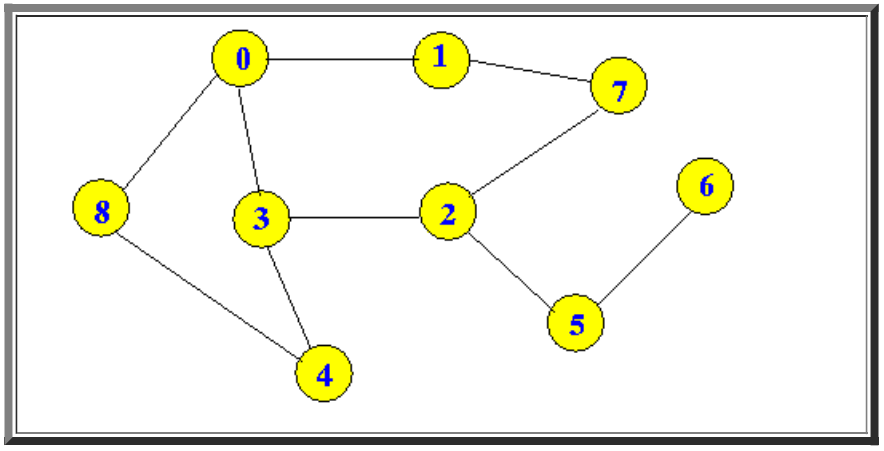
Graph traversal algorithms: Depth First Search

- **Caveat in graph traversal**
  - Unlike **trees**, a **graph** can have **cycles**:



  - We **must** maintain some **visitation information**, otherwise we will **loop forever**

- **Visitation information**
  - **visited[]:**

```
boolean visited[];        // denote whether a node has been visited
```

- **Depth First Search: go deep (before going wide)**
  - **Depth first search:**

> - **Visit** an **arbitrary node** ***x***
>
>   **Mark** node ***x*** as **visited**
>
> - Visit ***each*** **unvisited node** that is **incident to** ***x***

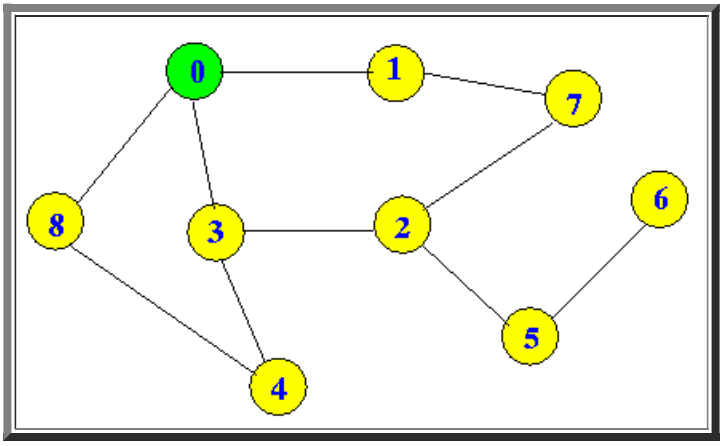
  - **Java code:**

```
public void dfs(int i)
{
   int j;

   visited[i] = true;  // Mark node as "visited"

   printNode(i);

   for ( j = 0; j < NNodes; j++ )
   {
      if ( adjMatrix[i][j] > 0 && !visited[j] )
      {
         dfs(j);       // Visit node
      }
```

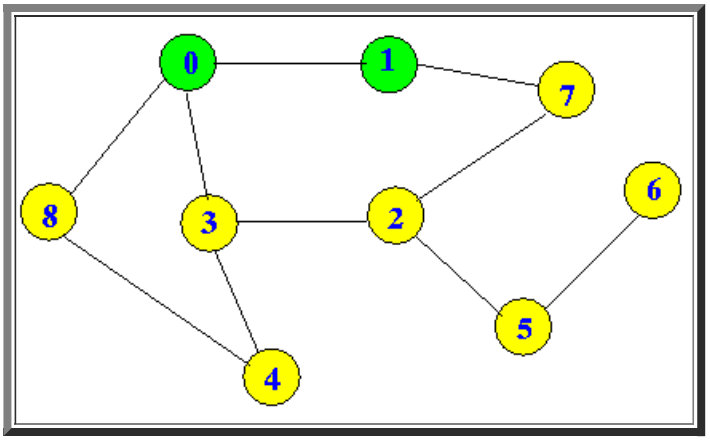
```
    }
}
```

- **Example:** dfs(0)

