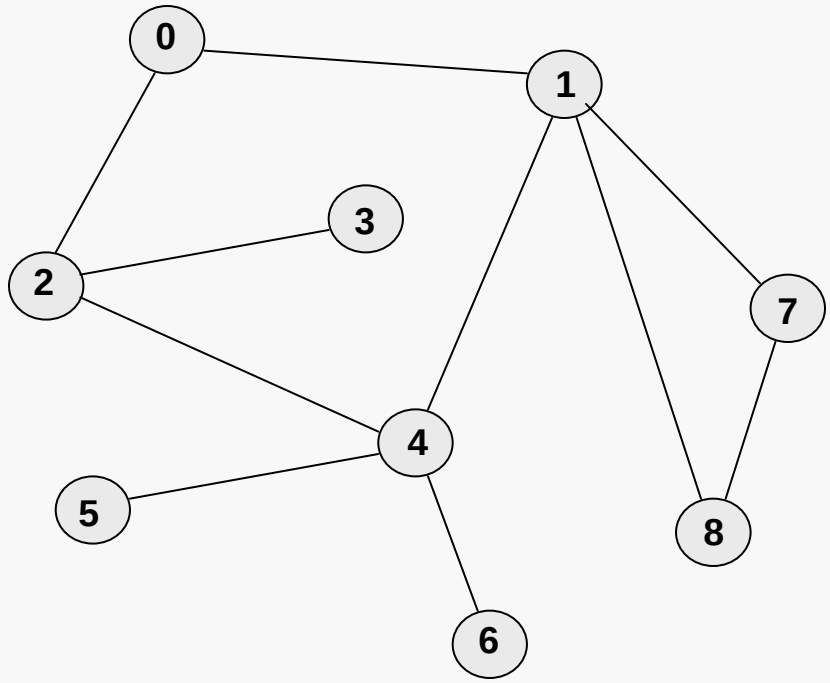
Graph Traversals

Some algorithms require that every vertex of a graph be visited exactly once.

The order in which the vertices are visited may be important, and may depend upon the particular algorithm.

The two common traversals:

- depth-first
- breadth-first



During a traversal we must keep track of which vertices have been visited; the most common approach is to provide some sort of “marking” support.

