

CSCI 241

Scott Wehrwein

Graph Traversals:
Depth-First Search (Recursively)

Goals

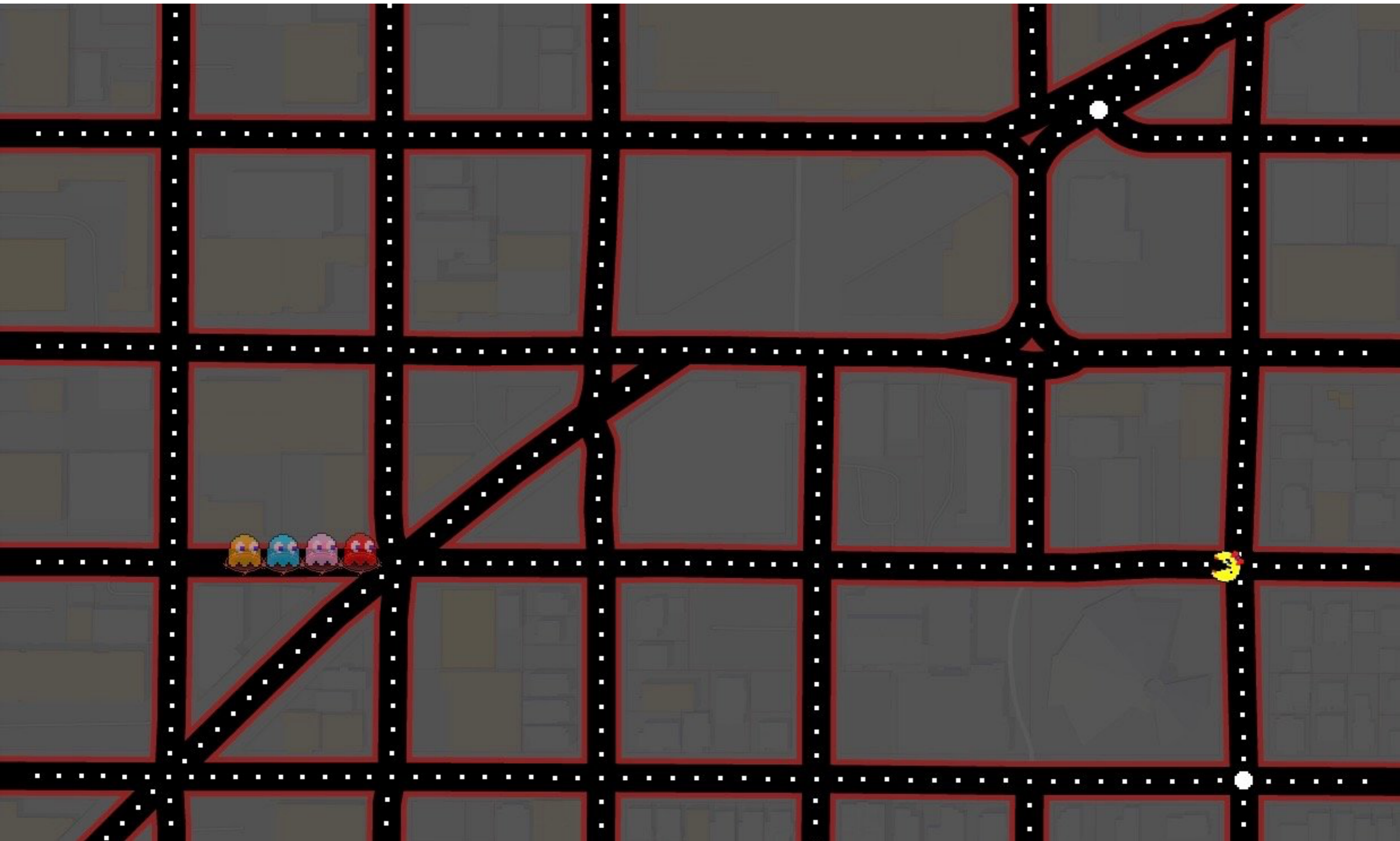
Be able to execute and implement **depth-first search** to search or traverse a graph.

Graph Algorithms

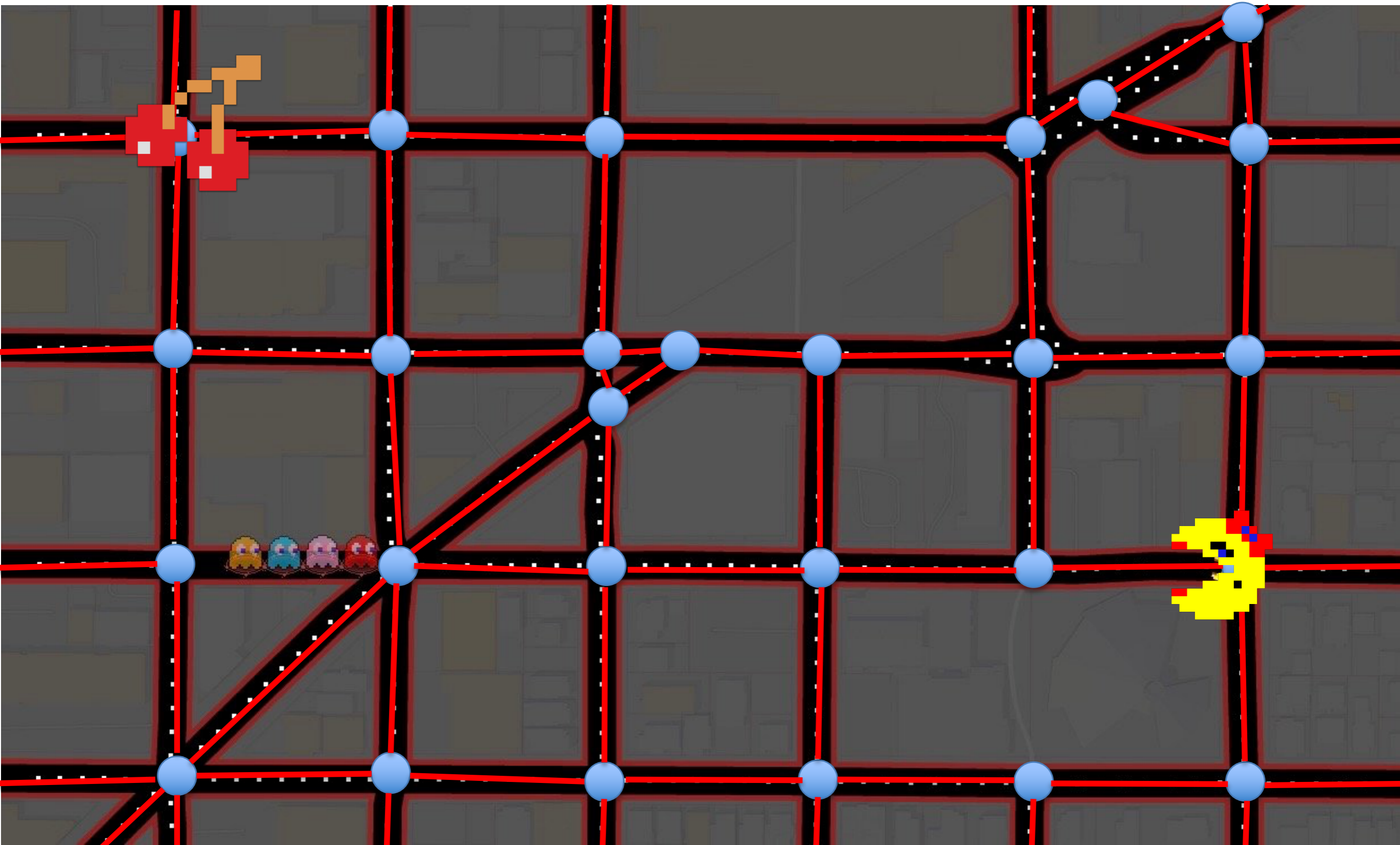
You can take entire graduate-level courses on graph algorithms. In this class:

- Search/traversal: search for a particular node or traverse all nodes (this video; Lab 6)
 - Breadth-first
 - Depth-first
- Shortest Paths (A4)
- (as time allows) Spanning trees, topological sort

Look, a graph!