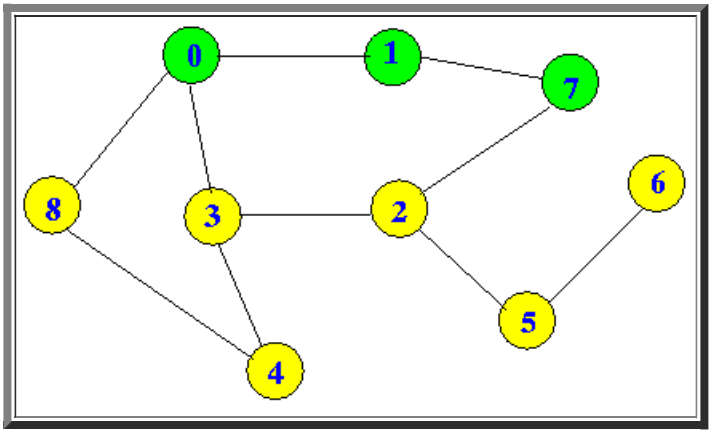
  - dfs(0):



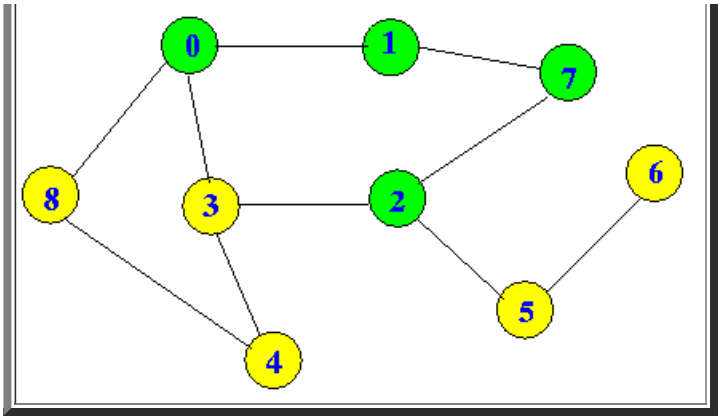
  - dfs(0) → dfs(1)



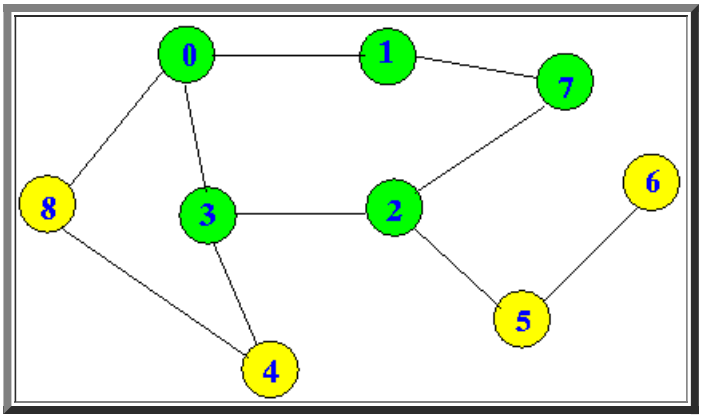
  - ~~dfs(1) → dfs(0)~~ (because node 0 is "visited"); dfs(1) → dfs(7)



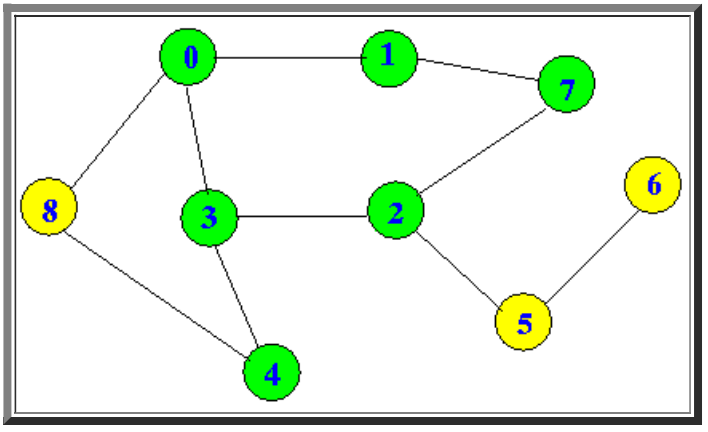
  - ~~dfs(7) → dfs(1)~~; dfs(7) → dfs(2)



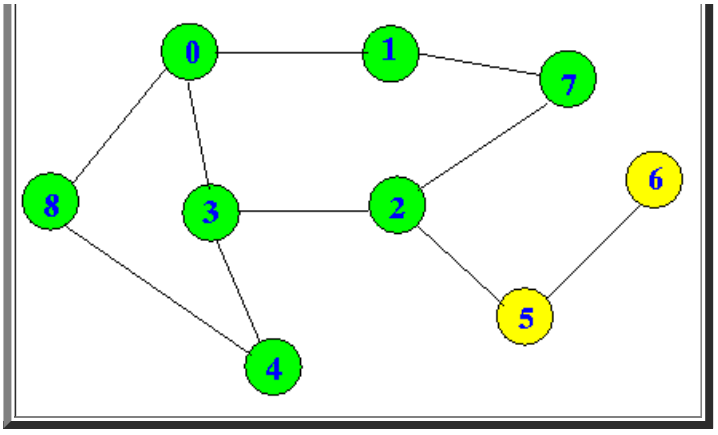
- dfs(2) → dfs(3)



