Review of Graphs:

- A **directed graph** \( G = (V, E) \) (aka. **digraph**) is an ordered pair consisting of:
  - A set \( V \) of nodes
  - A set \( E \subseteq V \times V \) of edges

- In an **undirected graph** \( G = (V, E) \), the edge set \( E \) consists of unordered pairs of nodes:
  i.e. \((u, v) \in E \iff (v, u) \in E\).

Observation:

- (i): \( |E| = \Theta(V^2) \), often written as \( E = O(V^2) \).
- (ii): \( \text{If } G \text{ is connected } \Rightarrow |E| \geq |V| - 1 \Rightarrow \text{girth}(G) \geq 2 \)

Representations of Graphs: Let \( G = (V, E) \) be a graph with \( V = \{v_1, \ldots, v_n\} \).

- **Adjacency Matrix Representation**:
  - An \( n \times n \) matrix \( A = (a_{ij}) \).
  - \( a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases} \)
  - \( O(V^2) \) i.e. \( O(n^2) \) space.
  - Good for dense graphs: \( |E| \approx \Omega(V^2) \)

- **Adjacency List Representation**:
  - An adjacency list, one for each vertex
  - For \( v \in V \), \( \text{adj}(v) \) contains all \( w \) s.t. \( (v, w) \in E \).

- Directed Graphs:
  - In-degree \( \text{in-degree}(v) = \sum_{u \in V} \text{adj}(u) \) \( \in O(V^2) \)
  - Out-degree \( \text{out-degree}(v) = \sum_{u \in V} \text{adj}(u) \) \( \in O(V^2) \)
  - Adjacency matrix of \( G \): \( A = (a_{ij}) \) where \( a_{ij} = 1 \) if \( (v_i, v_j) \in E \), otherwise \( a_{ij} = 0 \).

- **Digraphs**:
  - Adjacency matrix of \( G \): \( A \) where \( a_{ij} = 1 \) if \( (v_i, v_j) \in E \), otherwise \( a_{ij} = 0 \).
- Weighted graphs:
  - \( w: E \rightarrow \mathbb{R} \) is the weight function.
  - Adjacency Matrix: \( a_{ij} = \begin{cases} w((i,j)), & \text{if } (i,j) \in E \\ 0/0/\text{null}, & \text{otherwise} \end{cases} \)
  - Adjacency List: Store \( w((u,v)) \) with vertex \( v \) in the adjacency list.
**Depth-First Search (DFS):**

**Given:** A graph \( G = (V, E) \), directed/undirected, and a source vertex \( v \in V \).

**Goal:** Explore each vertex (and edge) systematically, recursively.

- Use a counter (i.e., a clock) to timestamp vertices. Clock is global.
- Time stamp:
  - \( d(v) = \) discovery time i.e. the node is seen for the first time.
  - \( f(v) = \) finishing time i.e. the entire set of edges reachable from \( v \) have already been visited, search goes back (i.e., backtracks) from \( v \).
- \( d(v) \) and \( f(v) \) are unique integers from 1 to \( 2|V| \). \( \forall v \in V, d(v) < f(v) \).

**DFS(G):**

1. \( \forall v \in V \)
2. \( \text{color}(v) \leftarrow \text{WHITE} \)
3. \( \text{TTW} \leftarrow \text{NIL} \)
4. \( \text{clock} \leftarrow 0 \)
5. \( \forall v \in V \)
6. \( \text{if color}(v) = \text{WHITE} \)
7. \( \text{DFS-VISIT}(v) \)

**DFS-VISIT(v):**

1. \( \text{color}(v) \leftarrow \text{GRAY} \) \( \leftarrow \) white vertex \( u \) discovered
2. \( \text{clock} \leftarrow \text{clock} + 1 \) \( \leftarrow \) timestamp the discovery time
3. \( \text{TTW} \leftarrow \text{clock} \) \( \leftarrow \)
4. \( \forall \ e \in A_{G}(v) \) \( \leftarrow \) Explore all edges \( (v, e) \)
5. \( \text{if color}(v) = \text{WHITE} \)
6. \( \text{TTW} \leftarrow u \)
7. \( \text{DFS-VISIT}(v) \)
8. \( \text{color}(v) \leftarrow \text{BLACK} \) \( \leftarrow \) \( v \) is finished
9. \( \text{clock} \leftarrow \text{clock} + 1 \)
10. \( \text{TTW} \leftarrow \text{clock} \) \( \leftarrow \) timestamp the finalization time.

**Time Complexity:**

- Lines 1-3 & 5-7 run \( O(|V|) \) times.
- DFS-visit runs once per edge in directed graph, twice per edge in undirected graph.
- \( \text{Time} = O(E + V) \) at 0.
Example:

1. Tree edge
2. Back edge
3. Forward edge
4. Cross edge

Diagram:

- Tree edges (T)
- Back edges (B)
- Forward edges (F)
- Cross edges (C)
Properties of DFS:

**Theorem:** (Parenting theorem)

In DFS of a graph, for u, v, exactly one of the following holds:

1. \((d_u < f_v) < (d_v < f_u)\) ? neither u nor v are descendant
   or \(d_v < f_u\) if \(d_u < f_v\) of each other
2. \((d_u < d_v < f_v < f_u)\) \(v\) is a descendant of \(u\)
3. \((d_v < d_u < f_u < f_v)\) \(u\) is a descendant of \(v\)

So BAD NESTING allowed!

**Corollary:** (Nesting of descendants' descants)

\(v\) is a proper descendant of \(u\) if \(d_v < d_u < f_v < f_u\).

**Theorem:** (White-paste theorem)

\(v\) is a descendant of \(u\) if \(u\) is at time \(d_u\), \(v\) past \(u\) and \(v\) consists of white vertices (except for \(v\), which is \(G\)RAY).

**Classification of Edges:**

* **Tree Edge:** (GRAY to WHITE) MEMBER of DFS spanning forest.
  Edge \((u,v)\) is a tree edge if found by exploring edge \((u,v)\).
* **Back Edge:** (GRAY to GRAY) Parent to descendant to ancestor.
* **Forward Edge:** (GRAY to BLACK) non-tree edge from ancestor to descendant
* **Cross Edge:** (GRAY to BLACK) Remainder of the edges. If in the
  same tree cross edges cannot go from ancestor to descendant, or
  vice versa. They are the edges between tree tops in a
  DFS forest.

[See previous page for a figure]
Theorem: (No forward/cover edges in undirected graphs)

In a DFS of an undirected graph, all edges are classified as either tree edges or back edges.

DFS can be used to find cycles in graphs.

- Presence of a back-edge $\Rightarrow$ cycle.

- Directed graph. Takes $O(V+E)$ time.
- Undirected graph. Takes $O(V)$ time.
  
- If $n$ edges in the graph with $n$ vertices,
then one edge must be a back edge.