Graph Theory and Algorithms

Lecture 11: Graph Representations and Properties

Instructor: Dr. Mudassir Shabbir

ITU

February 26, 2025

Scribe: Safwan Ahmad & Shariq Shoaib

Graph Definition

Definition: A graph is a pair of two sets $G(V, E)$ where:

- V is the set of vertices.
- E is the set of edges, a subset of $V \times V$.

Graph means:

- **Undirected**: Edges have no direction.
- **Unweighted**: Edges have no weights.
- **Simple**: No multiple edges or loops.

Types of Graphs:

- **Directed Graph**: $(u, v) \in E \neq (v, u) \in E$. (e.g., Twitter)
- **Undirected Graph**: $(u, v) \in E = (v, u) \in E$. (e.g., Facebook, LinkedIn)
- **Weighted Graph**: Graph with weights on edges, $W : E \to \mathbb{R}$.
- **Simple Graph**: No multiple edges or loops.

Connectivity of Graph

Example Graph:

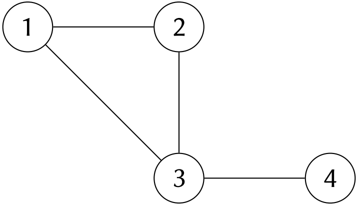


Figure 1

Connectivity of Graph: A graph is connected if every pair of vertices has a path.

- If $(u, v) \in E$, then u and v are **Adjacent** or **Incident**.
- Every consecutive pair in a walk is adjacent.

Walk

Walk: A walk in a graph is a sequence of vertices and edges where both edges and vertices can be repeated.

- Formally, $\langle V_1, V_2, V_3, \ldots, V_k \rangle$ where $(V_i, V_{i+1}) \in E$.
- **Example:** in Figure 1 $\langle 1, 2, 3, 4 \rangle$ is a walk.
- **Non-Example:** in Figure 1 $\langle 1, 4 \rangle$ is not a walk (no direct edge).

Key points to note about a walk:

- Edges can be repeated.
- Vertices can be repeated.

Path and Simple Path

Path: A walk with no repeated edges.

- **Example:** in Figure 1 $\langle 1, 2, 3 \rangle$ is a path.

Simple Path: a path in a graph which does not have repeating vertices.

- **Example:** in Figure 1 $\langle 1, 2, 3 \rangle$ is a simple path.

Length of a Path

Definition: For a path $P = \langle v_1, v_2, \ldots, v_k \rangle$, the length is the number of edges involved.

- Length $l = k - 1$, where k is the total number of vertices.

Examples:

- $\langle 1, 4 \rangle$ has length $l = 1$ (1 edge).
- $\langle 1 \rangle$ has length $l = 0$ (no edges).

Shortest Path:

- For an **unweighted graph**, the shortest path is the path from v_1 to v_k with the minimum number of edges.
- **Example:** In Figure 1, the shortest path from vertex 1 to vertex 4 is $\langle 1, 3, 4 \rangle$ with 2 edges.

- For a **weighted graph**, the shortest path is the path from v_1 to v_k where the sum of edge weights is minimized.
- **Example:** Suppose the weights on the edges in Figure 1 are:
  - (1,2): 3, (1,3): 1, (2,3): 1, (3,4): 2

  Then the shortest path from vertex 1 to vertex 4 is $\langle 1, 3, 4 \rangle$ with total weight $1 + 2 = 3$.

Cycle, Simple Cycle

Cycle: A path where the first and last vertex are the same.

- **Example:** in Figure 1 $\langle 1, 2, 3, 1 \rangle$ is a cycle.

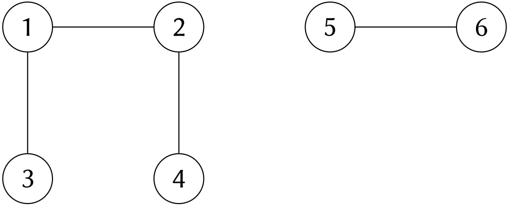
Simple Cycle: A simple path where the first and last vertex are the same.

- **Example:** in Figure 1 $\langle 1, 2, 3, 1 \rangle$ is a simple cycle.

Forest

Forest: A forest is a graph with no cycles.

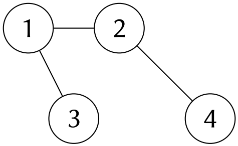
- A forest can consist of multiple disconnected trees.
- **Example:** The graph below is a forest with two trees.



