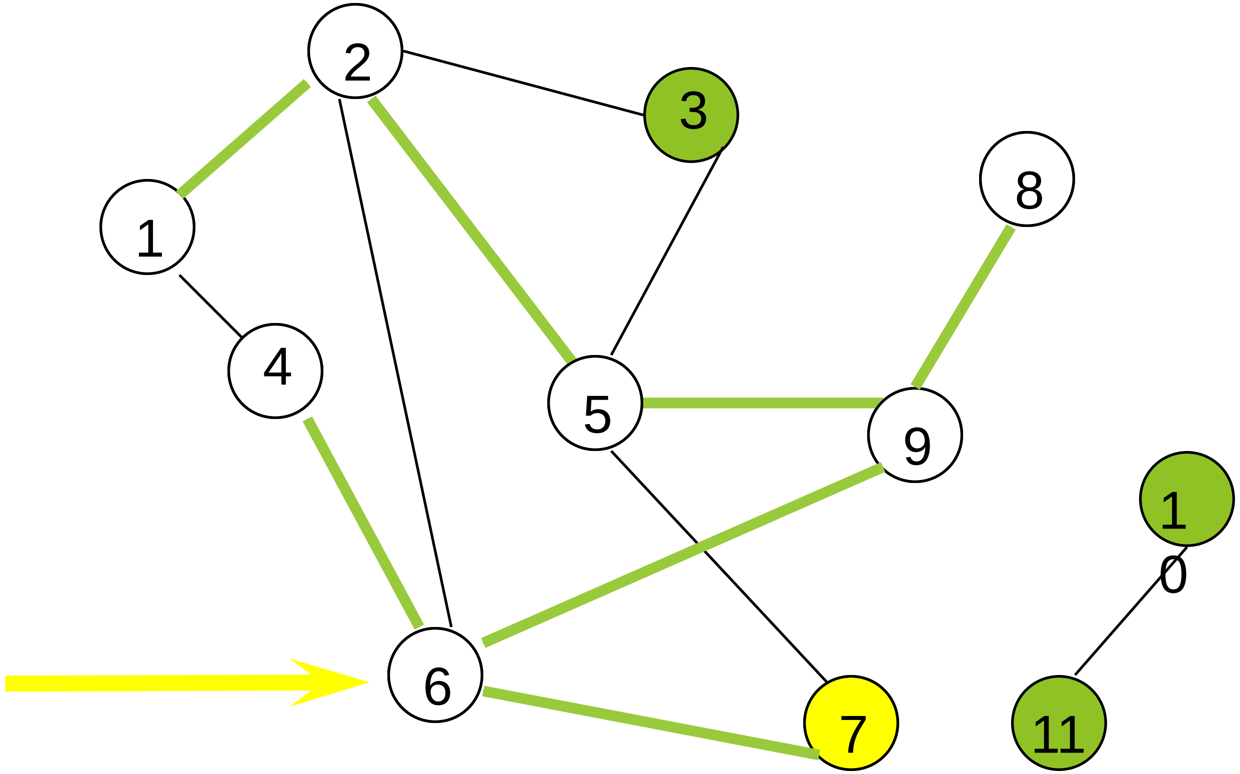
Depth-First Search Example



Label vertex 6 and do a depth first search from either 4 or 7.

Suppose that vertex 4 is selected.