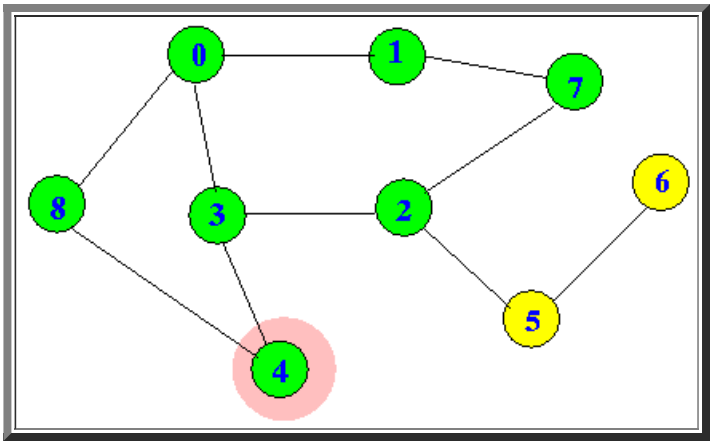
- ~~dfs(3) → dfs(0)~~; ~~dfs(3) → dfs(2)~~; dfs(3) → dfs(4)



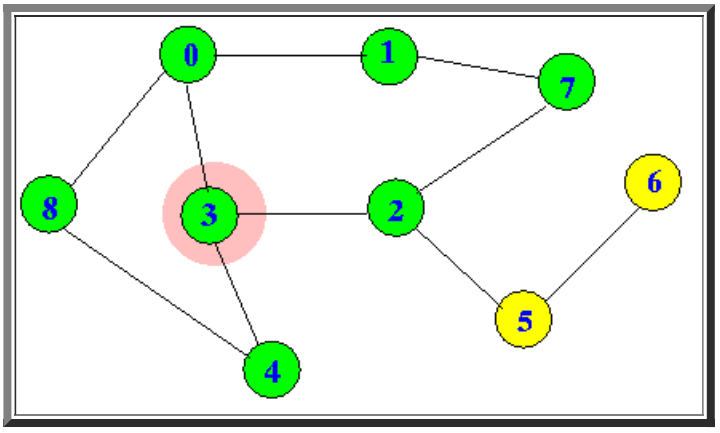
- ~~dfs(4) → dfs(3)~~; dfs(4) → dfs(8)



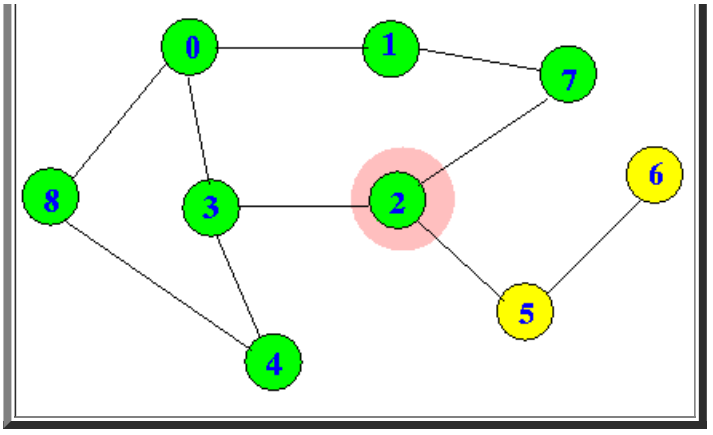
- ~~dfs(8) → dfs(0)~~; ~~dfs(8) → dfs(4)~~; **return to dfs(4)**



