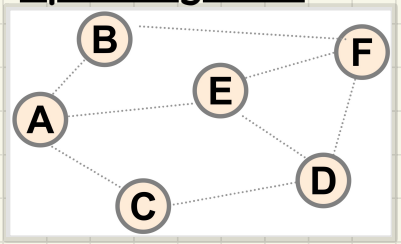
Properties: Structure vs. |V| and |E|

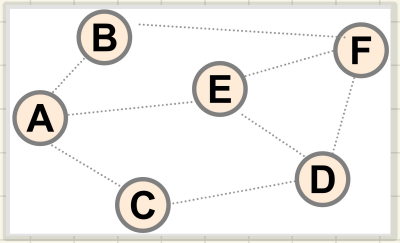
Given $G = (V, G)$ an **undirected** graph with $|V| = n$, $|E| = m$:

$$\begin{cases} m = n - 1 & \textbf{if G is a } \textit{spanning tree} \\ m \leq n - 1 & \textbf{if G is a } \textit{forest} \\ m \geq n - 1 & \textbf{if G is } \textit{connected} \\ m \geq n & \textbf{if G contains a } \textit{cycle} \end{cases}$$

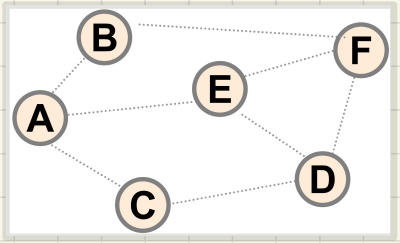
Spanning Tree



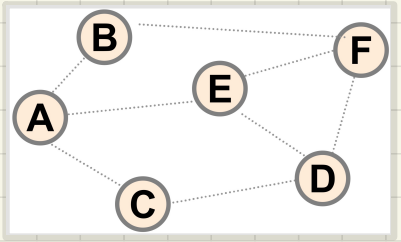
Forest



Connected



Cyclic

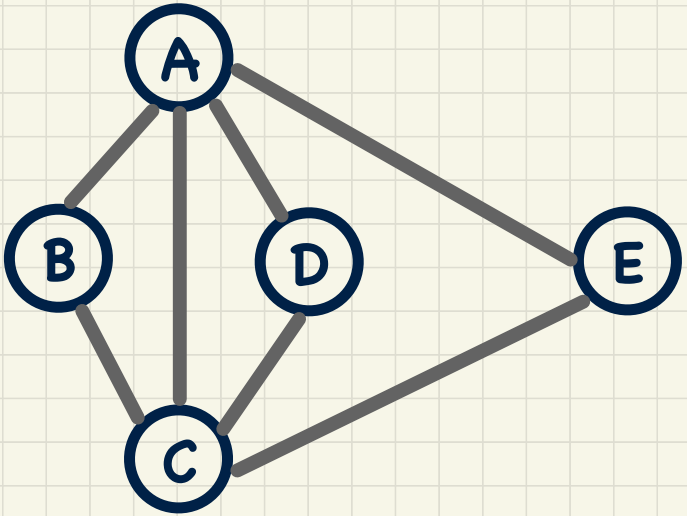


Properties: Structure vs. |V| and |E|

Given $G = (V, G)$ an **undirected** graph with $|V| = n$, $|E| = m$:

$$\begin{cases} m = n - 1 & \textbf{if G is a } \textit{spanning tree} \\ m \leq n - 1 & \textbf{if G is a } \textit{forest} \\ m \geq n - 1 & \textbf{if G is } \textit{connected} \\ m \geq n & \textbf{if G contains a } \textit{cycle} \end{cases}$$