Connected

Connected:

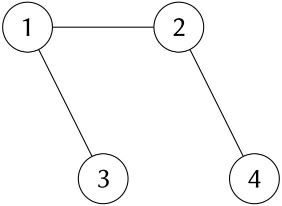
- Two vertices $u, v \in V$ are connected if there is a walk from u to v.
- A graph G is connected if every pair of vertices $u, v \in V$ is connected.
- **Key Points:**
  - If there is an edge between two vertices, they are connected and adjacent.
  - If a path exists between two vertices, they are connected but not necessarily adjacent.
  - They are adjacent only if there is a direct edge between them.
- **Example:** In the graph below, vertices 2 and 3 are connected but not adjacent.



Tree and Examples

Tree: A tree is a connected forest.

- A tree is a connected graph with no cycles.
- **Example:** The graph below is a tree.



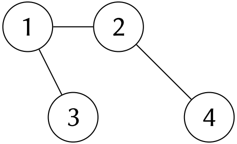
Additional Examples:

- **Connected but Not Adjacent:** In the tree above, vertices 1 and 4 are connected (path: $\langle 1, 2, 4 \rangle$) but not adjacent.
- **Adjacent and Connected:** In the tree above, vertices 1 and 2 are both connected and adjacent.

Neighbours and Degrees

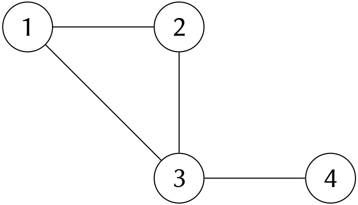
Definition: If $(u, v) \in E$, then v is a neighbour of u.

- For directed graphs, $(u, v) \in E$ means v is an out-neighbour of u and u is an in-neighbour of v.
- $N(u) = \{v : (u, v) \in E\}$
- $|N(u)| = d(u)$, where $d(u)$ is the degree of u.



- $N(1) = \{2, 3\}$ and $d(1) = 2$.
- $N(2) = \{1, 4\}$ and $d(2) = 2$.
- $N(3) = \{1\}$ and $d(3) = 1$.
- $N(4) = \{2\}$ and $d(4) = 1$.

Representation



MST Graph:

- Nodes: 1, 2, 3, 4

Adjacency List

Adjacency List:

Vertex	Adjacent Vertices
1	2, 3
2	1, 3
3	1, 2, 4
4	3

Explanation:

- Each vertex is listed with its adjacent vertices.
- For example, vertex 1 is connected to vertices 2 and 3.

Adjacency Matrix

Adjacency Matrix:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Explanation:

- Rows and columns represent vertices 1, 2, 3, 4.
- A ‘1‘ indicates an edge between the corresponding vertices.
- For example, the edge between 1 and 2 is represented by a ‘1‘ in the first row, second column.

DFS Algorithm

Depth First Search (DFS):

- DFS is a graph traversal algorithm.
- It explores as far as possible along each branch before backtracking.
- Used in applications like:
  - Pathfinding
  - Cycle detection
  - Topological sorting
  - Solving puzzles and mazes

DFS Algorithm Steps

How DFS Works:

- Start at a source vertex $V1$.
- Mark $V1$ as visited.
- For each neighbor $v \in N(V1)$:
  - If v is not visited, recursively apply DFS to v.

DFS Algorithm: Without Wrapper

Pseudocode for DFS (Connected Graph):

- Use it when the graph is connected and a single start vertex is sufficient.
- Traversal starts from a source node and explores all reachable vertices.

```
procedure DFS(V)
    V.visited ← True
    for each neighbor u ∈ N(V) do
        if u.visited = False then
            DFS(u)
        end if
    end for
end procedure
```

Pseudocode for DFS (Disconnected Graph):

- Wrapper function ensures that **all graph components** are visited.
- Used when the graph may have disconnected parts.

```
procedure DFS-WRAPPER(Graph G)
    for each vertex V ∈ G do
        if V.visited = False then
            DFS(V)
        end if
    end for
end procedure
```

DFS Algorithm: Recursive DFS Function

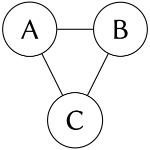
Inner DFS Procedure:

- Recursively visits all vertices reachable from the starting vertex.
- Called by the wrapper for unvisited vertices.

```
procedure DFS(V)
    V.visited ← True
    for each neighbor u ∈ N(V) do
        if u.visited = False then
            DFS(u)
        end if
    end for
end procedure
```

DFS Example – Connected Graph

Connected Graph:

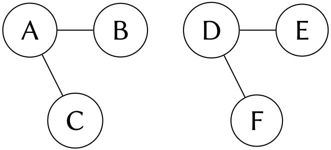


DFS Process:

- Start at A: Visit A $\rightarrow$ B $\rightarrow$ C.
- Traversal order may vary depending on neighbor order.

DFS Example – Disconnected Graph

Disconnected Graph:



DFS Process:

- Start at A: Visit A $\rightarrow$ B $\rightarrow$ C.
- Since the graph is disconnected, apply DFS again from D: Visit D $\rightarrow$ E $\rightarrow$ F.

