Graph Traversals

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

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

- Review Questions.

Depth-First Traversal Algorithm

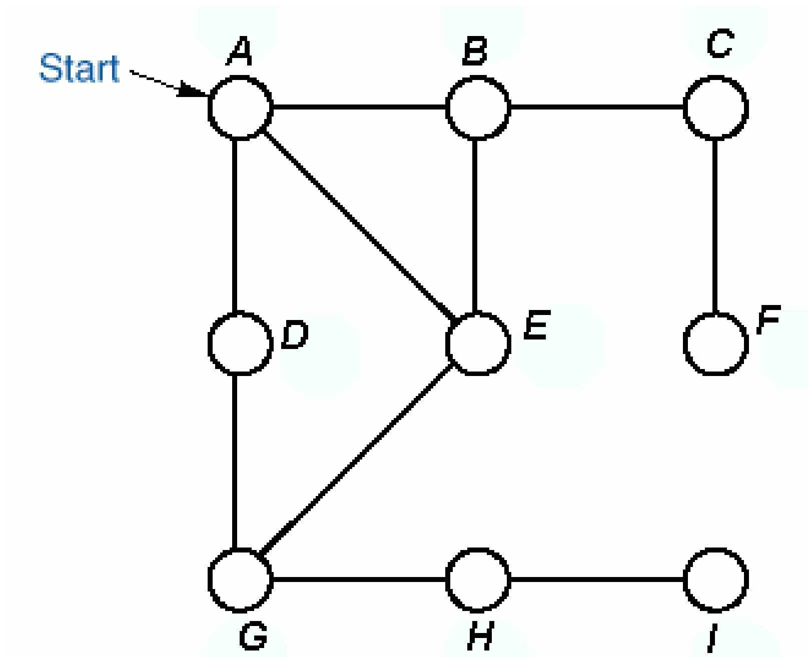
- In this method, After visiting a vertex v, which is adjacent to w1, w2, w3, ...; Next we visit one of v's adjacent vertices, w1 say. Next, we visit all vertices adjacent to w1 before coming back to w2, etc.
- Must keep track of vertices already visited to avoid cycles.
- The method can be implemented using recursion or iteration.
- The iterative preorder depth-first algorithm is:

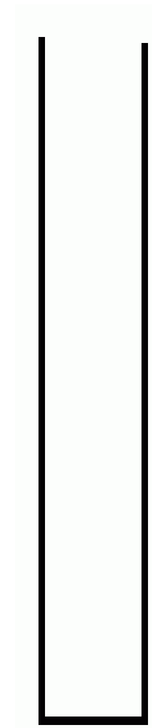
```
push the starting vertex onto the stack
while(stack is not empty){
    pop a vertex off the stack, call it v
    if v is not already visited, visit it
    push vertices adjacent to v, not visited, onto the stack
 }
```

- 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.





