Graph Traversals

• Depth-First Traversals.
  – Algorithms.
  – Example.
  – Implementation.

• Breadth-First Traversal.
  – The Algorithm.
  – Example.
  – Implementation.

• Review Questions.
Depth-First Traversal Algorithm

- In this method, after visiting a vertex v, which is adjacent to w1, w2, w3, ..., next we visit one of v's adjacent vertices, w1 say. Next, we visit all vertices adjacent to w1 before coming back to w2, etc.

- Must keep track of vertices already visited to avoid cycles.

- The method can be implemented using recursion or iteration.

- The iterative preorder depth-first algorithm is:

1. push the starting vertex onto the stack
2. while(stack is not empty){
3.   pop a vertex off the stack, call it v
4.   if v is not already visited, visit it
5.   push vertices adjacent to v, not visited, onto the stack
6. }

- Note: Adjacent vertices can be pushed in any order; but to obtain a unique traversal, we will push them in reverse alphabetical order.
Example

- Demonstrates depth-first traversal using an explicit stack.

<table>
<thead>
<tr>
<th>Order of Traversal</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>D</td>
<td>H</td>
<td>I</td>
</tr>
</tbody>
</table>
Recursive preorder Depth-First Traversal Implementation

defPreorder(v){
    visit v;
    for(each neighbour w of v)
        if(w has not been visited)
            dfsPreorder(w);
}

- The following is the code for the recursive preorderDepthFirstTraversal method of the AbstractGraph class:

```java
public void preorderDepthFirstTraversal(Visitor visitor, Vertex start)
{
    boolean visited[] = new boolean[numberOfVertices];
    for(int v = 0; v < numberOfVertices; v++)
        visited[v] = false;
    preorderDepthFirstTraversal(visitor, start, visited);
}
```
private void preorderDepthFirstTraversal(Visitor visitor, Vertex v, boolean[] visited)
{
    if(visitor.isDone())
        return;
    visitor.visit(v);
    visited[getIndex(v)] = true;

    Iterator p = v.getSuccessors();
    while(p.hasNext())
    {
        Vertex to = (Vertex) p.next();
        if(! visited[getIndex(to)])
            preorderDepthFirstTraversal(visitor, to, visited);
    }
}
At each stage, a set of unvisited adjacent vertices of the current vertex is generated.
Recursive postorder Depth-First Traversal Implementation

dfsPostorder(v){
    mark v;
    for(each neighbour w of v)
        if(w is not marked)
            dfsPostorder(w);
    visit v;
}

•The following is the code for the recursive postorderDepthFirstTraversal method of the AbstractGraph class:

```java
public void postorderDepthFirstTraversal(Visitor visitor, Vertex start)
{
    boolean visited[] = new boolean[numberOfVertices];
    for(int v = 0; v < numberOfVertices; v++)
        visited[v] = false;
    postorderDepthFirstTraversal(visitor, start, visited);
}
```
private void postorderDepthFirstTraversal(
    Visitor visitor, Vertex v, boolean[] visited)
{
    if(visitor.isDone())
        return;

    // mark v
    visited[getIndex(v)] = true;

    Iterator p = v.getSuccessors();
    while(p.hasNext()){
        Vertex to = (Vertex) p.next();
        if(! visited[getIndex(to)])
            postorderDepthFirstTraversal(visitor, to, visited);
    }

    // visit v
    visitor.visit(v);
}
Recursive postorder Depth-First Traversal Implementation (cont’d)

At each stage, a set of unmarked adjacent vertices of the current vertex is generated.
Breadth-First Traversal Algorithm

• In this method, After visiting a vertex v, we must visit all its adjacent vertices w1, w2, w3, ..., before going down next level to visit vertices adjacent to w1 etc.

• The method can be implemented using a queue.

• A boolean array is used to ensure that a vertex is enqueued only once.

1 enqueue the starting vertex
2 while(queue is not empty){
3 dequeue a vertex v from the queue;
4 visit v.
5 enqueue vertices adjacent to v that were never enqueued;
6 }

• Note: Adjacent vertices can be enqueued in any order; but to obtain a unique traversal, we will enqueue them in alphabetical order.
Example

- Demonstrating breadth-first traversal using a queue.

<table>
<thead>
<tr>
<th>Order of Traversal</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>C</td>
<td>G</td>
<td>F</td>
<td>H</td>
<td>I</td>
</tr>
</tbody>
</table>
public void breadthFirstTraversal(Visitor visitor, Vertex start) {
    boolean enqueued[] = new boolean[numberOfVertices];
    for (int i = 0; i < numberOfVertices; i++) enqueued[i] = false;

    Queue queue = new QueueAsLinkedList();
    enqueued[getIndex(start)] = true;
    queue.enqueue(start);

    while (!queue.isEmpty() && !visitor.isDone()) {
        Vertex v = (Vertex) queue.dequeue();
        visitor.visit(v);
        Iterator it = v.getSuccessors();
        while (it.hasNext()) {
            Vertex to = (Vertex) it.next();
            int index = getIndex(to);
            if (!enqueued[index]) {
                enqueued[index] = true;
                queue.enqueue(to);
            }
        }
    }
}
Review Questions

1. Consider a depth-first traversal of the undirected graph GA shown above, starting from vertex a.
   - List the order in which the nodes are visited in a preorder traversal.
   - List the order in which the nodes are visited in a postorder traversal

2. Repeat exercise 1 above for a depth-first traversal starting from vertex d.

3. List the order in which the nodes of the undirected graph GA shown above are visited by a breadth-first traversal that starts from vertex a. Repeat this exercise for a breadth-first traversal starting from vertex d.

4. Repeat Exercises 1 and 3 for the directed graph GB.