Look, a graph!
Look, a graph!
Look, a graph!
Depth-First Search

• Given a graph and one of its nodes \( u \)

(say node 1 below)
Depth-First Search

• Given a graph and one of its nodes $u$ (say node 1 below)
• We want to “visit” each node reachable from $u$ (nodes 1, 0, 2, 3, 5)

There are many paths to some nodes.
How do we visit all nodes efficiently, without doing extra work?
Depth-First Search

boolean[ ] visited;

• Node u is visited means: visited[u] is true
• To visit u means to: set visited[u] to true
• v is explorable from u if there is a path (u, ..., v) in which all nodes of the path are unvisited.

Suppose all nodes are unvisited.
boolean[ ] visited;

- Node u is visited means: visited[u] is true
- To visit u means to: set visited[u] to true
- v is **explorable** from u if there is a path (u, ..., v) in which all nodes of the path are unvisited.

Suppose all nodes are unvisited.

Nodes **explorable** from node 1: {1, 0, 2, 3, 5}
Depth-First Search

boolean[ ] visited;

• Node u is visited means: visited[u] is true
• To visit u means to: set visited[u] to true
• v is explorable from u if there is a path (u, ..., v) in which all nodes of the path are unvisited.

Suppose all nodes are unvisited.

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

Nodes explorable from 4: {4, 5, 6}
boolean[ ] visited;

- Node u is visited means: visited[u] is true
- To visit u means to: set visited[u] to true
- v is explorable from u if there is a path \( (u, ..., v) \) in which all nodes of the path are unvisited.

**Green**: visited  
**Blue**: unvisited
Depth-First Search

boolean[ ] visited;

• Node u is visited means: visited[u] is true
• To visit u means to: set visited[u] to true
• v is explorable from u if there is a path (u, ..., v) in which all nodes of the path are unvisited.

Green: visited
Blue: unvisited

Nodes explorable from node 1: {1, 0, 5}
Depth-First Search

boolean[ ] visited;

- Node u is visited means: visited[u] is true
- To visit u means to: set visited[u] to true
- v is exploriable from u if there is a path (u, ..., v) in which all nodes of the path are unvisited.

Not even 4 itself, because it’s already been visited!
Depth-First Search

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

Let u be 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) {

Let u be 1

The nodes explorable from 1 are 1, 0, 2, 3, 5
/** Visit all nodes that are explorable from u. Precondition: u is unvisited. */

```java
public static void dfs(int u) {
    visited[u] = true;
}
```

Let u be 1

The nodes explorable from 1 are 1, 0, 2, 3, 5
/** Visit all nodes that are explorable from u. Precondition: u is unvisited. */

```java
public static void dfs(int u) {
    visited[u] = true;
}
```

Let u be 1 (visited)

The nodes to be visited are 0, 2, 3, 5
/** Visit all nodes that are explorable from u. Precondition: u is unvisited. */

```java
public static void dfs(int u) {
    visited[u] = true;
    for all edges (u, v) leaving u:
        if v is unvisited then dfs(v);
}
```

Let u be \textbf{1} (visited)

The nodes to be visited are \textbf{0, 2, 3, 5}

Have to do DFS on all unvisited neighbors of u!
/** Visit all nodes that are explorable from u. Precondition: u is unvisited. */

```java
public static void dfs(int u) {
    visited[u] = true;
    for all edges (u, v) leaving u:
        if v is unvisited then dfs(v);
}
```

Suppose the `for` loop visits neighbors in numerical order. Then `dfs(1)` visits the nodes in this order: 1 ...
/** 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 then dfs(v);
}

Suppose the for loop visits neighbors in numerical order. Then \texttt{dfs(1)} visits the nodes in this order: 1, 0 ...
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 then dfs(v);
}

Suppose the for loop visits neighbors in numerical order. Then dfs(1) visits the nodes in this order: 1, 0, 2 ...
/** 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 then dfs(v);
}

Suppose the for loop visits neighbors in numerical order. Then dfs(1) visits the nodes in this order: 1, 0, 2, 3 ...
/** Visit all nodes that are explorable from u. Precondition: u is unvisited. */

```java
public static void dfs(int u) {
    visited[u] = true;
    for all edges (u, v) leaving u:
        if v is unvisited then dfs(v);
}
```