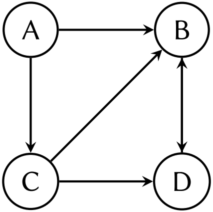
DFS Edge Classifications – Definitions & Explanations

Types of Edges in DFS:

- **Tree Edge**: An edge (u, v) where v is unvisited when (u, v) is explored. These form the DFS tree and represent discovery of new vertices.
- **Back Edge**: An edge (u, v) that connects u to an ancestor v in the DFS tree (v.visited = True). Indicates a cycle if the graph is directed.
- **Forward Edge**: An edge (u, v) where v is a descendant of u in the DFS tree and already visited. Traverses deeper parts of the tree, but not new vertices.
- **Cross Edge** (optional): An edge (u, v) that connects nodes in separate branches or components (not ancestor/descendant). Typically occurs in directed graphs during DFS.

DFS Edge Classifications – Example

Graph:



Edge Classifications:

- **Tree Edges**: $(A, B), (B, D), (A, C)$
- **Back Edge**: (D, B)
- **Forward Edge**: (C, D)
- **Cross Edge**: (C, B)

BFS Algorithm:

- Start at a vertex U.
- Use a queue Q to manage vertices.
- Mark the starting vertex as visited and enqueue it.
- While the queue is not empty, dequeue a vertex and explore its neighbors.
- Mark and enqueue unvisited neighbors.

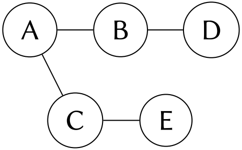
BFS Algorithm: Mathematical Form

Mathematical Form:

```
procedure BFS(U, Q)
    for each vertex v in graph: v.visited ← False        ▷ Initialize visited
    Push(U, Q)
    U.visited ← True
    while Q ≠ ∅ do
        U ← Pop(Q)
        for each neighbor Y ∈ N(U) do
            if Y.visited = False then
                Y.visited ← True
                Push(Y, Q)
            end if
        end for
    end while
end procedure
```

BFS Example: Connected Graph

Connected Graph:



BFS Process:

- Start at A: Visit A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D $\rightarrow$ E.

BFS Queue Process

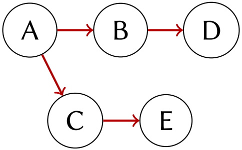
Queue Process:

- Initialize: $Q = [A]$
- Step 1: Pop A, enqueue B and C. $Q = [B, C]$
- Step 2: Pop B, enqueue D. $Q = [C, D]$
- Step 3: Pop C, enqueue E. $Q = [D, E]$
- Step 4: Pop D. $Q = [E]$
- Step 5: Pop E. $Q = []$

BFS Tree Edges

Tree Edges:

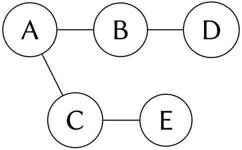
- Tree edges are the edges that connect a vertex to its unvisited neighbors.
- In BFS, tree edges always go from a vertex at level L to a vertex at level $L + 1$.
- Example: A $\rightarrow$ B, A $\rightarrow$ C, B $\rightarrow$ D, C $\rightarrow$ E.



BFS Cross Edges

Cross Edges:

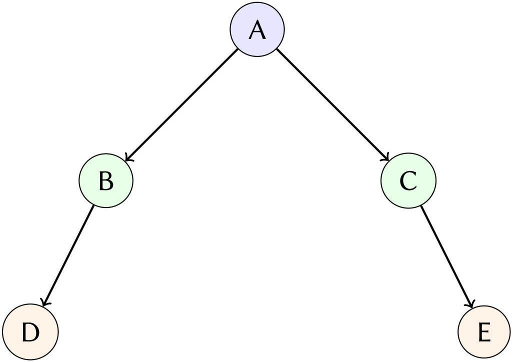
- Cross edges are edges between vertices at the same level.
- In BFS, cross edges do not exist in trees but can exist in general graphs.
- Example: None in this graph (since it is a tree).



BFS Levels

BFS Levels:

- Level 0: A
- Level 1: B, C
- Level 2: D, E



Level 0

Level 1

Level 2

Analysis of DFS and BFS

Time Complexity of DFS and BFS:

- Both DFS and BFS visit each vertex once and each edge once.
- Let V be the number of vertices and E be the number of edges.
- **DFS Time Complexity:** $O(V + E)$
  - Each vertex is visited once: $O(V)$
  - Each edge is traversed once: $O(E)$
- **BFS Time Complexity:** $O(V + E)$
  - Each vertex is enqueued and dequeued once: $O(V)$
  - Each edge is traversed once: $O(E)$

Next Class

Topics:

- New examples beyond connectivity.
- Applications of DFS and BFS in real-world problems.