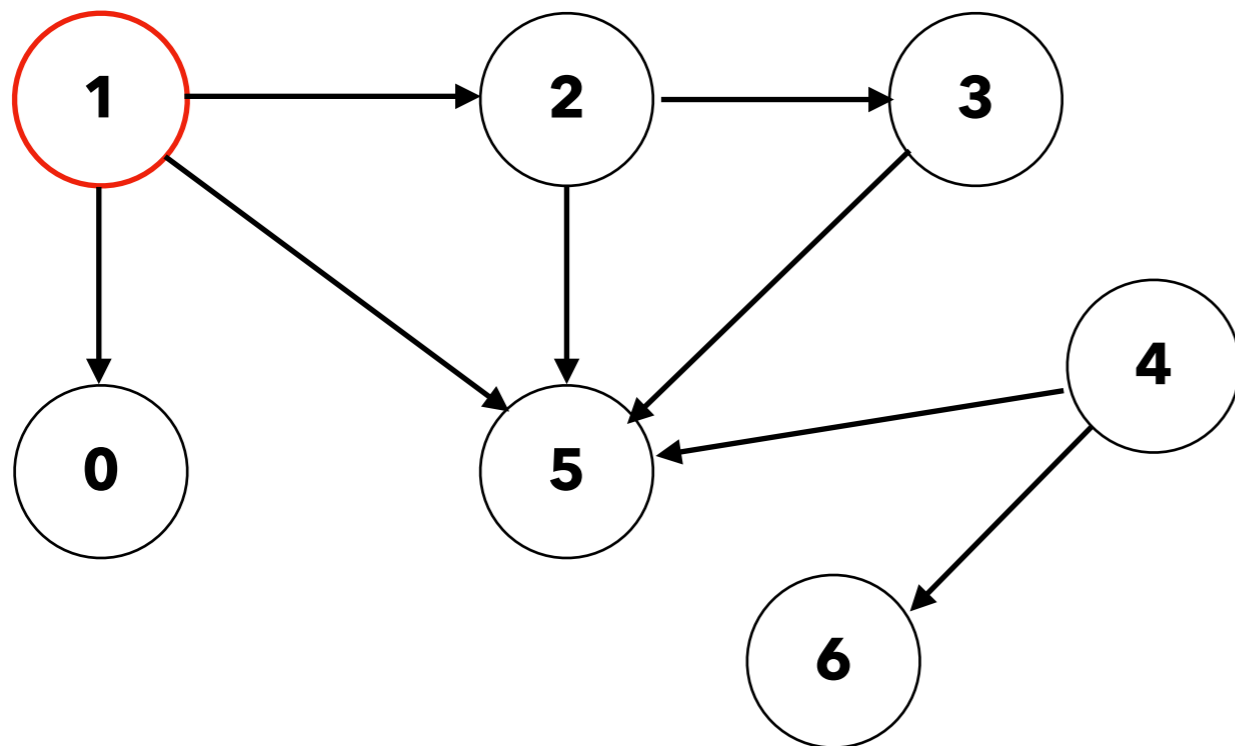


Look, a graph!



Depth-first Search

Given a graph and one of its nodes u ,
(example: node 1)



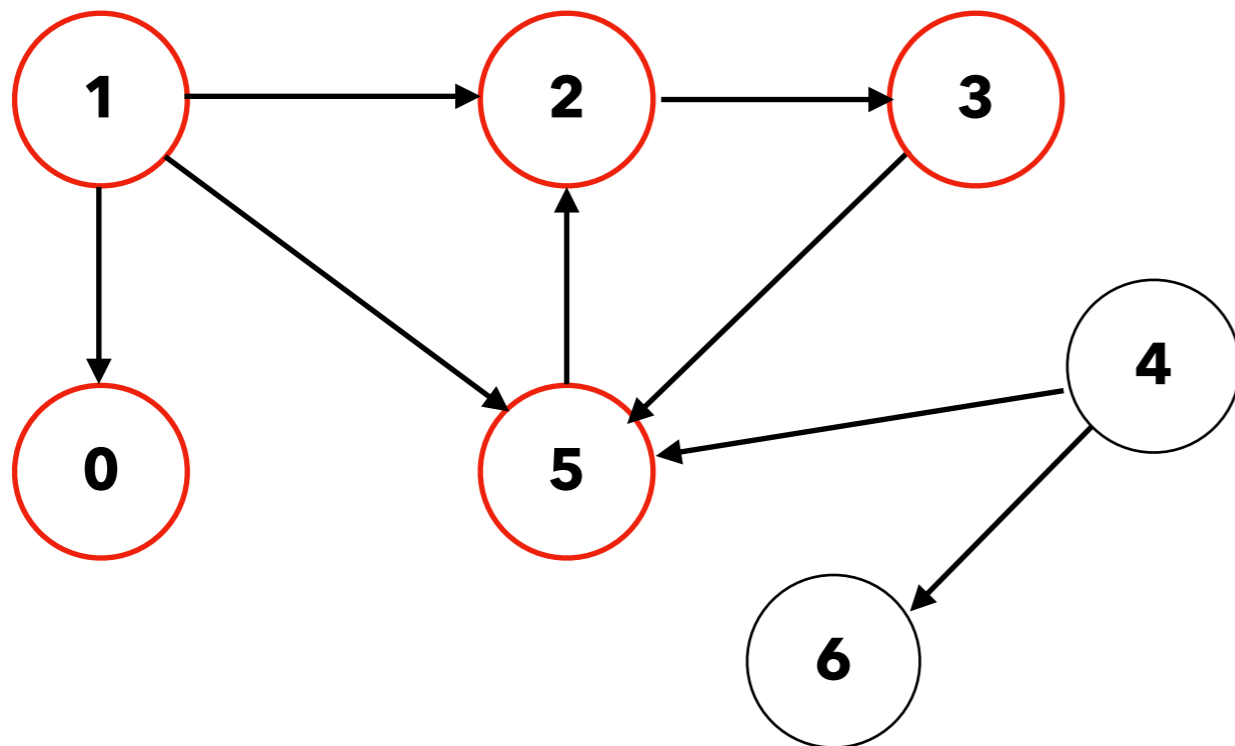
Depth-first Search

Given a graph and one of its nodes u ,

(example: node 1)

"Visit" each node **reachable** from u

(1, 0, 2, 3, 5)



Problem: multiple ways to get to the same node.

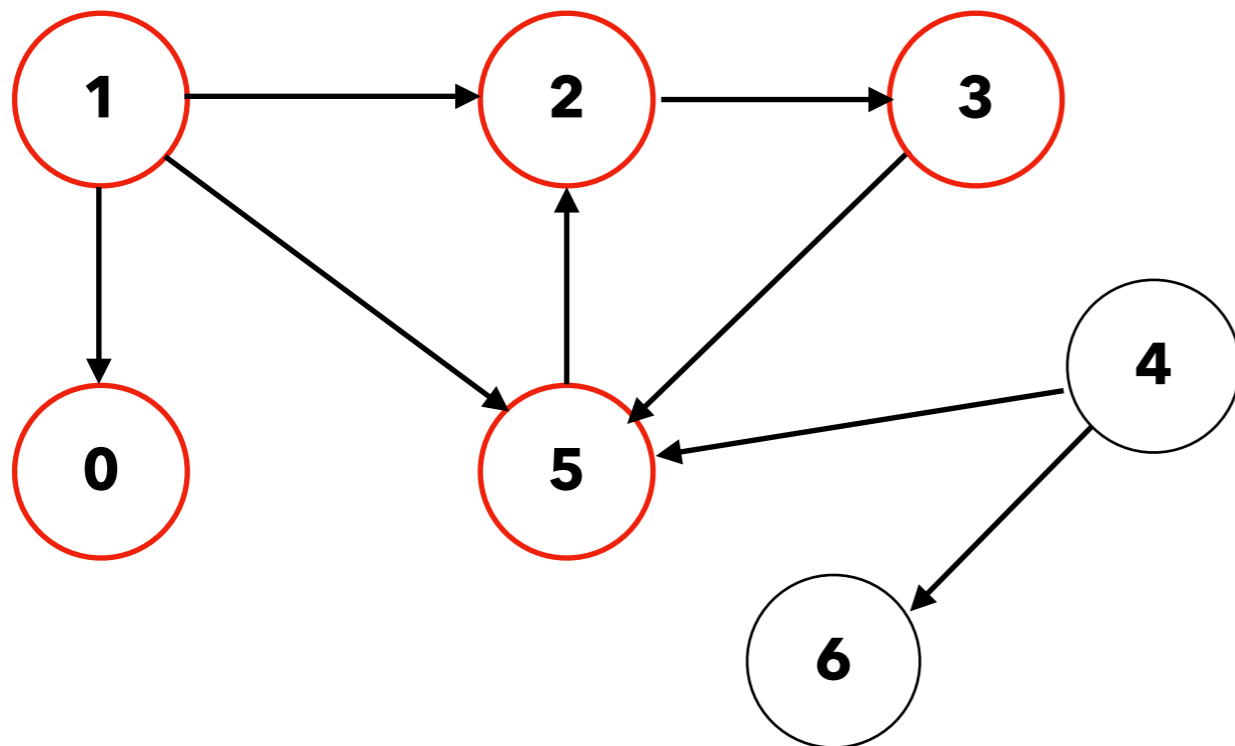
How do we visit all nodes efficiently, without doing extra work?

Key Idea: keep track of where we've been.