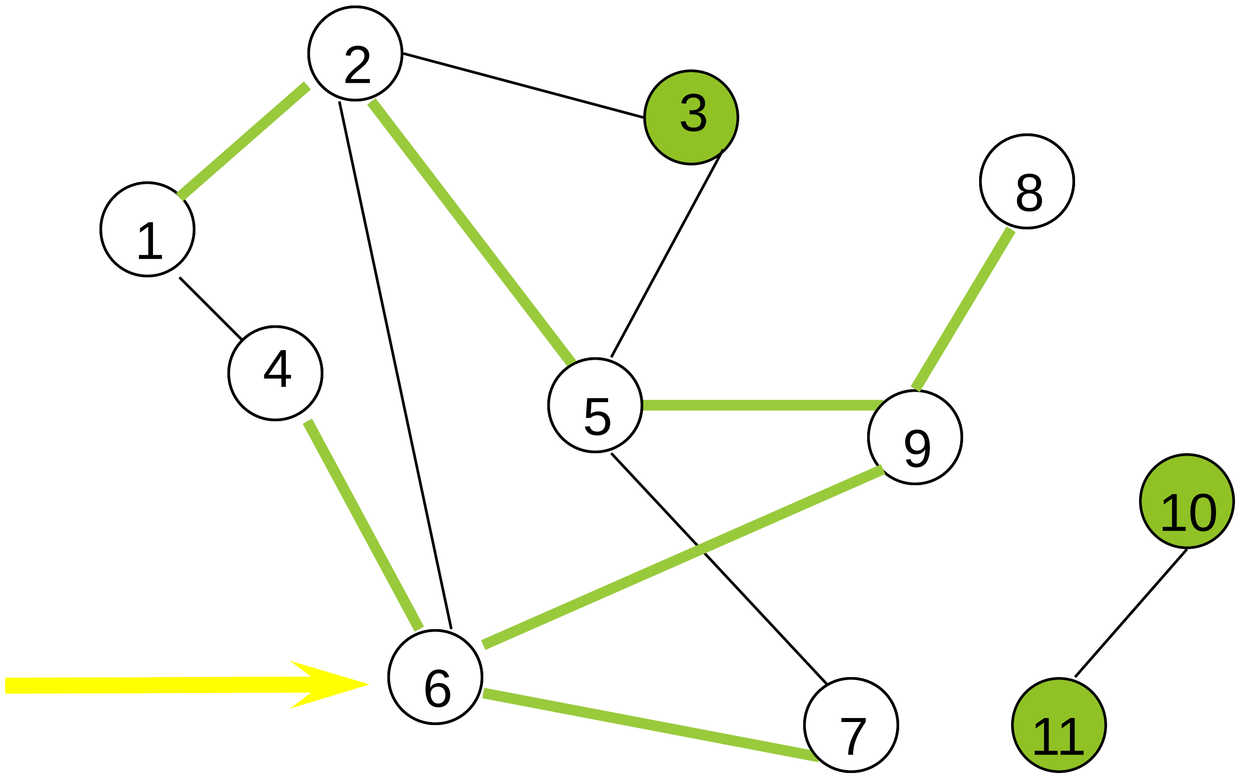
Depth-First Search Example



Label vertex 4 and return to 6.

From vertex 6 do a DFS(7).