Stack

Order of Traversal

1	2	3	4	5	6	7	8	9
A	B	C	F	E	G	D	H	I

Recursive preorder Depth-First Traversal Implementation

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

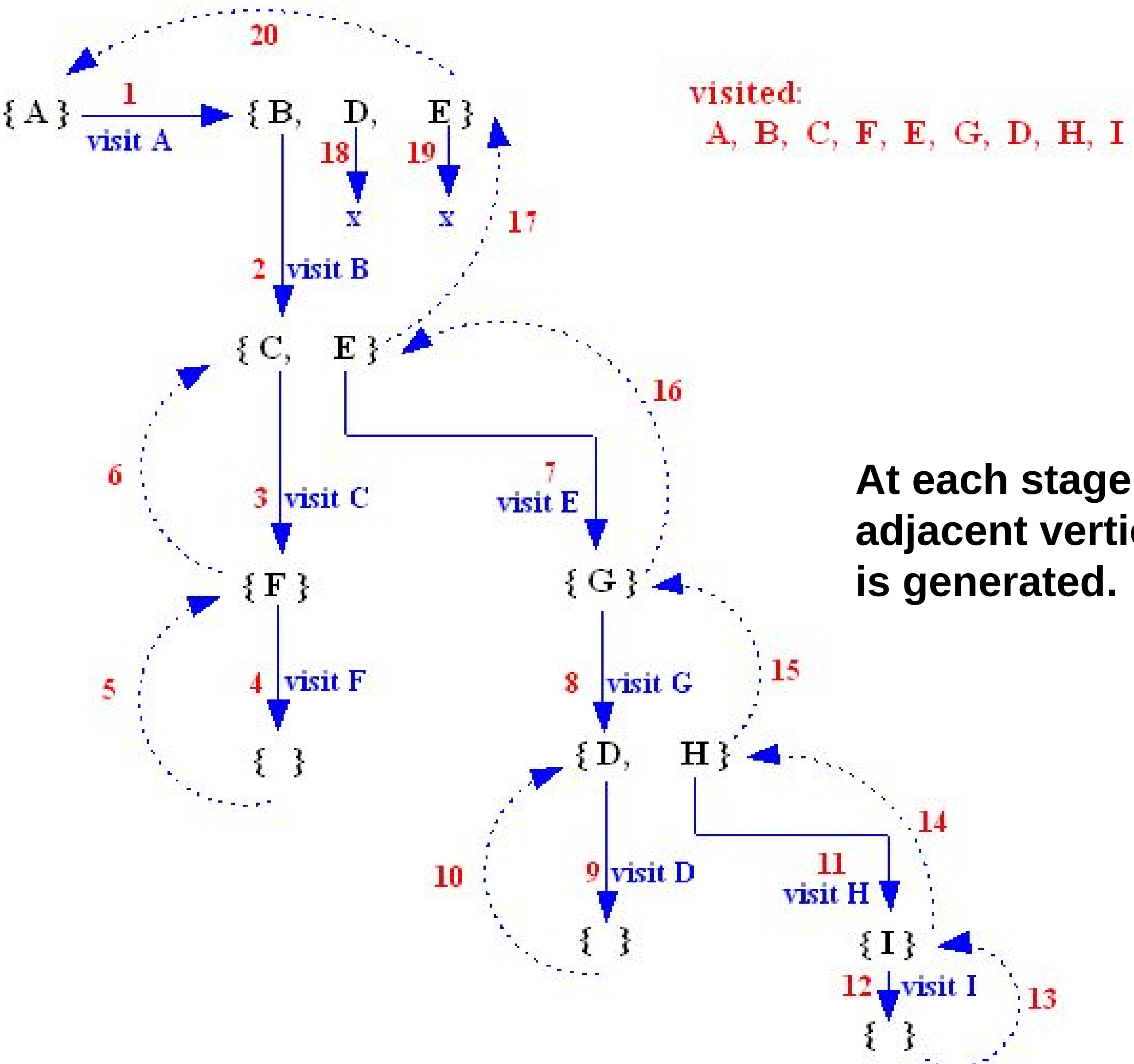
- The following is the code for the recursive preorderDepthFirstTraversal method of the AbstractGraph class:

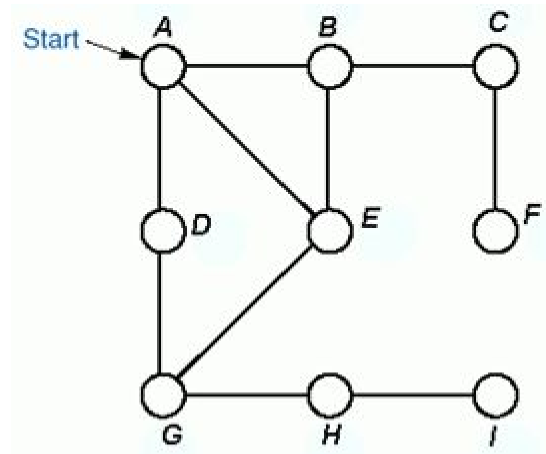
```java
public void preorderDepthFirstTraversal(Visitor visitor, Vertex start)
{
   boolean visited[] = new boolean[numberOfVertices];
   for(int v = 0; v < numberOfVertices; v++)
      visited[v] = false;
      preorderDepthFirstTraversal(visitor, start, visited);
}
```

Recursive preorder Depth-First Traversal Implementation (cont'd)

```
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);
   }
}
```

Recursive preorder Depth-First Traversal Implementation (cont’d)