Depth-first Search

```
boolean visited[];
```

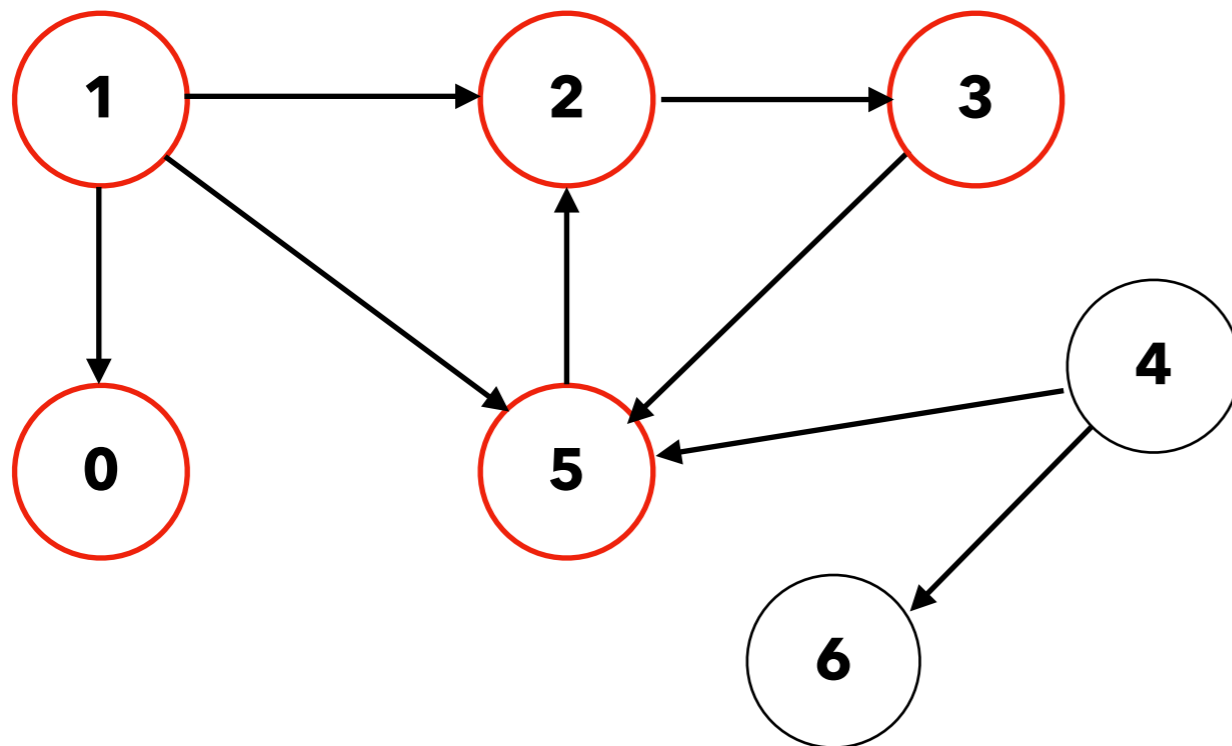
- `visited[u]` is true iff Node `u` has been visited
- Visiting `u` means setting `visited[u] = true;`
- `v` is **explorable** from `u` if there is a path (u, \dots, v) in which all nodes along the path are unvisited.



Depth-first Search

```
boolean visited[];
```

- `visited[u]` is true iff Node `u` has been visited
- Visiting `u` means setting `visited[u] = true;`
- `v` is **explorable** from `u` if there is a path (u, \dots, v) in which all nodes along the path are unvisited.



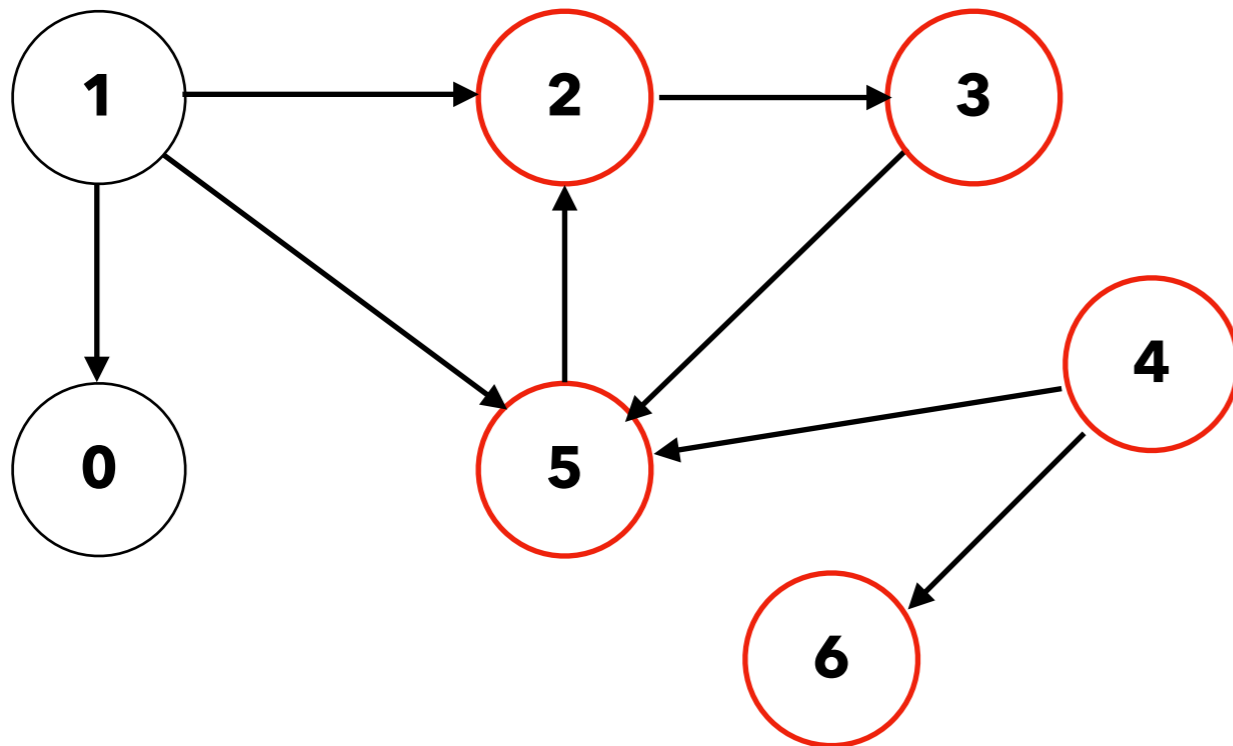
If all nodes are unvisited,

Nodes **explorable** from 1:
{1, 0, 2, 3, 5}.

Depth-first Search

```
boolean visited[];
```

- `visited[u]` is true iff Node `u` has been visited
- Visiting `u` means setting `visited[u] = true;`
- `v` is **explorable** from `u` if there is a path (u, \dots, v) in which all nodes along the path are unvisited.



If all nodes are unvisited,

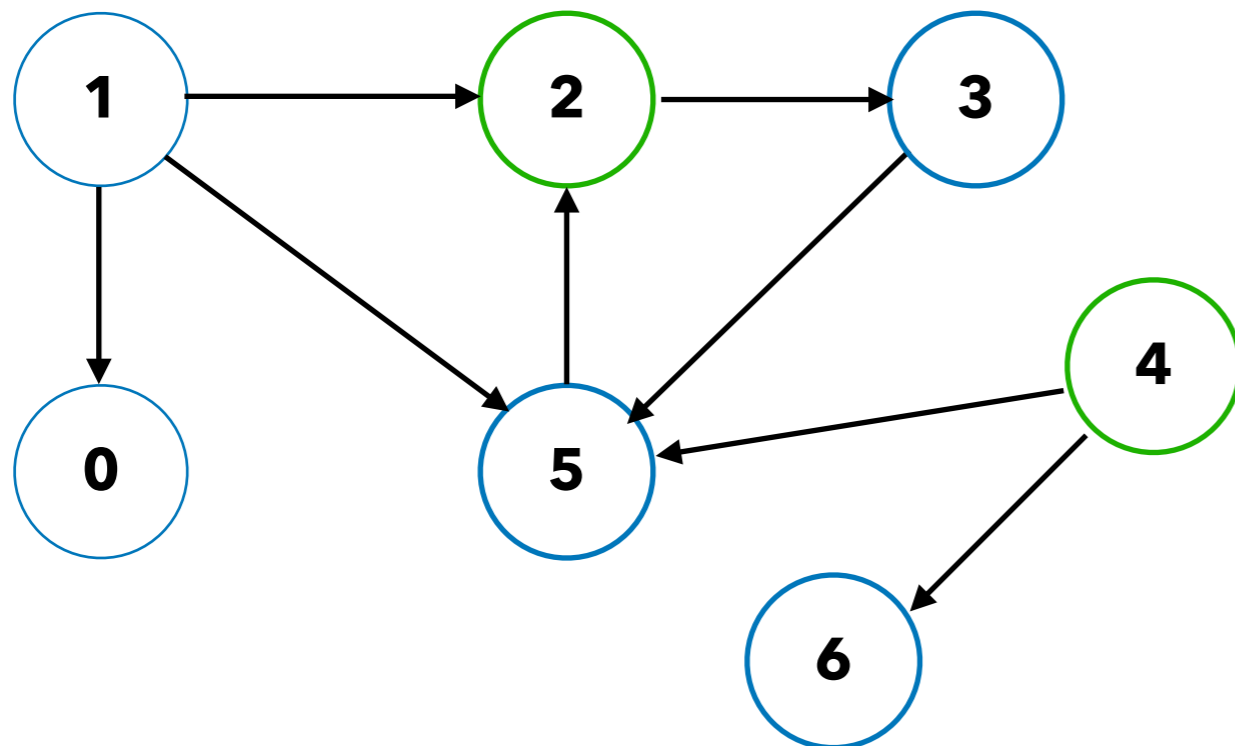
Nodes **explorable** from 1:
{1, 0, 2, 3, 5}.