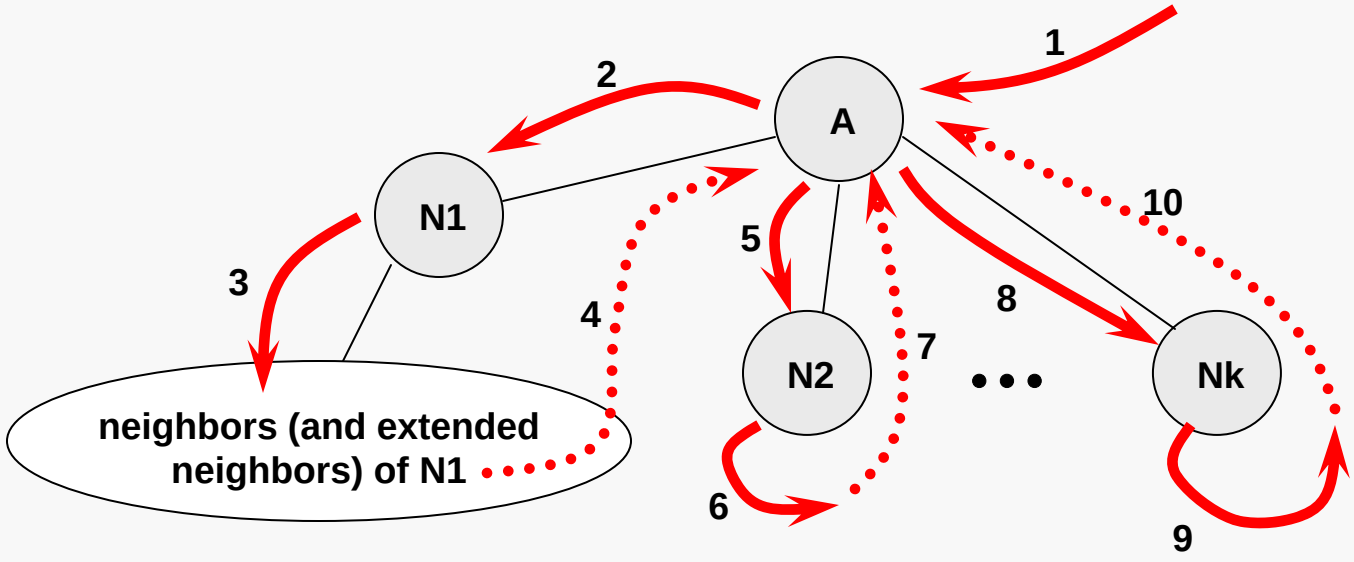
Graph Traversals: Depth-First

Assume a particular node has been designated as the starting point.

Let A be the last node visited and suppose A has neighbors N1, N2, …, Nk.

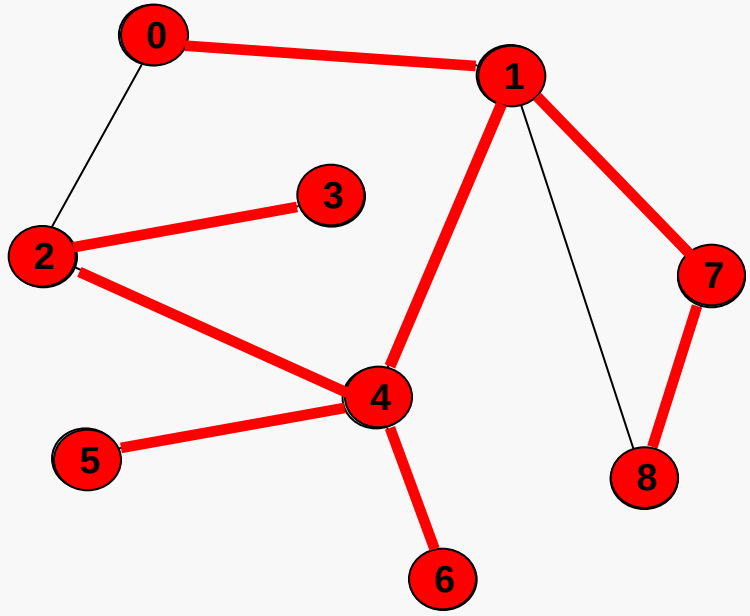
A depth-first traversal will:

- visit N1, then
- proceed to traverse all the unvisited neighbors of N1, then
- proceed to traverse the remaining neighbors of A in similar fashion.



Depth-First Traversal

If we pick node 0 as our starting point:

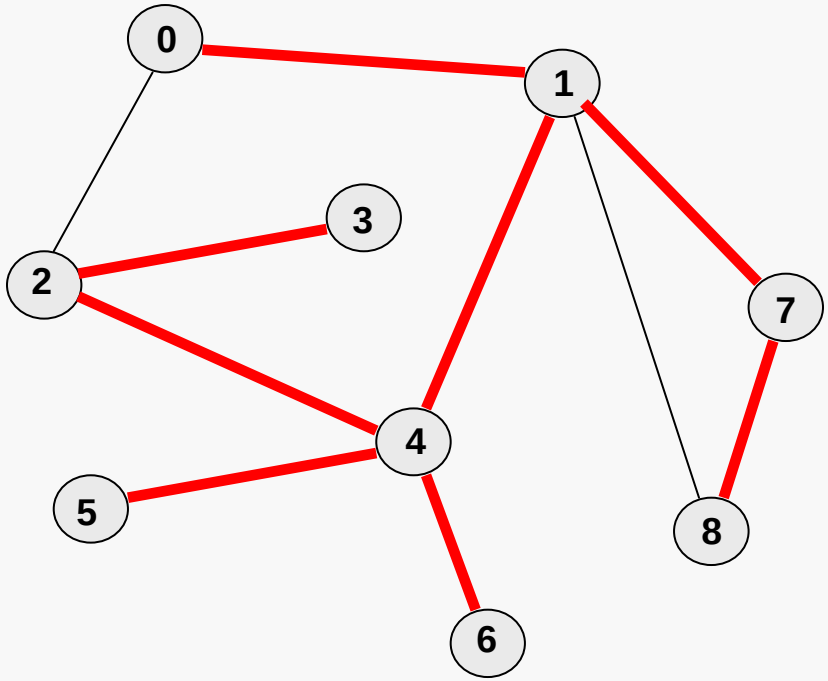



```
Visited = {0, 1, 2, 3, 4, 5, 6, 7, 8}
```