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;
}
```

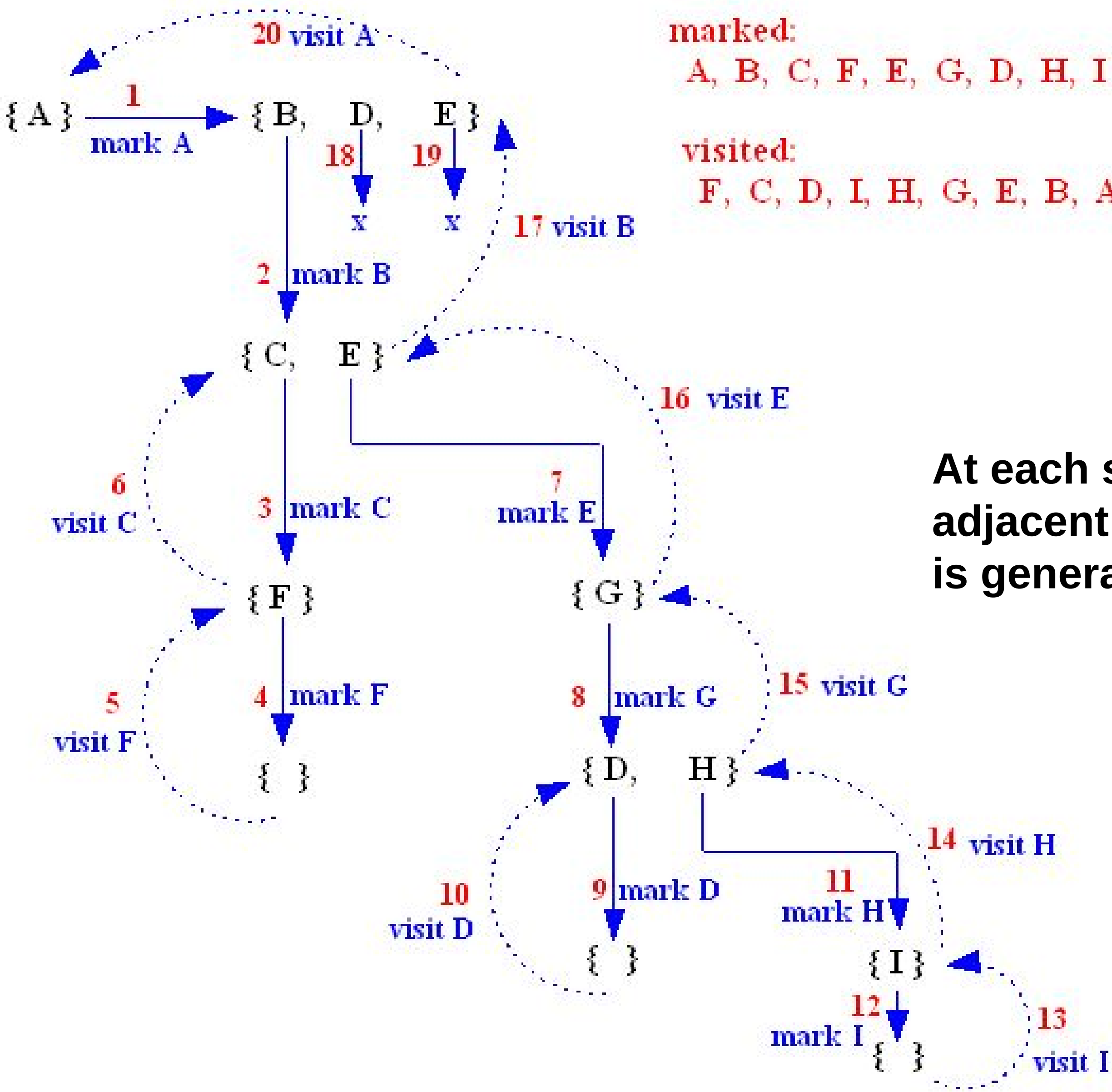
- The following is the code for the recursive postorderDepthFirstTraversal method of the AbstractGraph class:

```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);
}
```

Recursive postorder Depth-First Traversal Implementation (cont'd)

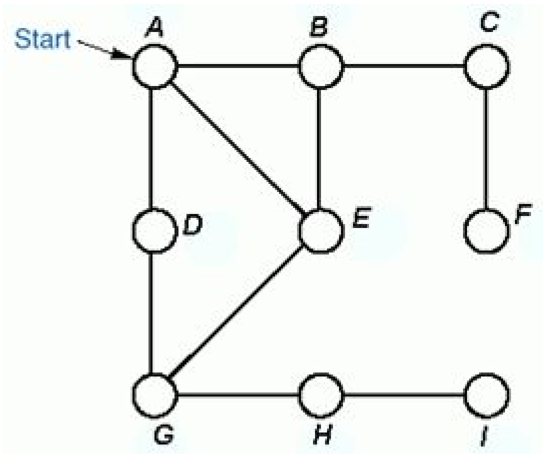
```
private void postorderDepthFirstTraversal(
           Visitor visitor, Vertex v, boolean[] visited)
{
   if(visitor.isDone())
      return;

   // 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)



marked:
A, B, C, F, E, G, D, H, I

visited:
F, C, D, I, H, G, E, B, A



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.