Nodes **explorable** from 4:
{4, 5, 6, 2, 3}

Depth-first Search

```
boolean visited[];
```

- `visited[u]` is true iff Node `u` has been visited
- Visiting `u` means setting `visited[u] = true;`
- `v` is **explorable** from `u` if there is a path (u, \dots, v) in which all nodes along the path are unvisited.

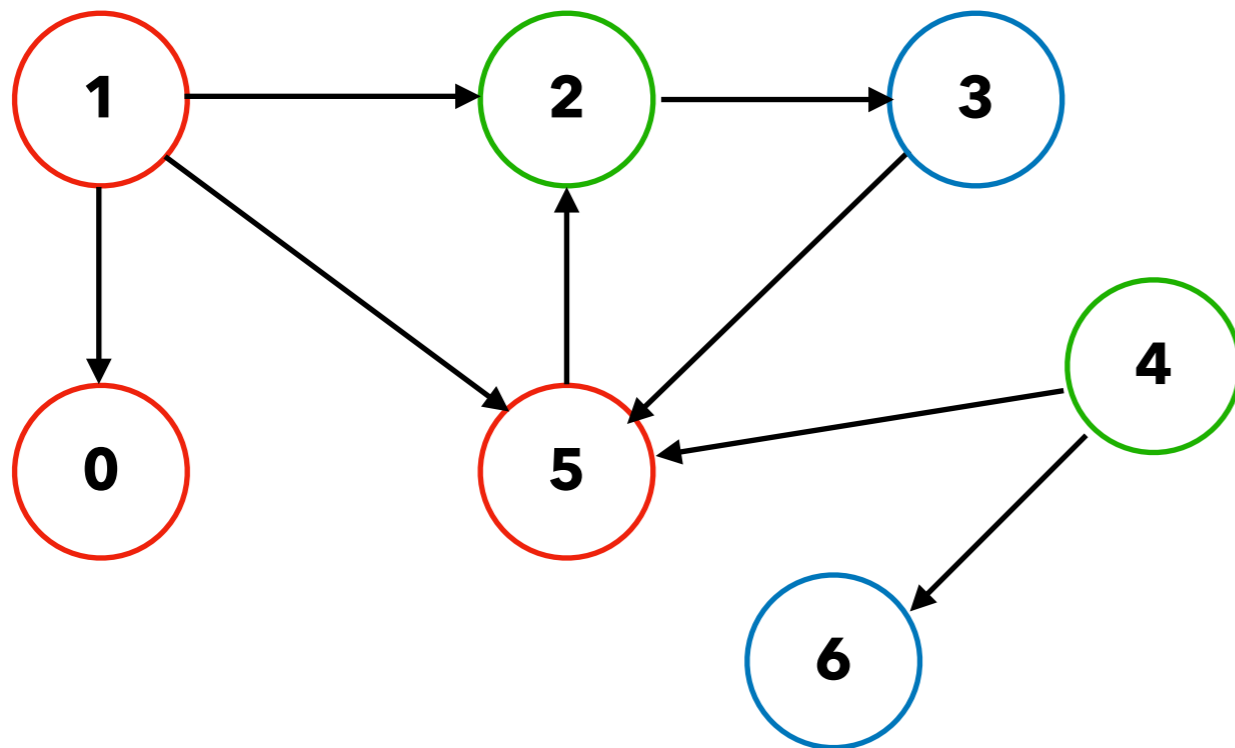




Green nodes : visited.
Blue nodes:  unvisited.

Depth-first Search

```
boolean visited[];
```

- `visited[u]` is true iff Node `u` has been visited
- Visiting `u` means setting `visited[u] = true;`
- `v` is **explorable** from `u` if there is a path (u, \dots, v) in which all nodes along the path are unvisited.



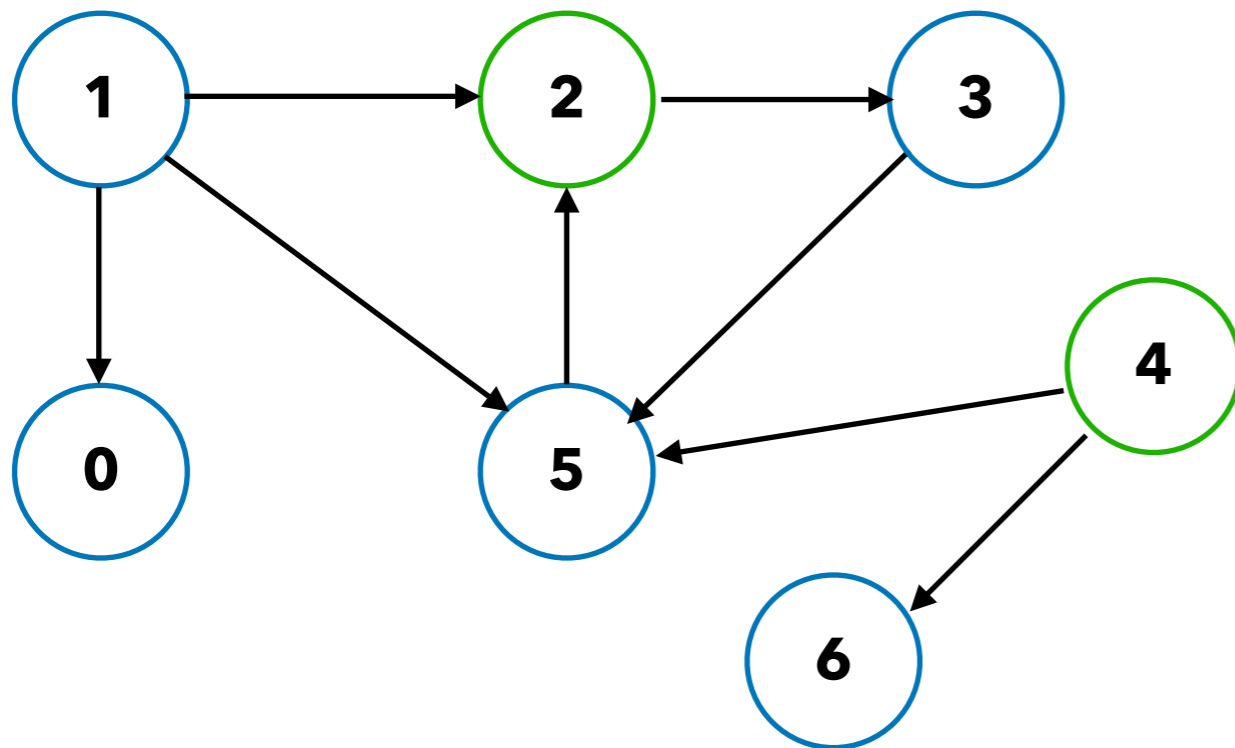
Green nodes : visited.
Blue nodes:  unvisited.



Nodes **explorable**
from 1: {1, 0, 5}.

Depth-first Search

```
boolean visited[];
```

- `visited[u]` is true iff Node `u` has been visited
- Visiting `u` means setting `visited[u] = true;`
- `v` is **explorable** from `u` if there is a path (u, \dots, v) in which all nodes along the path are unvisited.



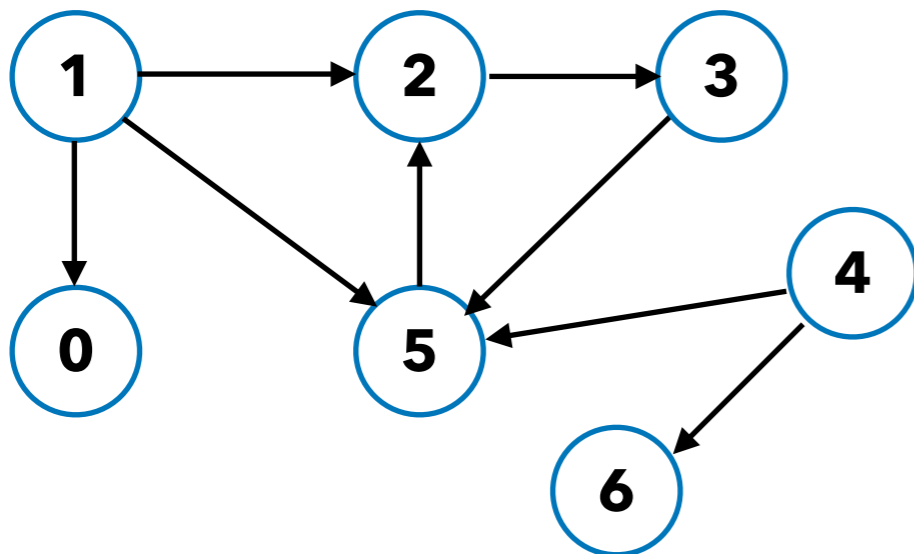
Green nodes : visited.
Blue nodes:  unvisited.