Graph Traversals: Depth-First

Assuming the node labeled **0** has been designated as the starting point, a depth-first traversal would visit the graph nodes in the order:

0 1 4 2 3 5 6 7 8



Note that if the edges taken during the depth-first traversal are marked, they define a tree (not necessarily binary) which includes all the nodes of the graph.

Such a tree is called a spanning tree for the graph.

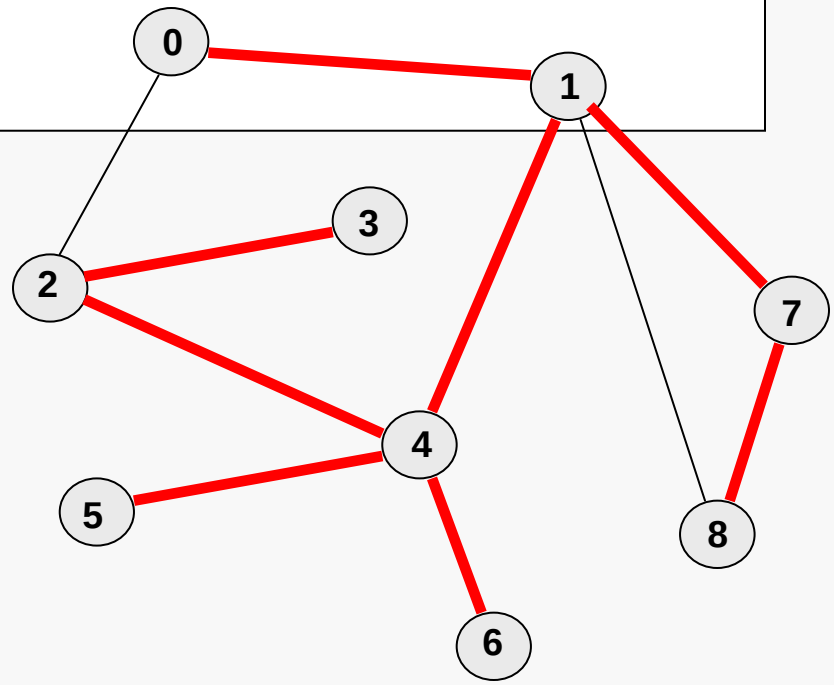
Implementing a Depth-First Traversal


```
public static void DFS(AdjMatrix G, int Start) {

   G.Mark(Start);
   for (int w = G.firstNeighbor(Start);
               G.hasEdge(Start, w); w = G.nextNeighbor(Start, w) ) {

        if ( !G.isMarked(w) ) {
           DFS(G, w);
        }
    }
}
```

If we modify `DFS()` to take another `AdjMatrix` object as a parameter, it is relatively trivial to have `DFS()` build a copy of the spanning tree.