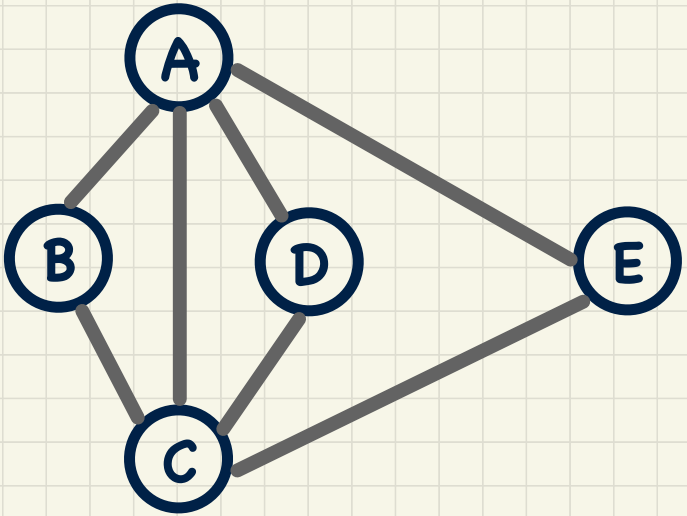
Depth-First Search (DFS): Example 1 (a)



Assumptions:

- Adjacent vertices visited in alphabetic order

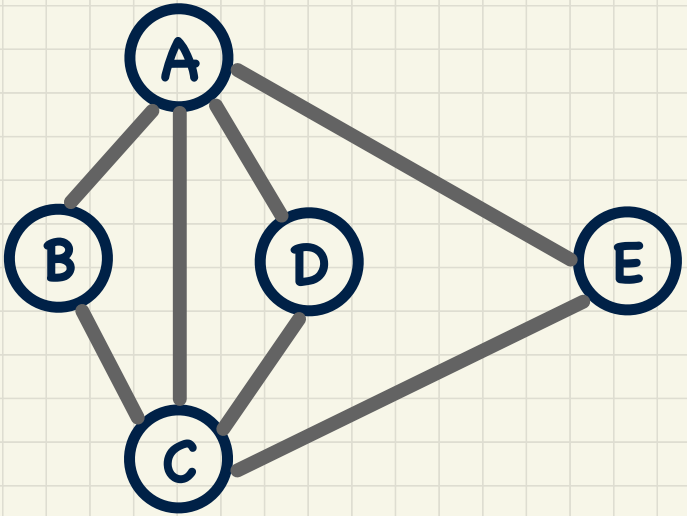
Depth-First Search (DFS): Example 1 (b)



Assumptions:

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

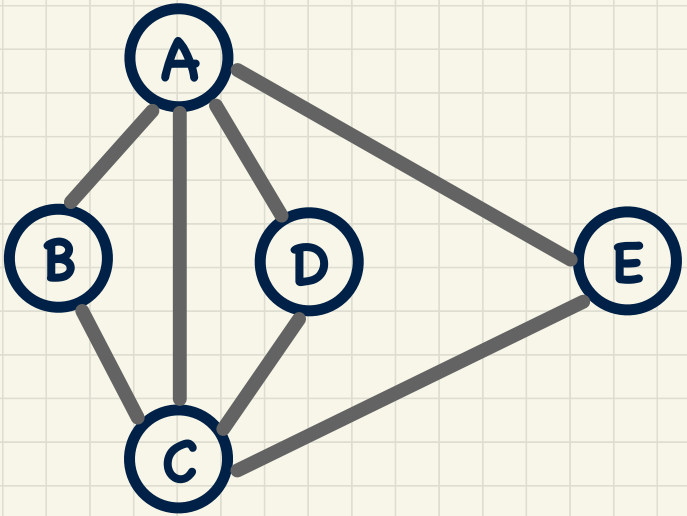
Depth-First Search (DFS): Example 1 (c)



Assumptions:

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

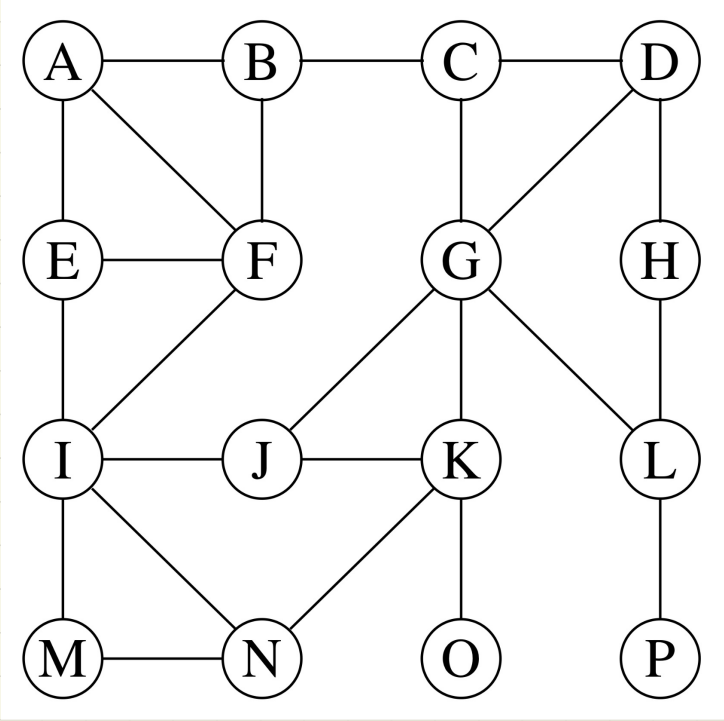
Depth-First Search (DFS): Example 1 (d)



Assumptions:

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

Depth-First Search (DFS): Example 2



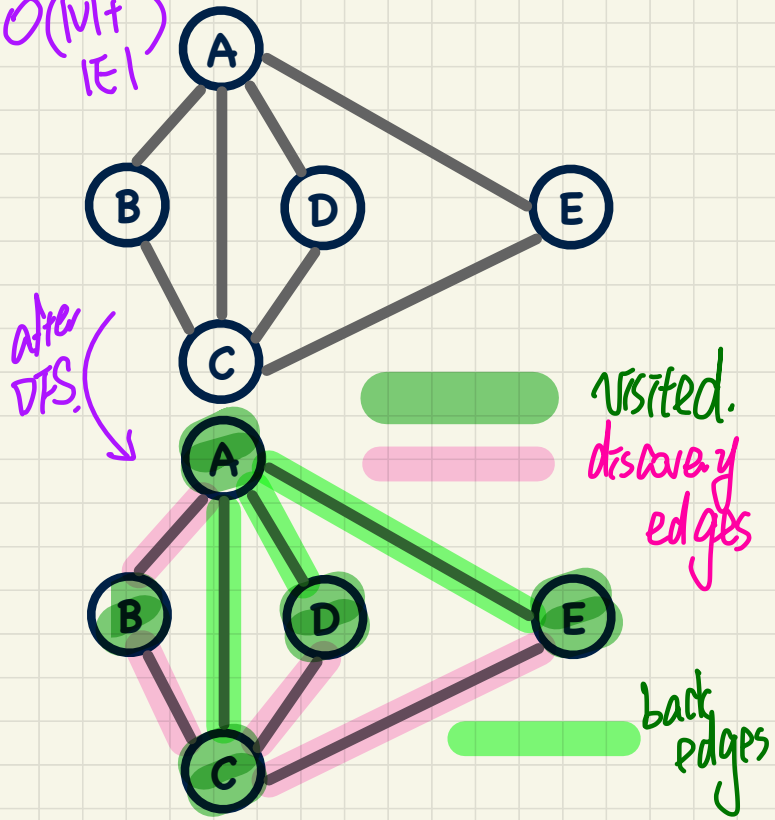
Assumptions:

- Adjacent vertices visited in alphabetic order

Graph Traversals: Definition & Applications

Efficient Traversal of Graph G:

$O(|V| + |E|)$

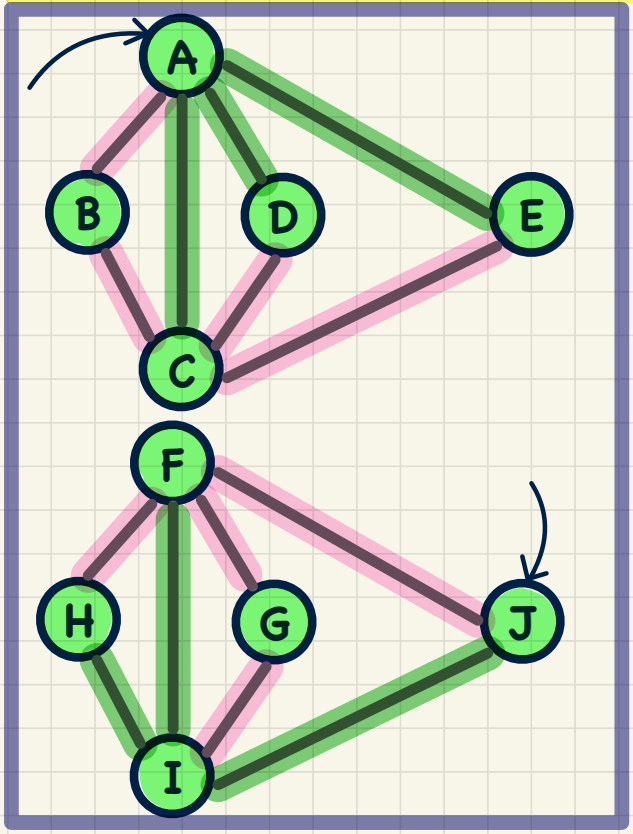


Applications: (visit later).

- **Reachable** Vertices from $v \in V$
- A **path** between $\{u, v\} \subseteq V$
- The **minimum path** between $\{u, v\} \subseteq V$
- Is G **connected**?
- Compute a **spanning tree** of a connected G.
- Compute the **connected components** of G.
- If G is cyclic, return a **cycle**.

Graph Traversals: Adapting DFS

Efficient Traversal of Graph G:



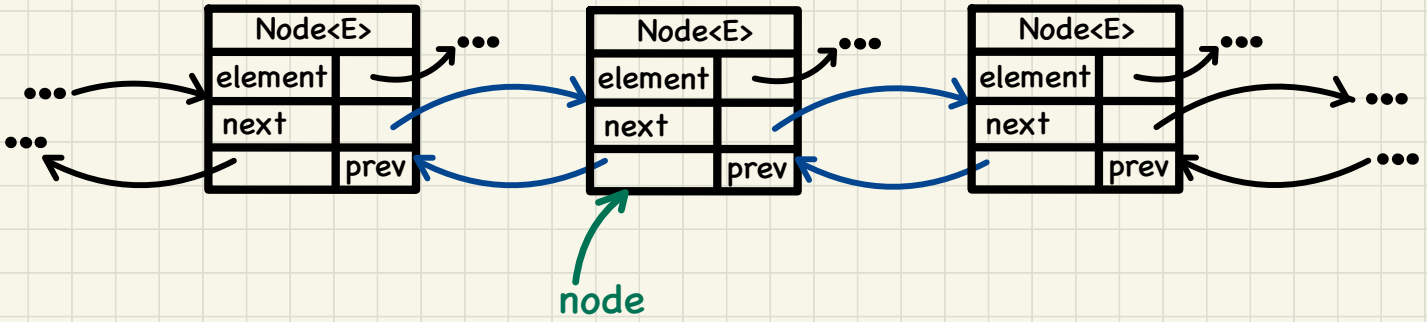
Graph Questions:

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

Graphs in Java: Doubly-Linked Nodes and Lists

```
public class DLNode<E> { /* Doubly-Linked Node */
  private E element;
  private DLNode<E> prev; private DLNode<E> next;
  public DLNode(E e, DLNode<E> p, DLNode<E> n) { ... }
  /* setters and getters for prev and next */
}
```

```
public class DoublyLinkedList<E> {
  private int size;
  private DLNode<E> header; private DLNode<E> trailer;
  public void remove (DLNode<E> node) {
    DLNode<E> pred = node.getPrev();
    DLNode<E> succ = node.getSucc();
    pred.setNext(succ); succ.setPrev(pred);
    node.setNext(null); node.setPrev(null);
    size --;
  }
}
```



Graphs in Java: Edge List Strategy (1)

```
public class EdgeListGraph<V, E> implements Graph<V, E> {
  private DoublyLinkedList<EdgeListVertex<V>> vertices;
  private DoublyLinkedList<EdgeListEdge<E, V>> edges;
  private boolean isDirected;

  /* initialize an empty graph */
  public EdgeListGraph(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>> vertextListPosition;
  /* 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 */
}
```

Graphs in Java: Edge List Strategy (2)