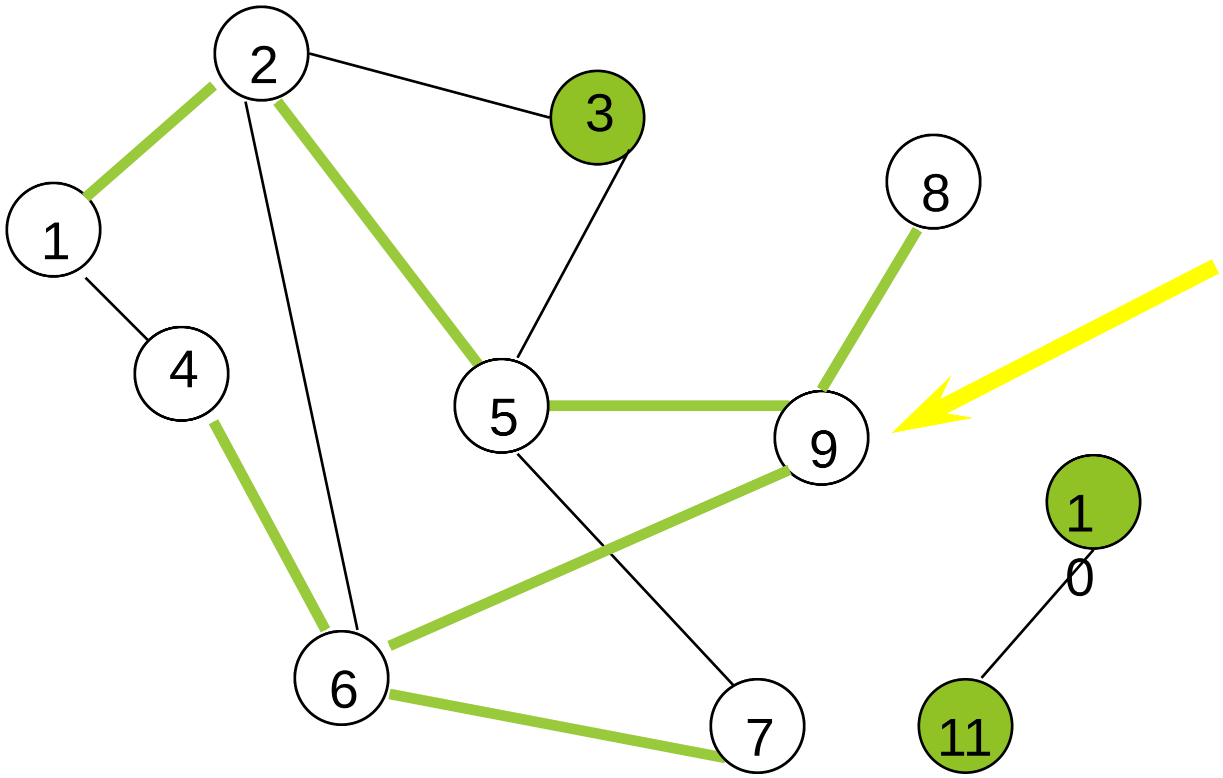
Depth-First Search Example



Label vertex 7 and return to 6.

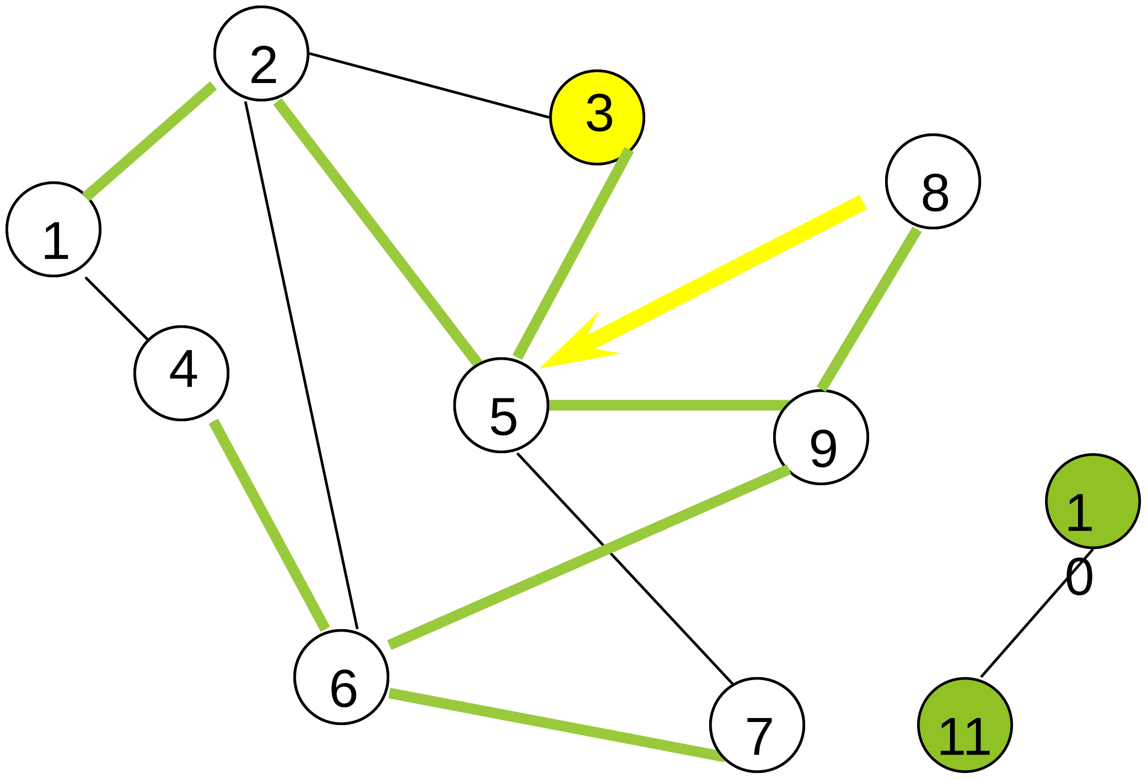
Return to 9.