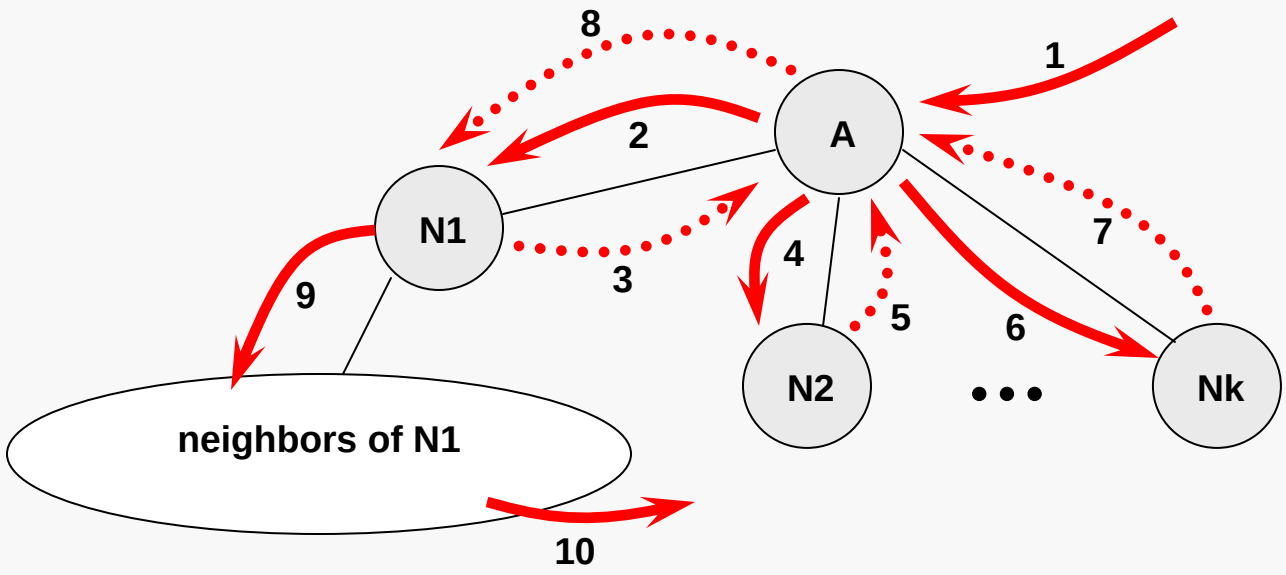
Graph Traversals: Breadth-First

Assume a particular node has been designated as the starting point.

Let A be the last node visited and suppose A has neighbors N1, N2, …, Nk.

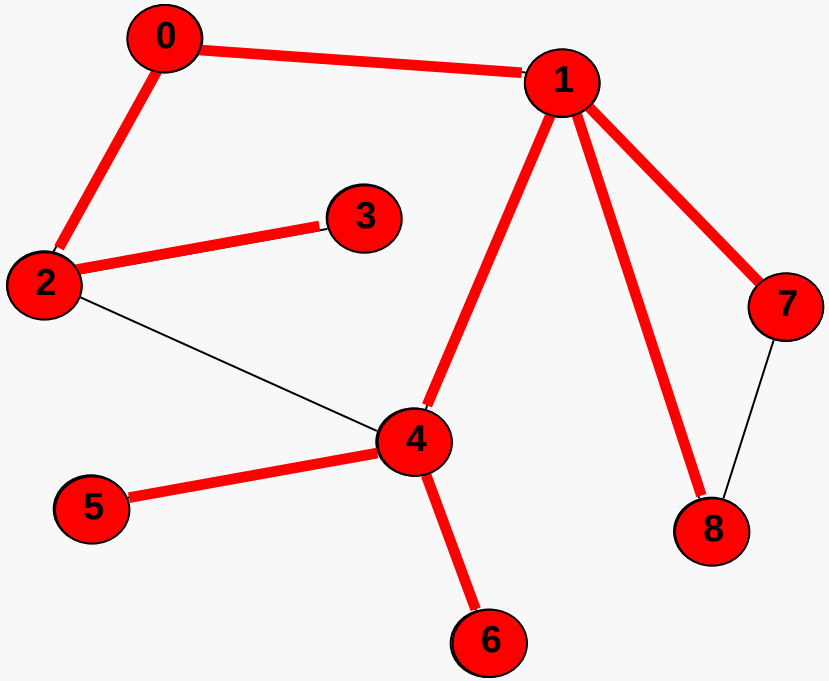
A breadth-first traversal will:

- visit N1, then N2, and so forth through Nk, then
- proceed to traverse all the unvisited immediate neighbors of N1, then
- traverse the immediate neighbors of N2, … Nk in similar fashion.



