CSCI 241

Scott Wehrwein

Graph Traversals:
Depth-First Search (Iteratively)
Goals

Be able to implement DFS iteratively using a stack.
/** Visit all nodes explorable from u.
 * Pre: u is unvisited. */

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

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

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

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

dfs(1)
Depth-first Search: 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 to be visited are
  //      explorable from some node in s
  while (s is not empty) {
    u = s.pop();
    if (u has not been visited) {
      visit u;
      for each edge (u, v) from u:
        s.push(v);
    }
  }
}

dfs(1)
Depth-first Search: 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 to be visited are
    //      explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}
/** Visit all nodes explorable from u.  
* Pre: u is unvisited. */

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

dfs(1)
Depth-first Search: Iteratively

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

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

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

dfs(1)
Depth-first Search: 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 to be visited are
  //      explorable from some node in s
  while (s is not empty) {
    u = s.pop();
    if (u has not been visited) {
      visit u;
      for each edge (u, v) from u:
        s.push(v);
    }
  }
}

dfs(1)
Depth-first Search: 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 to be visited are explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}

dfs(1)
**Depth-first Search: Iteratively**

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

Stack s:
```
dfs(1)
```

```
5 5
```
Depth-first Search: Iteratively

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

```java
public static void dfs(int nodeID) {
    Stack s = (nodeID); // Not Java!
    // inv: all nodes to be visited are
    //     explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}
```

Stack s:

dfs(1)
Depth-first Search: 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 to be visited are 
    //      explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}

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

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

dfs(1)
Depth-first Search: 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 to be visited are
    //      explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}

dfs(1)
Depth-first Search: 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 to be visited are explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}

Stack s:

dfs(1)
Depth-first Search: 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 to be visited are 
    //      explorable from some node in s
    while (s is not empty) {
        u = s.pop();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) from u:
                s.push(v);
        }
    }
}

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

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

dfs(1)