Depth-First Search Example



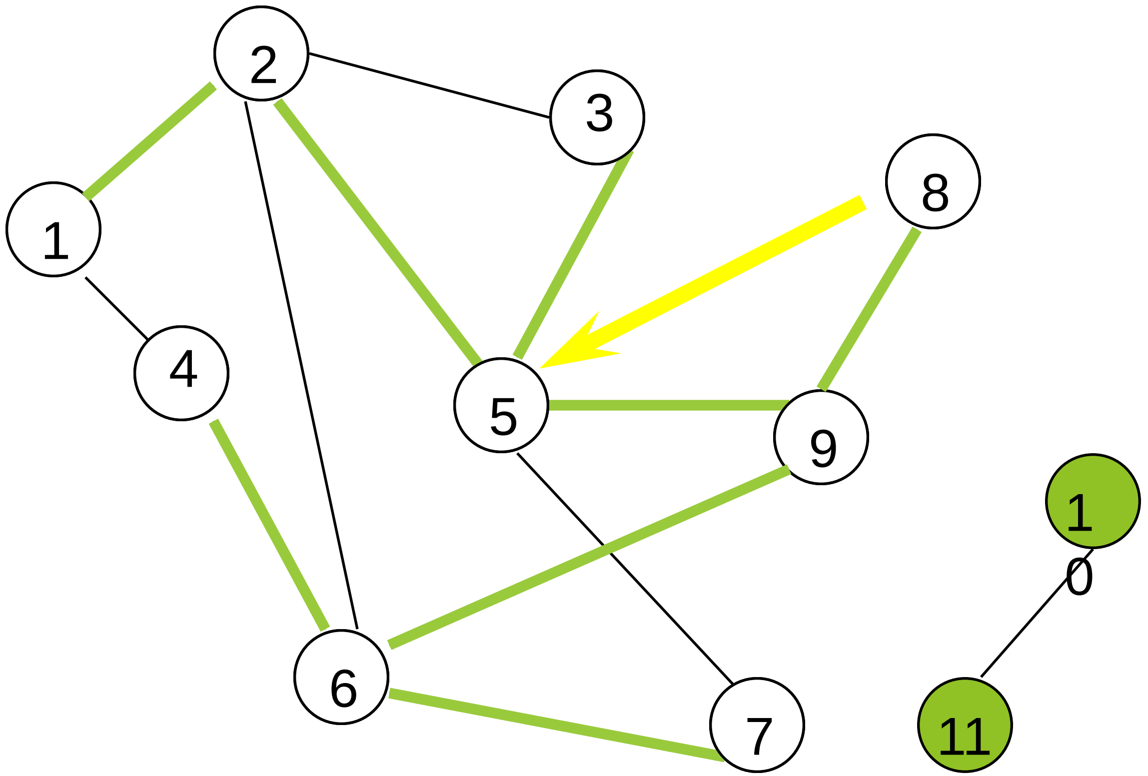
Return to 5.

Depth-First Search Example



Do a DFS(3).