Breadth-First Traversal

If we pick node **0** as our starting point:

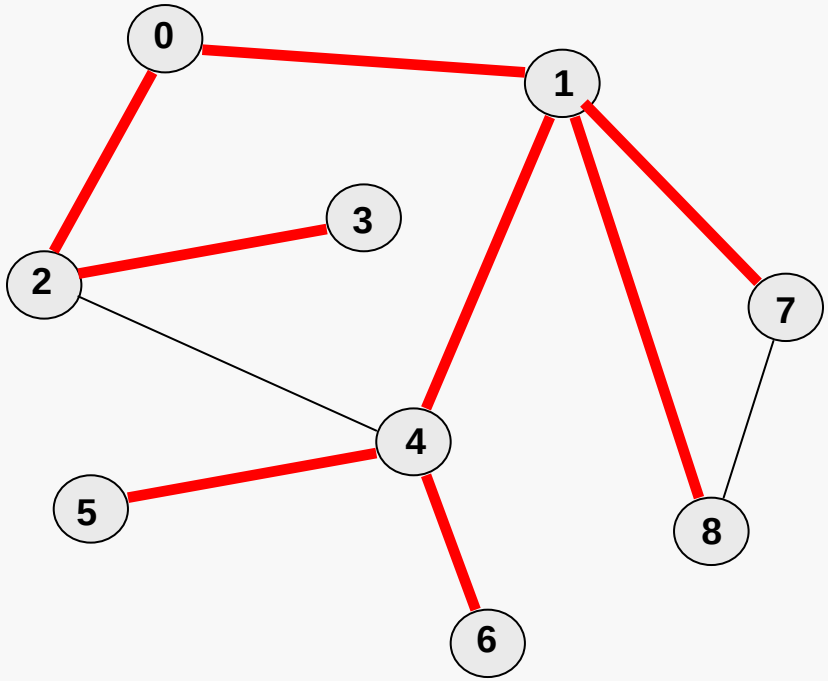


0 1 2 4 7 8 3 5 6

Graph Traversals: Breadth-First

Assuming the node labeled **a** has been designated as the starting point, a breadth-first traversal would visit the graph nodes in the order:

0 1 2 4 7 8 3 5 6

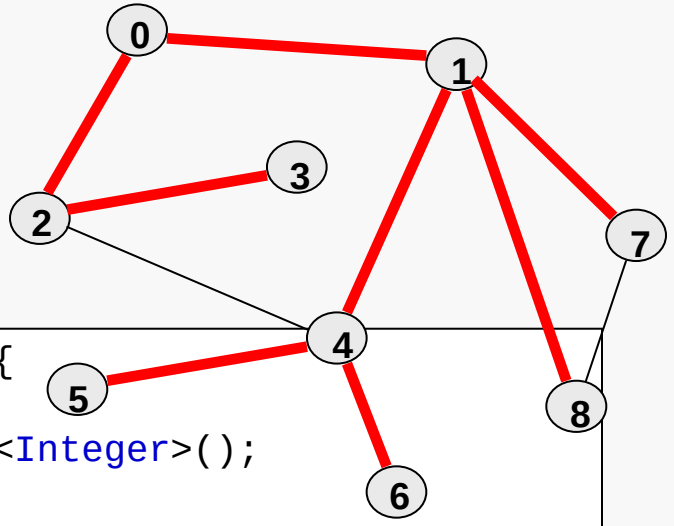


Note the edges taken during the breadth-first traversal also define a spanning tree for the given graph.

As is the case here, the breadth-first spanning tree is usually different from the depth-first spanning tree.

Implementing a Breadth-First Traversal

The breadth-first traversal uses a local queue to organize the graph nodes into the proper order:




```
public static void BFS(AdjMatrix G, int Start) {

   LinkedList<Integer> toVisit = new LinkedList<Integer>();
   toVisit.addLast(Start);
   G.Mark(Start);

   while ( !toVisit.isEmpty() ) {
       int VisitNow = toVisit.removeFirst();

       for (int w = G.firstNeighbor(VisitNow);
              G.hasEdge(VisitNow, w); w = G.nextNeighbor(VisitNow, w) ) {
           if ( !G.isMarked(w) ) {
              toVisit.addLast(w);
              G.Mark(w);
           }
       }
   }
}
```

The for loop schedules all the unvisited neighbors of the current node for future visits.