Nodes **explorable**
from 1: {1, 0, 5}.

Nodes **explorable**
from 4: {} none!

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
  
  
  
  
  
  
}
```



Depth-first Search

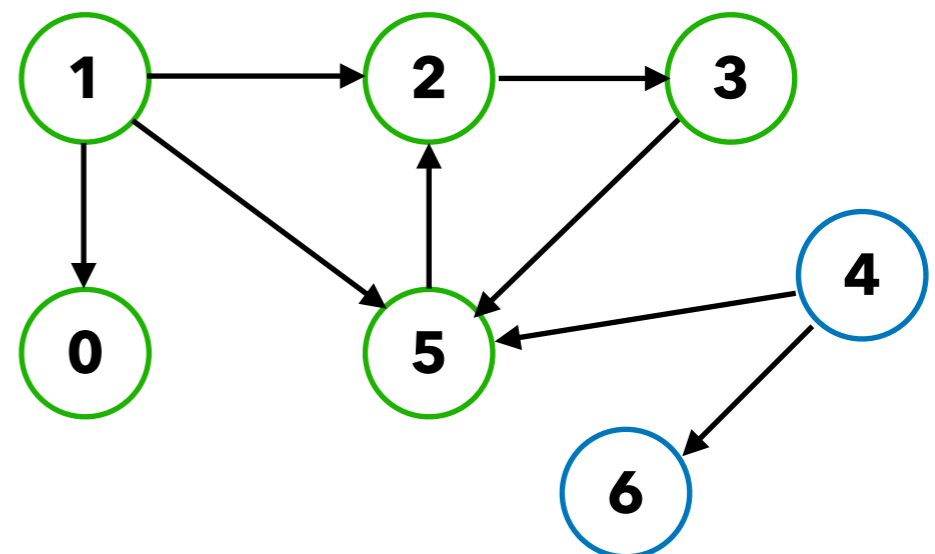
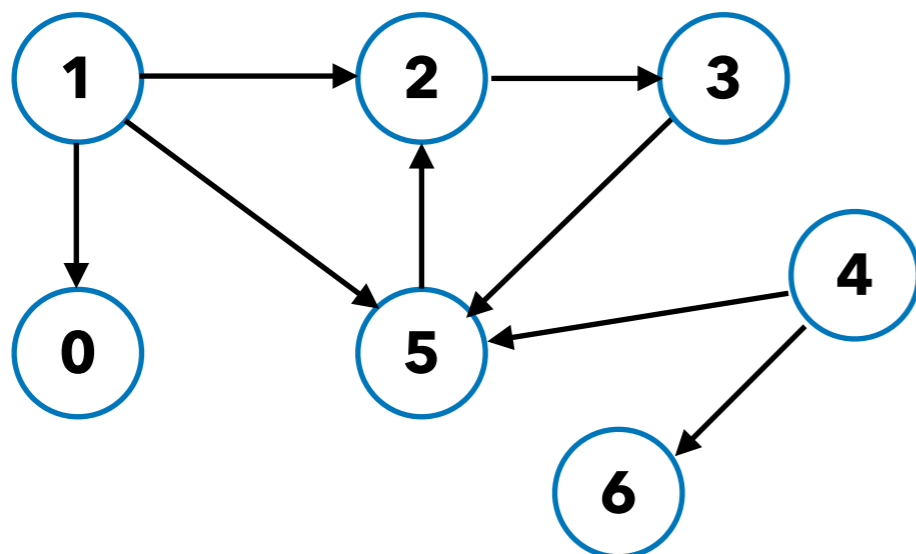
```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
  
}  
}
```

Example:

Let $u = 1$

The nodes explorable

End: from 1 are 1, 0, 2, 3, 5



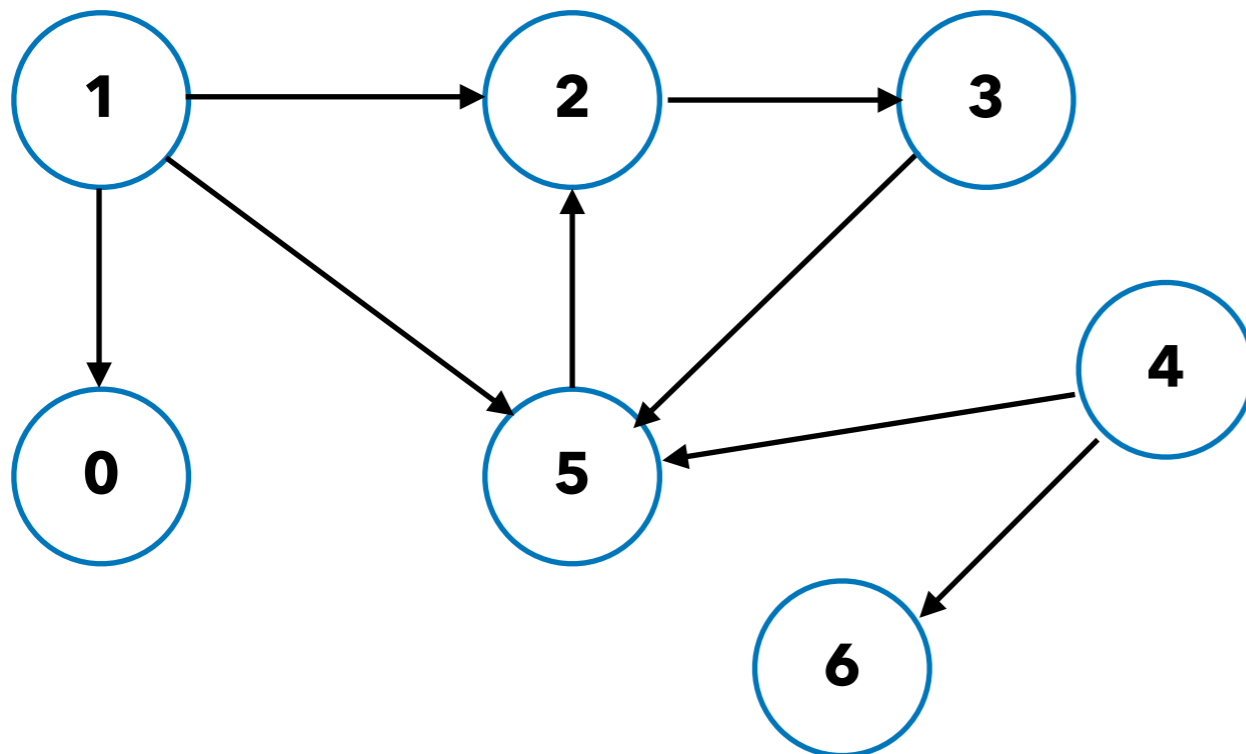
Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
  
}
```

Example:

Let $u = 1$

$dfs(1)$ The nodes explorable from 1 are 1, 0, 2, 3, 5



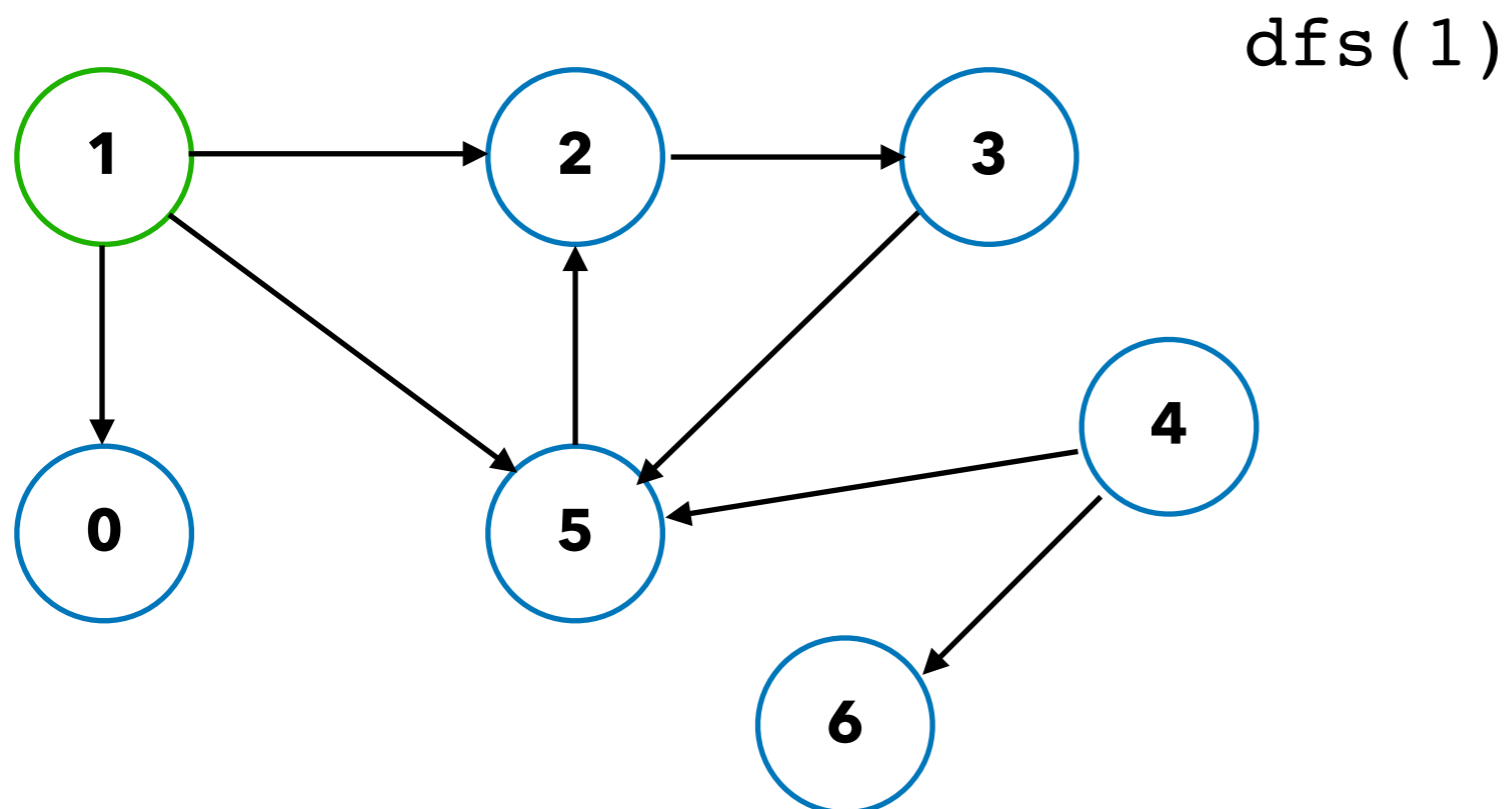
Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
  