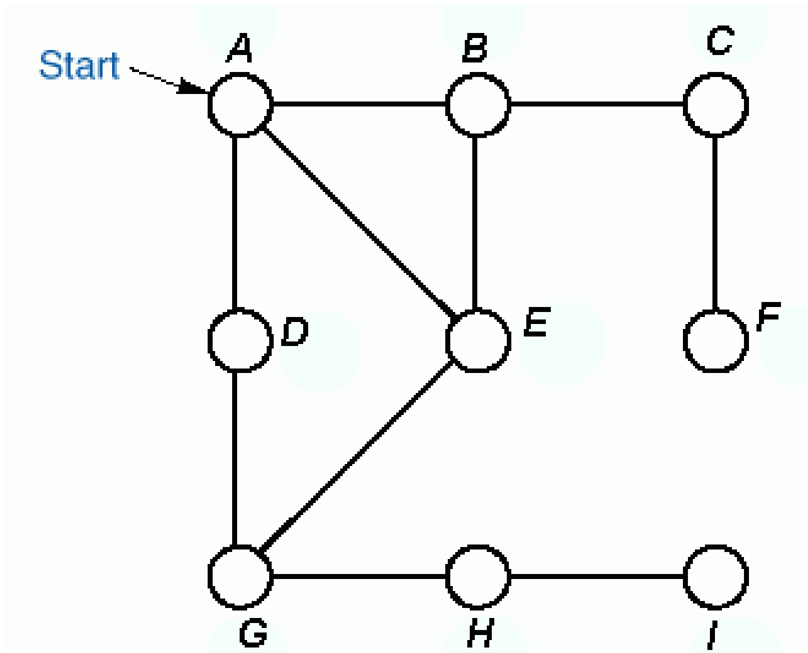
- A boolean array is used to ensure that a vertex is enqueued only once.

```
enqueue the starting vertex
while(queue is not empty){
     dequeue a vertex v from the queue;
      visit v.
     enqueue vertices adjacent to v that were never enqueued;
 }
```

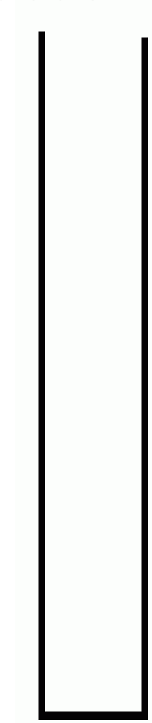
- Note: Adjacent vertices can be enqueued in any order; but to obtain a unique traversal, we will enqueue them in alphabetical order.

Example

- Demonstrating breadth-first traversal using a queue.



Queue front



Queue rear

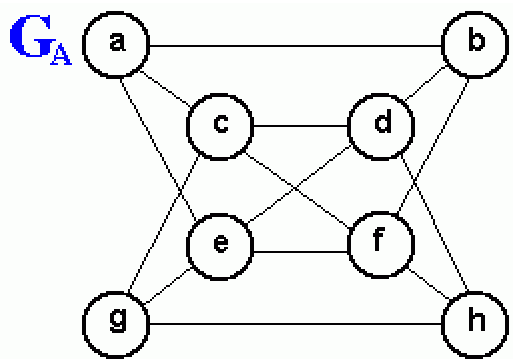
Order of Traversal

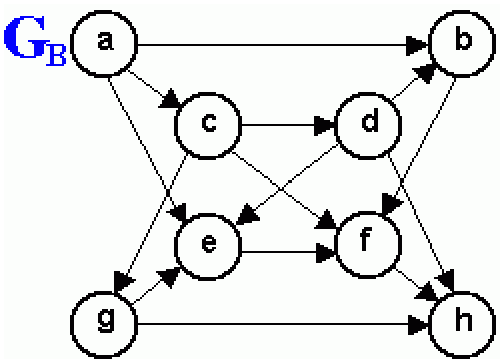
1	2	3	4	5	6	7	8	9
A	**B**	**D**	**E**	**C**	**G**	**F**	**H**	**I**

Breadth-First Traversal Implementation

```
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. Considera 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.