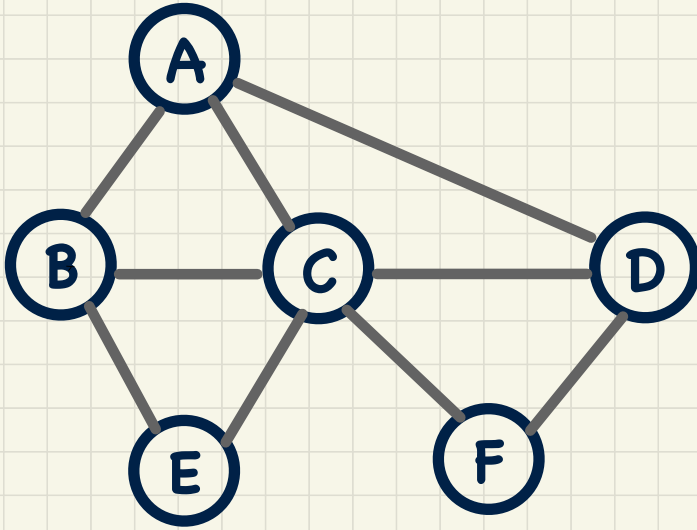


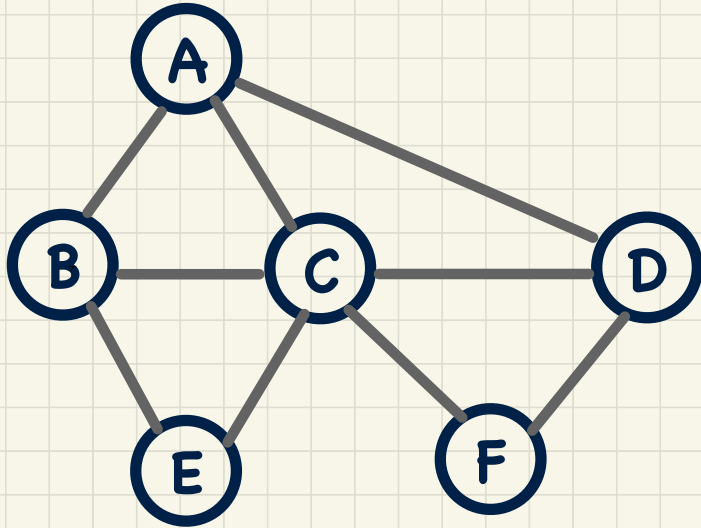
Breadth-First Search (BFS): Example 1 (a)



Assumptions:

- Adjacent vertices visited in alphabetic order

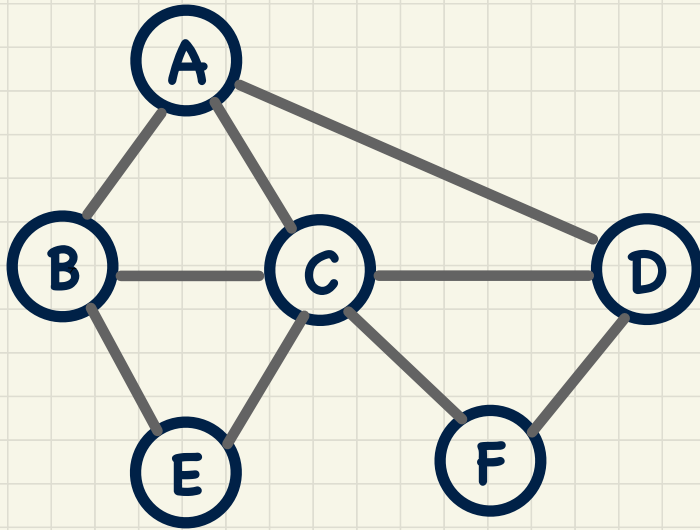
Breadth-First Search (BFS): Example 1 (b)



Assumptions:

- Adjacent vertices visited in alphabetic order
- Exception: Edge AC visited first