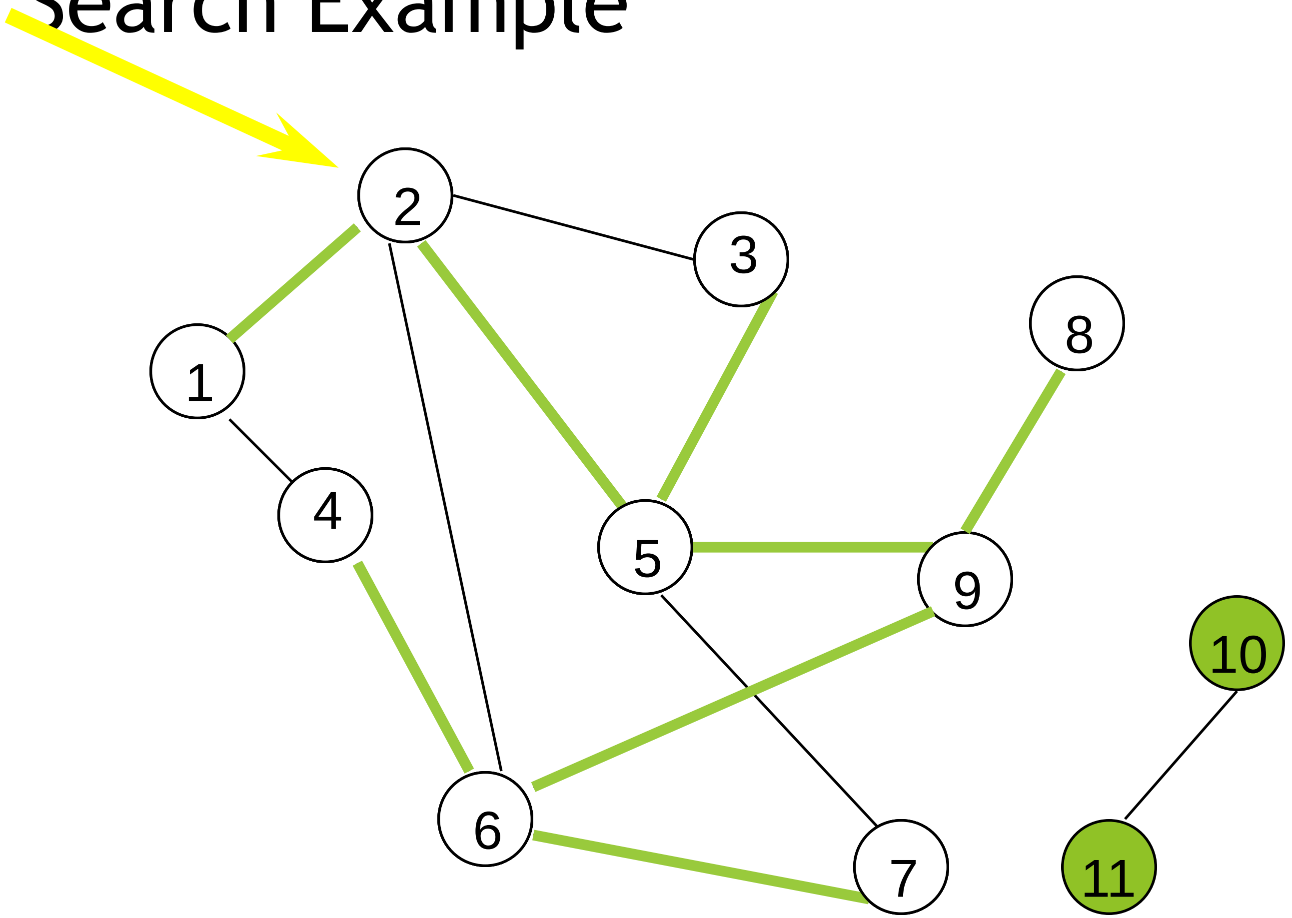
Depth-First Search Example



Label 3 and return to 5.