}
```

Example:

Let $u = 1$

Nodes still to be
visited: 0, 2, 3, 5



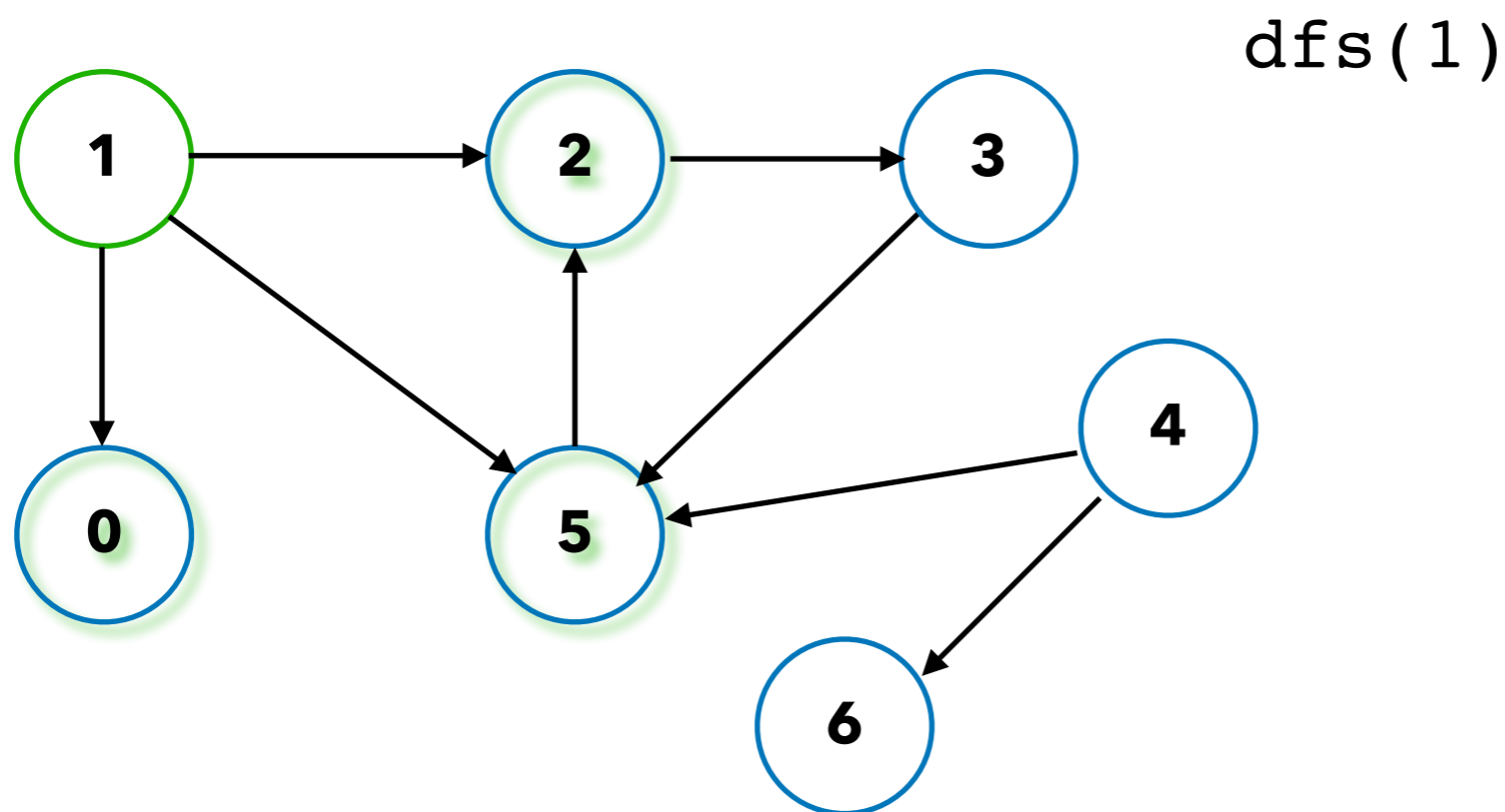
Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$

Nodes still to be
visited: 0, 2, 3, 5

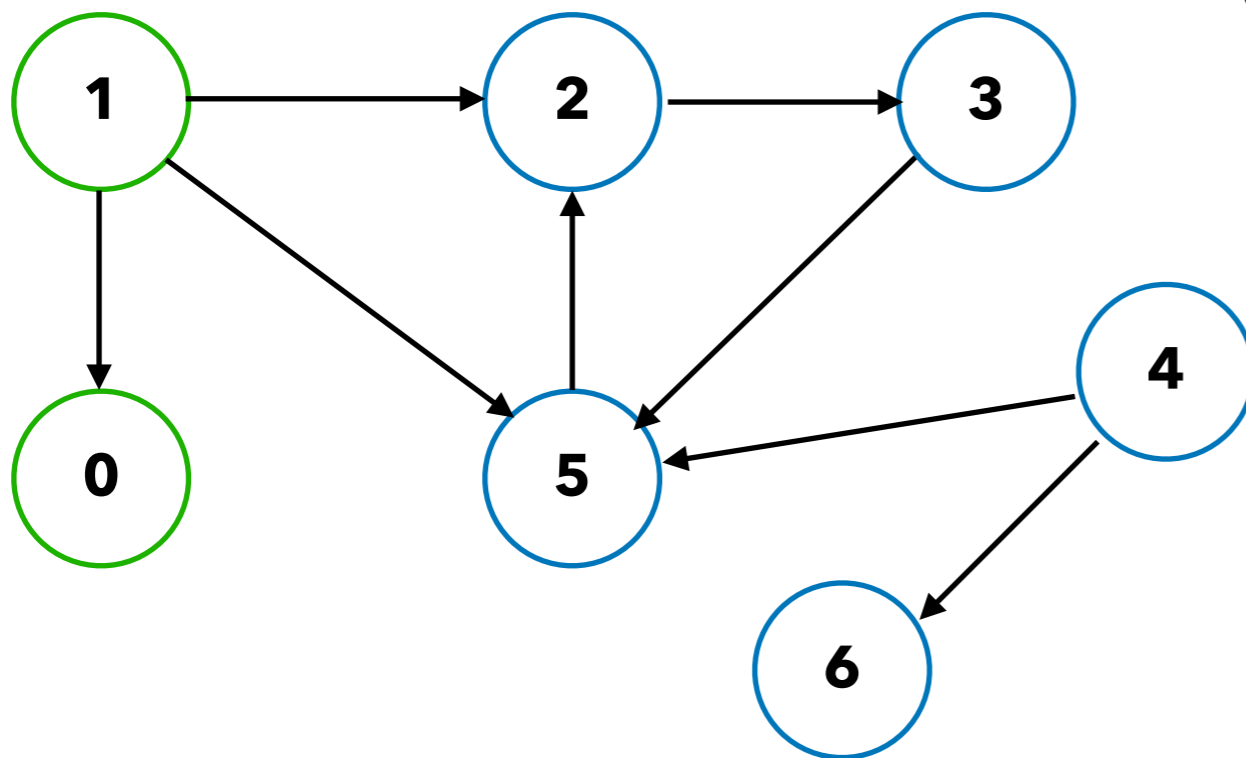
Do a DFS on all
neighbors of u !
Suppose we loop
over neighbors in
numerical order.



Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$



dfs(1)
dfs(0)

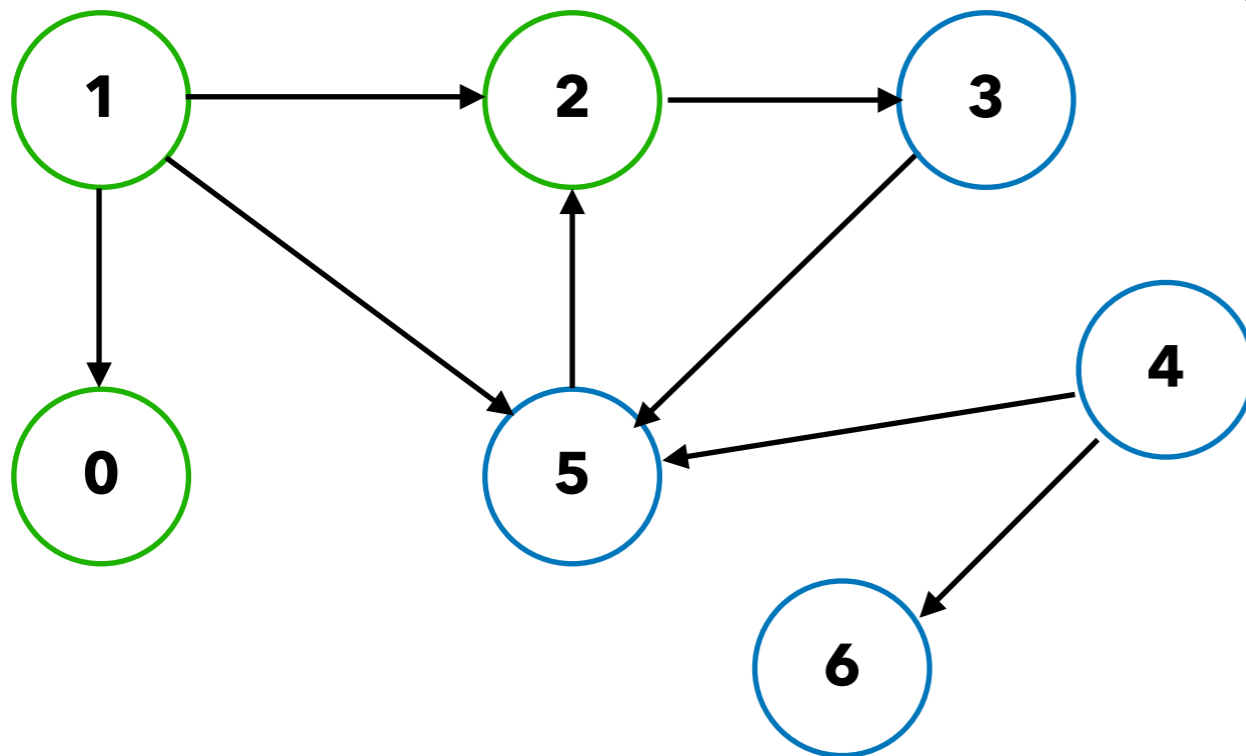
Nodes still to be
visited: 2, 3, 5

Do a DFS on all
neighbors of u !
Suppose we loop
over neighbors in
numerical order.

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$



dfs(1)
dfs(0)
dfs(2)

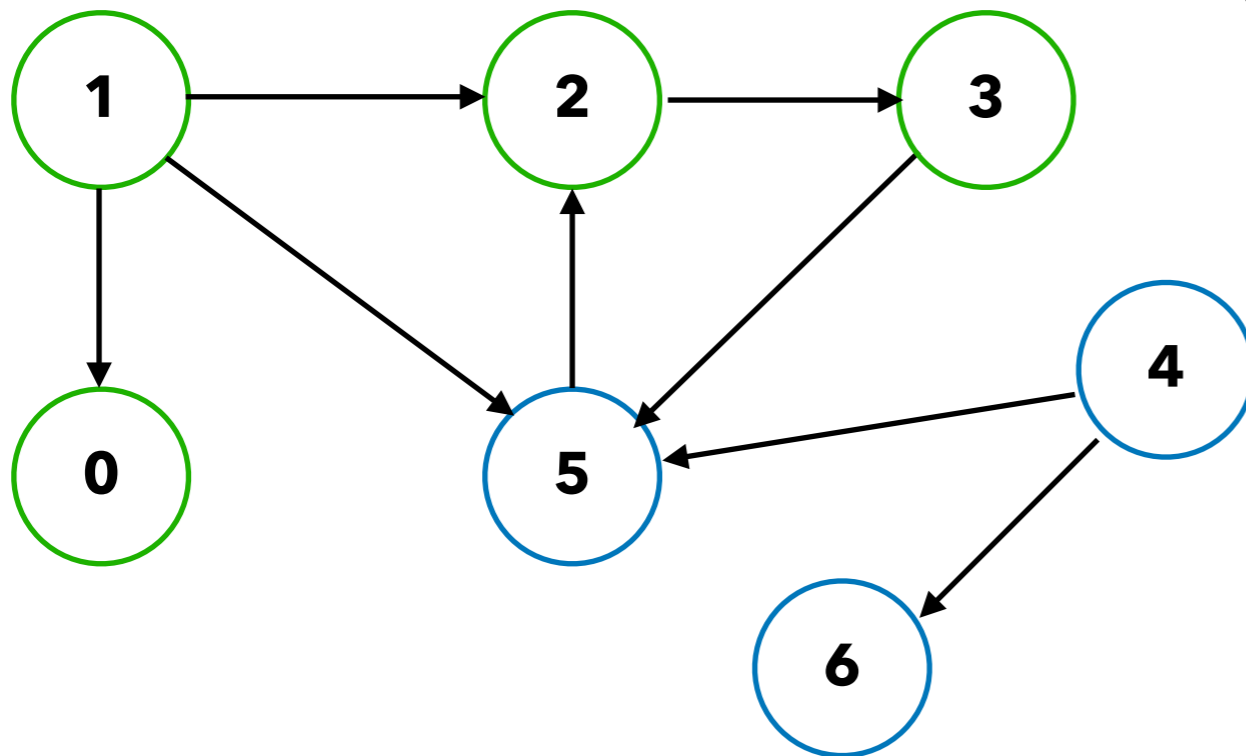
Nodes still to be visited: 3, 5

Do a DFS on all neighbors of u !
Suppose we loop over neighbors in numerical order.

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$



dfs(1)
dfs(0)
dfs(2)
dfs(3)

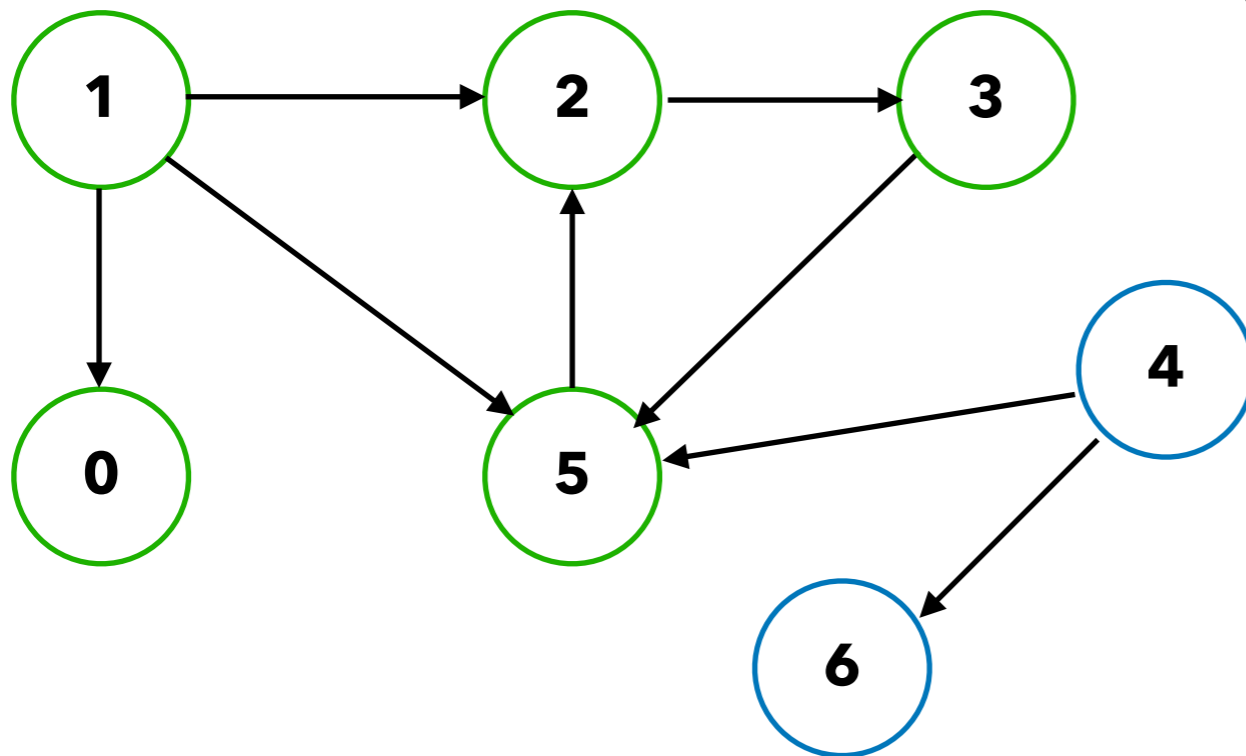
Nodes still to be
visited: 5

Do a DFS on all
neighbors of u !
Suppose we loop
over neighbors in
numerical order.

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$



dfs(1)
dfs(0)
dfs(2)
dfs(3)