Suppose the for loop visits neighbors in numerical order. Then `dfs(1)` visits the nodes in this order: 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;
    for all edges (u, v) leaving u:
        if v is unvisited then dfs(v);
}

Suppose \( n \) nodes are explorable along \( e \) edges (in total). What is
• Worst-case runtime?
• Worst-case space?
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 then dfs(v);
}

Suppose \( n \) nodes are explorable along \( e \) edges (in total). What is
• Worst-case runtime? \( O(n+e) \)
• Worst-case space? \( O(n) \)
/** Visit all nodes that are explorable from u. Precondition: u is unvisited. */

```java
public static void dfs(int u) {
    visited[u] = true;
    for all edges (u, v) leaving u:
        if v is unvisited then dfs(v);
}
```

Example: Use different way (other than array visited) to know whether a node has been visited

Example: We really haven’t said what data structures are used to implement the graph

That’s all there is to basic DFS. You may have to change it to fit a particular situation.

If you don’t have this spec and you do something different, it’s probably wrong.
Depth-First Search in OO fashion

```java
public class Node {
    boolean visited;
    List<Node> neighbors;

    /** Visit all nodes that are explorable from u. Precondition: u is unvisited */
    public void dfs() {
        visited = true;
        for (Node n: neighbors) {
            if (!n.visited) n.dfs();
        }
    }
}
```

Each node of the graph is an object of type Node. No need for a parameter. The object is the node.
Depth-First Search written iteratively

/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void dfs(int nodeID) {
    Stack s = (nodeID); // Not Java!
    // inv: all nodes that have to be visited are
    //      explorable from some node in s
    while (s is not empty) {
        u = s.pop(); // Remove top stack node, put in u
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

```
Call dfs(1)
```

Stack s
Depth-First Search written iteratively

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s = (u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void dfs(int u) {
    Stack s = new Stack();
    s.push(u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}

Call dfs(1)  Iteration 0

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s= new Stack();
    while (s is not empty) {
        u= s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}

Call dfs(1)  
Iteration 0

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s = (u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s = (u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void dfs(int u) {
    Stack s = new Stack();
    s.push(u);
    while (!s.isEmpty()) {
        u = s.pop();
        if (!isVisited(u)) {
            visit(u);
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}

Call dfs(1)  Iteration 1

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = (u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call `dfs(1)`

Iteration 1

Stack s

1 0 2 5 3 4 6

2
5
/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void dfs(int u) {
    Stack s = (u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
**Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    while (!s.isEmpty()) {
        int u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call `dfs(1)`

Iteration 2

Stack `s`
Depth-First Search written iteratively

/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    while (!s.isEmpty()) {
        int u = s.pop();
        if (!u.isVisited()) {
            u.visit();
            for (int v : u.getNeighbors()) {
                s.push(v);
            }
        }
    }
}
```

Call dfs(1)  
Iteration 2

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack(u);
    while (!s.isEmpty()) {
        int u = s.pop();
        if (!u.isVisited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call `dfs(1)`

Iteration 2

Yes, 5 is put on the stack twice, once for each edge to it. It will be visited only once.
Depth-First Search written iteratively

/\** Visit all nodes explorable from u. Pre: u is unvisited. */\n\public static void dfs(int u) \{
  Stack s= \( (u) \);
  while (s is not empty) \{
    u= s.pop();
    if (u has not been visited) \{
      visit u;
      for each edge \((u, v)\) leaving u:
        s.push(v);
    }
  }
\}"
**Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    s.push(u);
    while (!s.isEmpty()) {
        int u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call dfs(1)  

Iteration 3

Stack s