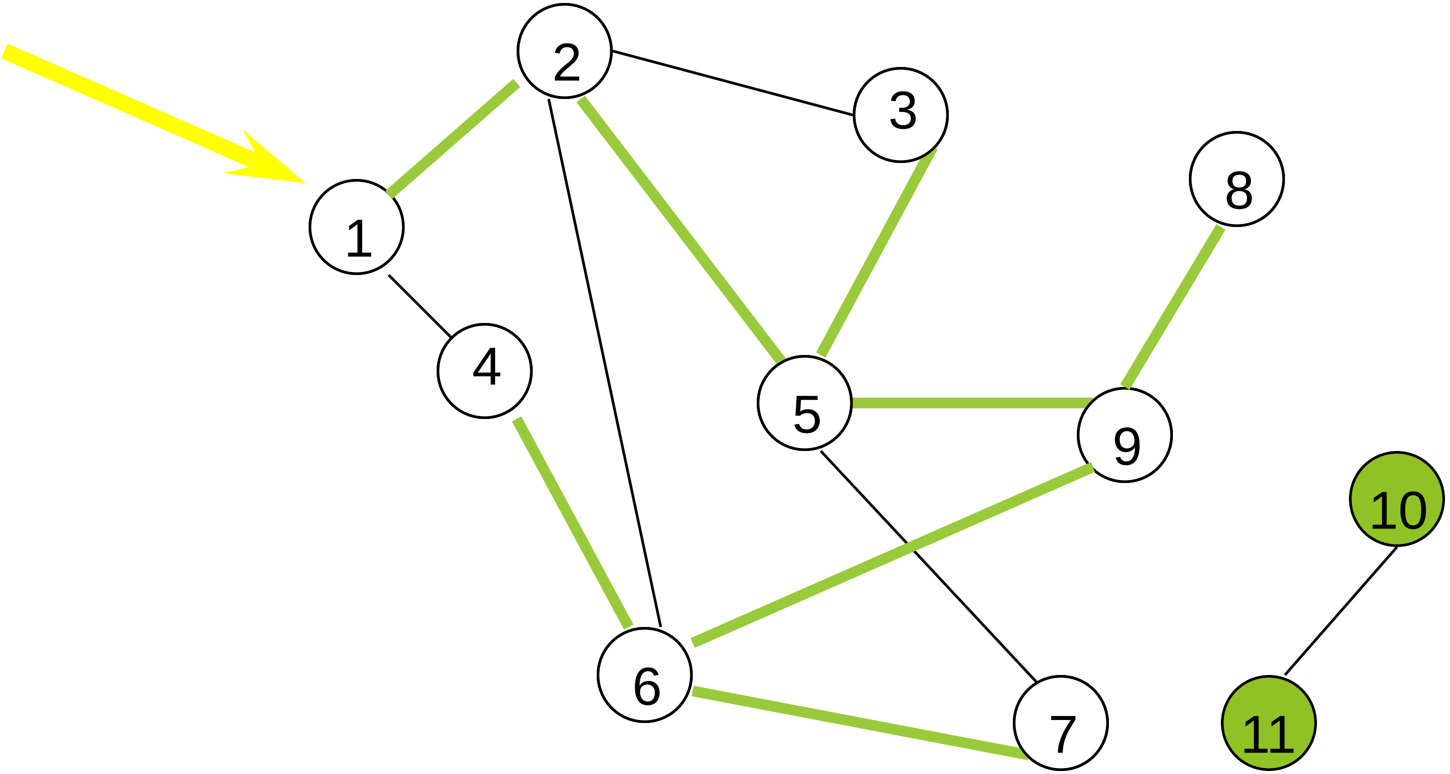
Return to 2.

Depth-First Search Example



Return to 1.

Depth-First Search Example



Return to invoking method.

Depth-First Search Properties

- Same complexity as BFS.
- Same properties with respect to path finding, connected components, and spanning trees.
- Edges used to reach unlabeled vertices define a depth-first spanning tree when the graph is connected.
- There are problems for which BFS is better than DFS and vice versa.

Thanks for your Attention

Q&A