dfs(5)

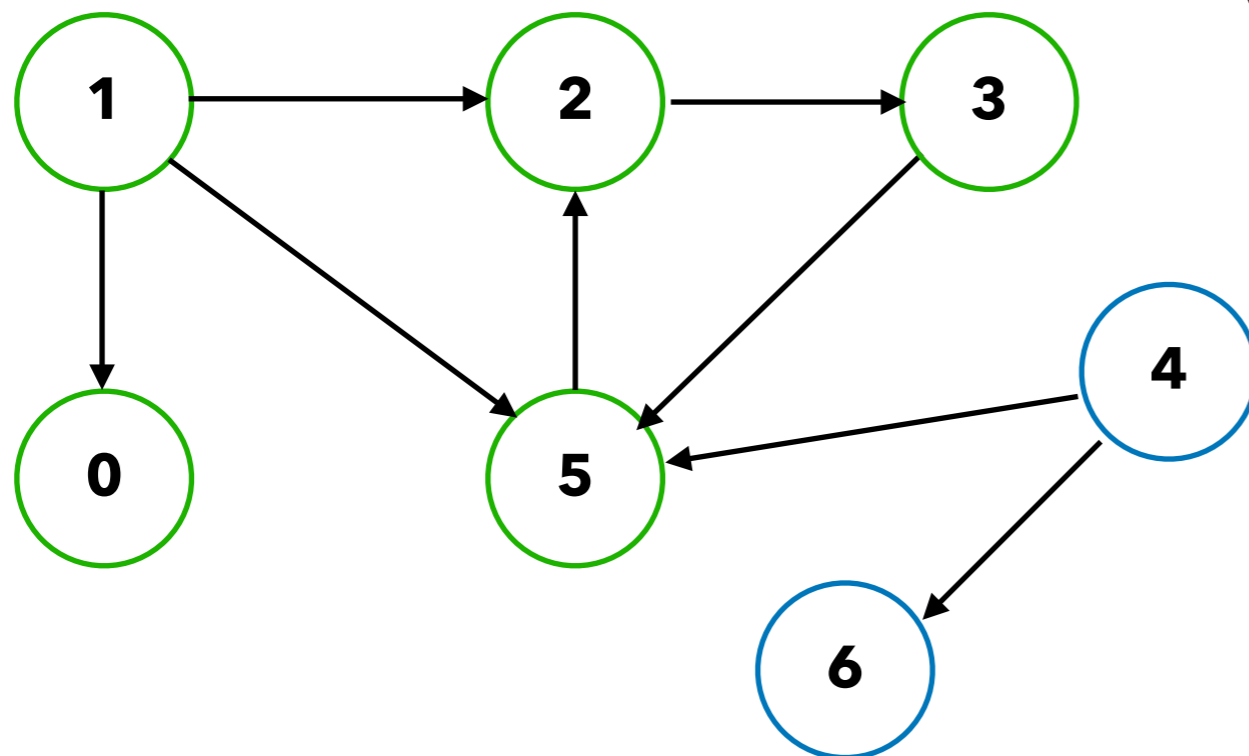
Nodes still to be visited:

Do a DFS on all neighbors of u !
Suppose we loop over neighbors in numerical order.

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$



dfs(1)
 dfs(0)
 dfs(2)
 dfs(3)
 dfs(5)
 ~~dfs(5)~~

Nodes still to be visited:

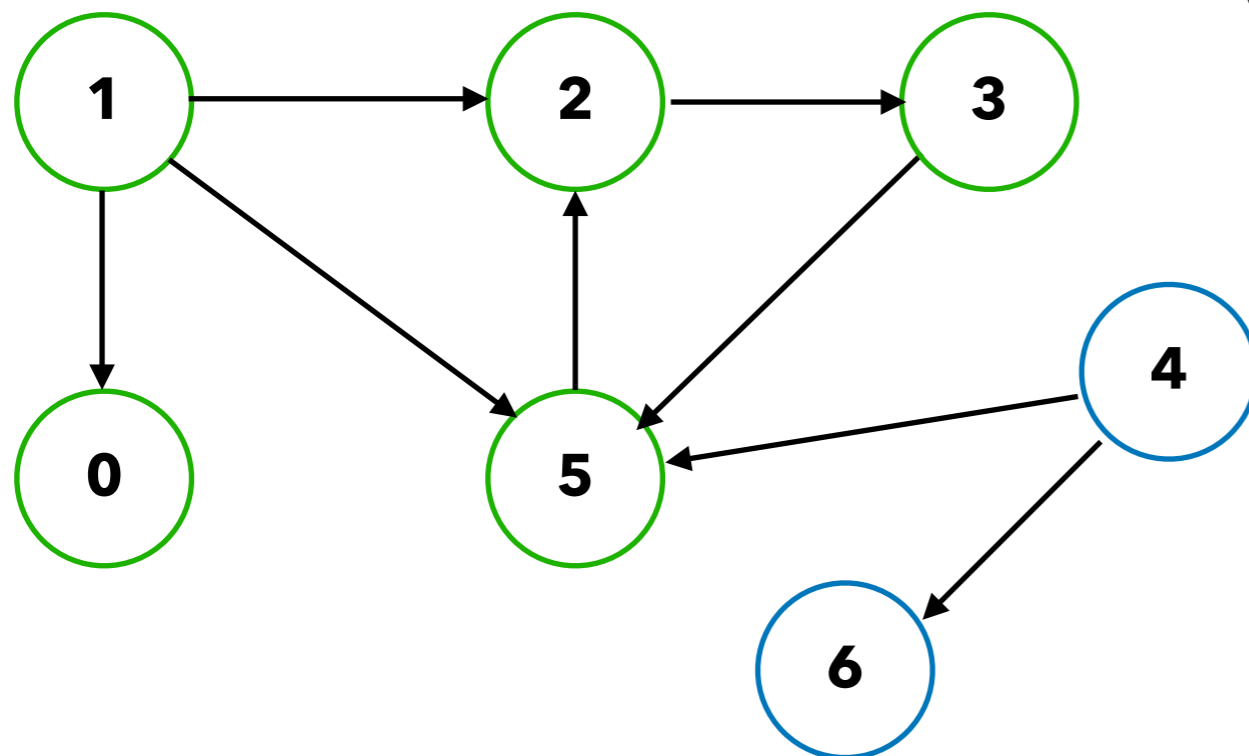
Do a DFS on all neighbors of u !

Suppose we loop over neighbors in numerical order.

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Example:
Let $u = 1$



dfs(1)
 dfs(0)
 dfs(2)
 dfs(3)
 dfs(5)
 ~~dfs(5)~~
 5 is already
 visited!

Nodes still to be
visited:

Do a DFS on all
neighbors of u !
Suppose we loop
over neighbors in
numerical order.

Depth-first Search

```
/** Visit all nodes that are explorable from u.  
 * Precondition: u is unvisited. */  
public static void dfs(int u) {  
    visited[u] = true;  
    for all edges (u, v) leaving u:  
        if v is unvisited, dfs(v);  
}
```

Usually, you want to do things in addition to just "visiting" each node, such as:

- Print the node or something about it.
- Check if it's the node you were searching for and terminate if so.