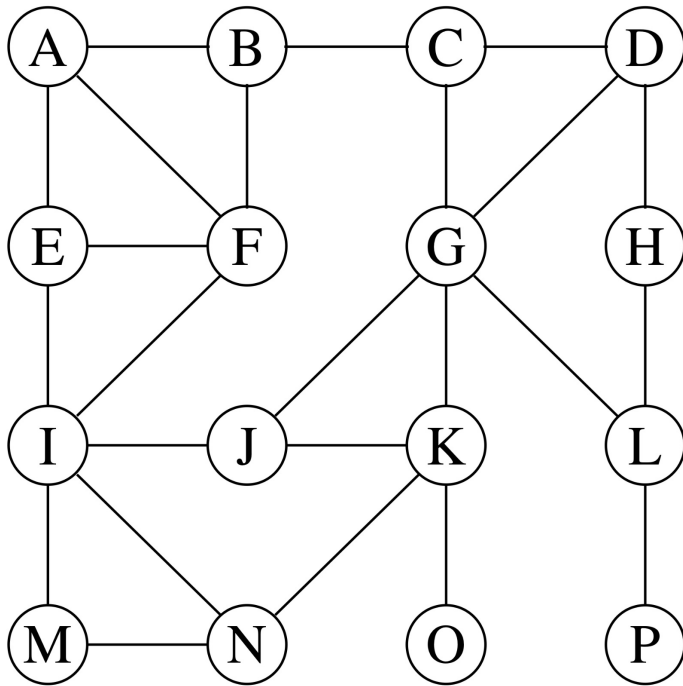
Breadth-First Search (BFS): Example 1 (c)



Assumptions:

- Adjacent vertices visited in alphabetic order
- Exception: Edge AD visited first

Breadth-First Search (BFS): Example 2

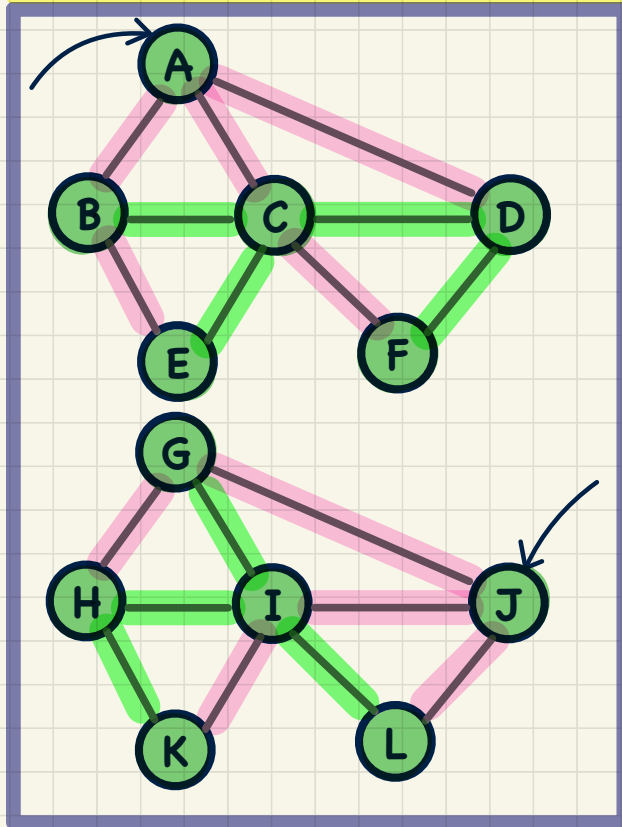


Assumptions:

- Adjacent vertices visited in alphabetic order

Graph Traversals: Adapting BFS

Efficient Traversal of Graph G:



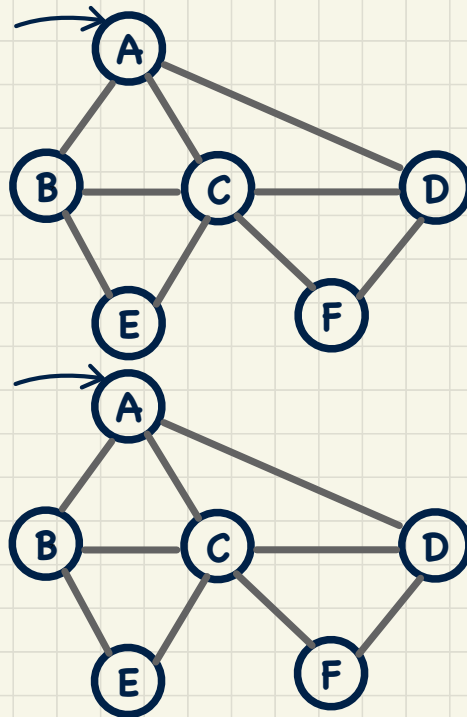
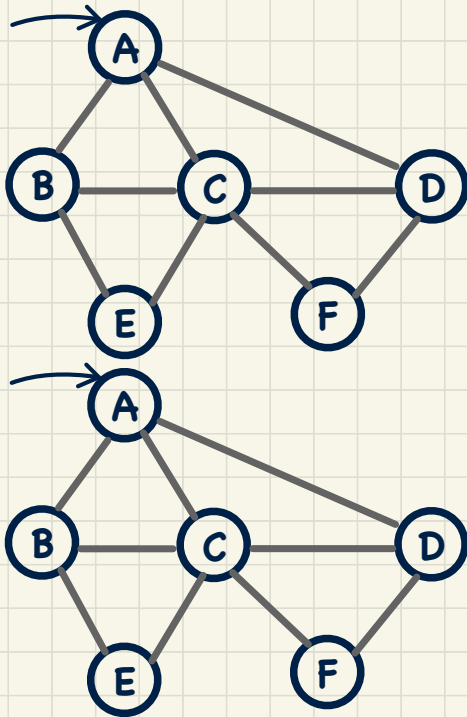
Graph Questions:

- Find a **path** between $\{u, v\} \subseteq V$
- Is v **reachable** from u
- Find all **connected components** of G .
- Compute a **spanning tree** of a connected G .
- Is G **connected**?
- If G is cyclic, return a **cycle**.

Back Edge (DFS) vs. Cross Edge (BFS): Cyclic?

Does a **back edge** always imply the existence of a **cycle**?

Does a **cross edge** always imply the existence of a **cycle**?



Graphs in Java: Adjacency List Strategy (1)

```
class AdjacencyListGraph<V, E> implements Graph<V, E> {  
    private DoublyLinkedList<AdjacencyListVertex<V>> vertices;  
    private DoublyLinkedList<AdjacencyListEdge<E, V>> edges;  
    private boolean isDirected;  
  
    /* initialize an empty graph */  
    AdjacencyListGraph(boolean isDirected) {  
        vertices = new DoublyLinkedList<>();  
        edges = new DoublyLinkedList<>();  
        this.isDirected = isDirected;  
    }  
}
```

```
public class Vertex<V> {  
    private V element;  
    public Vertex(V element) { this.element = element; }  
    /* setter and getter for element */  
}
```

```
public class Edge<E, V> {  
    private E element;  
    private Vertex<V> origin;  
    private Vertex<V> dest;  
    public Edge(E element) { this.element = element; }  
    /* setters and getters for element, origin, and destination */  
}
```

```
public class EdgeListVertex<V> extends Vertex<V> {  
    public DLNode<Vertex<V>> vertexListPosition;  
    /* setter and getter for vertexListPosition */  
}
```

```
public class EdgeListEdge<E, V> extends Edge<E, V> {  
    public DLNode<Edge<E, V>> edgeListPosition;  
    /* setter and getter for edgeListPosition */  
}
```

```
class AdjacencyListVertex<V> extends EdgeListVertex<V> {  
    private DoublyLinkedList<AdjacencyListEdge<E, V>> incidentEdges;  
    /* getter for incidentEdges */  
}
```

```
class AdjacencyListEdge<V> extends EdgeListEdge<V> {  
    DLNode<Edge<E, V>> originIncidentListPos;  
    DLNode<Edge<E, V>> destIncidentListPos;  
}
```