```
1 2 3

1 2 3
```
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    s.push(u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call `dfs(1)`

Iteration 3

Stack `s`

1 2 3

4 5 6 0
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack(u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call `dfs(1)`  
Iteration 3  

Stack `s`
/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s = (u);
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}

Call dfs(1)  Iteration 4

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    while (!s.isEmpty()) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call dfs(1)  

Iteration 4

Stack s
** Depth-First Search written iteratively **

/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void dfs(int u) {
    Stack s = new Stack();
    s.push(u);
    while (!s.isEmpty()) {
        u = s.pop();
        if (!u.hasVisited()) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```

Call `dfs(1)`  
Iteration 4

G: 1 - 2 - 3 - 5 - 0 - 5 - 6 - 4

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s= (u);
    while (s is not empty) {
        u= s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
Depth-First Search written iteratively

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s= (u);
    while (s is not empty) {
        u= s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}

Call dfs(1)  
Iteration 6

Stack s
/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void dfs(int u) {
    Stack s= (u);
    while (s is not empty) {
        u= s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
That’s DFS!

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void dfs(int u) {
    Stack s= (u);  // Not Java!
    // inv: all nodes that have to be visited are
    //     explorable from some node in s
    while (s is not empty) {
        u= s.pop();  // Remove top stack node, put in u
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}

Want to see a magic trick?
/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void dfs(int u) {
    Stack s = new Stack(); // Not Java!
    // inv: all nodes that have to be visited are
    //      explorable from some node in s
    while (s is not empty) {
        u = s.pop(); // Remove top stack node, put in u
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q = (u); // Not Java!
    // inv: all nodes that have to be visited are
    //     explorable from some node in s
    while (q is not empty) {
        u = q.popFirst(); // Remove first node in queue, put in u
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v); // Add to end of queue
        }
    }
}
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void bfs(int u) {
    Queue q = (u); // Not Java!
    // inv: all nodes that have to be visited are
    //      explorable from some node in s
    while (q is not empty) {
        u = q.popFirst(); // Remove first node in queue, put in u
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v); // Add to end of queue
        }
    }
}

Breadth-First Search
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void bfs(int u) {
    Queue q= (u);
    while q is not empty) {
        u= q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

Call bfs(1)
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void bfs(int u) {
    Queue q = (u);
    while (q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

Call bfs(1)  Iteration 0

1 0 2 5
3 6 4

Queue q
Breadth-First Search

/** Visit all nodes explor able from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q = new Queue();
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}

Call bfs(1)  Iteration 0

Queue q

Call bfs(1)  Iteration 0

Queue q
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q = (u);
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
Breadth-First Search

/** Visit all nodes exploriable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while q is not empty) {
        u= q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}

Call bfs(1)  Iteration 0

0 2
Queue q
/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q = new Queue(u);
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q = (u);
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}

Call bfs(1)  
Iteration 1

1 2 3 4 5 6 7

Queue q

2
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while q is not empty) {
        u= q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}

Call bfs(1)  Iteration 1

Queue q
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */

public static void bfs(int u) {
    Queue q = (u);
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}

Call bfs(1)  Iteration 1

1 0 2 3 4 5 6 7

2 7
Queue q
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void bfs(int u) {
    Queue q = (u);
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

Call bfs(1)  

Iteration 2

Queue q
/** Visit all nodes explorable from u. Pre: u is unvisited. */

```java
public static void bfs(int u) {
    Queue q = new Queue(u);
    while (!q.isEmpty()) {
        int u = q.poll();
        if (!visited(u)) {
            visit(u);
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

Call bfs(1)  
Iteration 2  

---

Breadth-First Search

Queue q

1 2 3 4 5 6 7

Iteration 2

Node 1 visited
Node 2 visited
Node 3 visited
Node 4 visited
Node 5 visited
Node 6 visited
Node 7 visited
Breadth-First Search

/** Visit all nodes explorible from u. Pre: u is unvisited. */

```java
public static void bfs(int u) {
  Queue q = new Queue(u);
  while q is not empty) {
    u = q.popFirst();
    if (u has not been visited) {
      visit u;
      for each edge (u, v) leaving u:
        q.append(v);
    }
  }
}
```

Call bfs(1)

Iteration 2

Queue q
Breadth-First Search

/** Visit all nodes explorable from u. Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q = (u);
    while q is not empty) {
        u = q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}

Call bfs(1)  

Iteration 2

Breadth first:
(1) Node u
(2) All nodes 1 edge from u
(3) All nodes 2 edges from u
(4) All nodes 3 edges from u
...

7 3 5
Queue q
Some working code for DFS

- [https://codeboard.io/projects/97448](https://codeboard.io/projects/97448)
- Sample graph constructed by the code:

- Suggested exercises:
  - Run DFS by hand
  - Run BFS by hand
  - Code BFS
Questions to Ponder

• BFS(root) on a tree corresponds to which tree traversal?

• Write out the order nodes are visited in this undirected graph, when calling:
  – BFS(5)
  – DFS(5)
  – DFS(0)

(if there are ties, visit the lower # first)