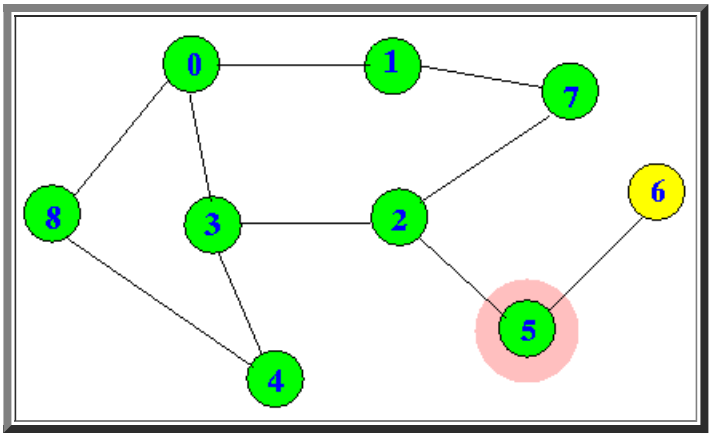
- **return to dfs(3)**



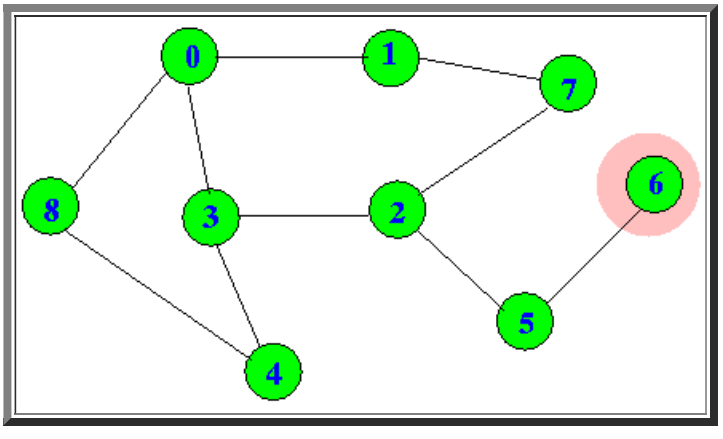
- **return to dfs(2)**



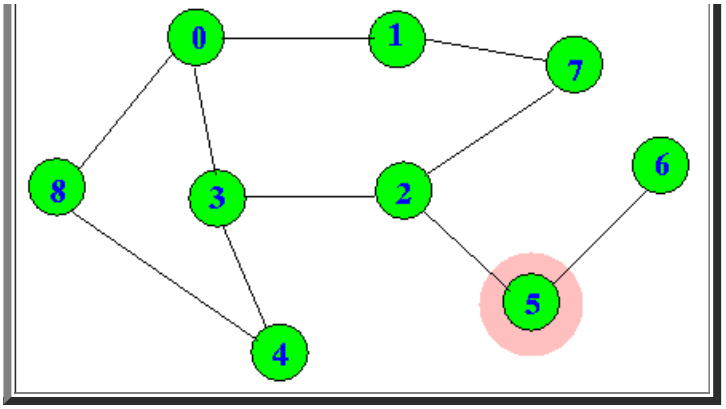
- ~~dfs(2) → dfs(3)~~; dfs(2) → dfs(5)



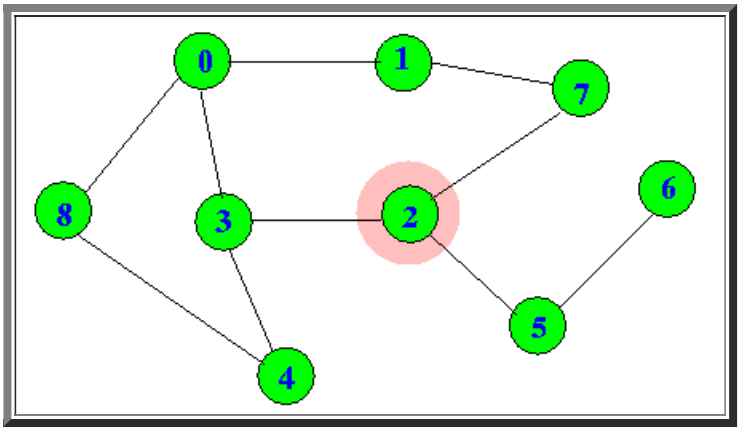
- ~~dfs(5) → dfs(2)~~; dfs(5) → dfs(6)



- ~~dfs(6) → dfs(5)~~; return to dfs(5)



- **return to dfs(2)**



- **return to dfs(7)**

- **return to dfs(1)**

- **return to dfs(0)**

  DONE

- **Example Program:** (Demo above code) *Example*
  - The **DFS** Prog file: click here
  - Test program file: click here

- **Alternative implementation: use a *stack***
  - We can **avoid** using **recursion** by **pushing *active* nodes** onto a **stack**

  - **Active node**

    - **Active node** = a **node** where we ***still*** have to **visit** all its **neighbor nodes**

  - **Psuedo code:**

```
/* ================================================
```

```
   Dept First Traversal of a graph without recursion
   ================================================= */
dfs()
{
   pick a node x....
   push(x);
   visited[x] = true;

   while ( stack != empty )
   {
      n = node at stack top (peek only);

      nextNode = an unvisited node adjacent to n;

      if ( nextNode exists )
      {
         visited[nextNode] = true;

         push(nextNode);              // Process this node first
      }
      else
      {  /* ------------------------------------------------
            Node at top of stack has no unvisited neighbor nodes
            ------------------------------------------------ */
         pop();       // Move on to the next node on the stack
      }
   }
}
```

- **Example Program:** (Demo above code) *Example*
  - The **DFS** ***without*** using recursion Prog file: click here
  - Test program file: click here