GATE Data Science and AI Study Materials

Graph Theory

by Piyush Wairale

Instructions:

- Kindly go through the lectures/videos on our website www.piyushwairale.com
- Read this study material carefully and make your own handwritten short notes. (Short notes must not be more than 5-6 pages)
- Attempt the mock tests available on portal.
- Revise this material at least 5 times and once you have prepared your short notes, then revise your short notes twice a week
- If you are not able to understand any topic or required a detailed explanation and if there are any typos or mistake in study materials. Mail me at piyushwairale100@gmail.com

For GATE DA Course & Test Series , visit: www.piyushwairale.com

Contents

1 Graph Theory **4**

1.1 Key Terminology 5

2 Directed Graph **10**

3 In-Degrees and Out-Degrees of Vertices of a Digraph **12**

4 Topological Sorting in Graph Theory **13**

5 Example: **14**

6 Graph Traversal: DFS and BFS **15**

6.1 Depth-First Search (DFS) 15

6.2 Breadth-First Search (BFS) 16

7 Spanning Tree **18**

7.1 Minimum Spanning Tree (MST) 18

7.2 Kruskal's Algorithm: 18

7.3 Prim's Algorithm: 19

8 Shortest Path **20**

8.1 Dijkstra's Algorithm for Non-Negative Weights 20

9 Adjacency Matrix **21**

Piyush Wairale

LinkedIn

Youtube Channel

Instagram

Telegram Group

Facebook

Download Andriod App

1 Graph Theory

A graph is one type of nonlinear data structure.

Graph and Multi-graphs

A graph G consists of:

1. A set V of elements called **nodes** (or points or vertices).
2. A set E of edges such that each edge e in E is identified with a unique unordered pair $[u, v]$ of nodes in V, denoted by $e = [u, v]$.

We denote a graph as $G = (V, E)$. Let's assume $e = [u, v]$:

- The nodes u and v are called the **endpoints** of e, and u and v are said to be **adjacent nodes** or **neighbors**.
- The **degree** of a node u (denoted as $\deg(u)$) is the number of edges containing u.
- If $\deg(u) = 0 \implies u$ does not belong to any edge, then u is called an **isolated node**.

Important Points

- **Connected Graph:** A graph G is said to be connected if there is a path between any two of its nodes.
- **Complete Graph:** A graph G is said to be complete if every node u in G is adjacent to every other node v in the graph. A complete graph T without any cycle is called a **tree graph** or simply a **tree**.
A tree is not always a complete graph, which means that, in particular, there is a unique simple path P between any two nodes u and v in T. If T is a finite tree with m nodes, then T will have $m - 1$ edges.
- **Labeled Graph:** A graph G is said to be labeled if its edges are assigned with some data.
- **Weighted Graph:** A graph G is said to be weighted if each edge e in G is assigned a nonnegative numerical value $w(e)$ called the weight or length of e.
In such a case, each path P in G is assigned a weight or length, which is the sum of the weights of the edges along the path P. If no other information about weights is given, we may assume any graph G to be weighted by assigning the weight $w(e) = 1$ to each edge e in G.
- **Multiple Edges:** Distinct edges e and e' are called multiple edges if they connect the same endpoints, i.e., if $e = [u, v]$ and $e' = [u, v]$. The graph containing multiple edges between two vertices is called a **multi-graph**.
- **Loops:** An edge e is called a loop if it has identical endpoints, i.e., $e = [u, u]$.
- **Finite Multi-graph:** A multi-graph M is said to be finite if it has a finite number of nodes and a finite number of edges.
- **Finite Graph:** A graph G with a finite number of nodes must automatically have a finite number of edges. This is not necessarily true for a multi-graph M, since M may have multiple edges.
- **Distance:** The distance, denoted by $d(u, v)$, between two vertices u and v is defined as the length of the shortest path joining u and v.
- **Directed Path:** A path is said to be a directed path if all arcs of the path are directed in the same direction. For example, $v_2 e_2 v_4 e_3 v_3$ is a directed path, but $v_1 e_1 v_2 e_2 v_4$ is not a directed path. Also, v_2 and v_4 are adjacent vertices, but v_2 and v_3 are not adjacent.

1.1 Key Terminology

- **Graph (G):** A set of vertices (V) and edges (E), denoted as $G = (V, E)$.
- **Vertex (Node):** A fundamental unit represented as a point in the graph.
- **Edge (Link):** A connection between two vertices.
- **Directed Graph (Digraph):** A graph where edges have a direction.
- **Undirected Graph:** A graph where edges do not have a direction.
- **Weighted Graph:** A graph where edges have weights.
- **Path:** A sequence of vertices connected by edges.
- **Cycle:** A path that starts and ends at the same vertex without repeating edges.
- **Degree:** The number of edges incident to a vertex.
  - **In-degree:** The number of edges directed towards a vertex.
  - **Out-degree:** The number of edges directed away from a vertex.

Examples 1

Let $V = \{a, b, c, d\}$ and

$$X = \{\{a, b\},\ \{a, c\},\ \{a, d\}\}.$$

Then

$$G = \{V, X\}$$

is a $(4, 3)$ graph. This graph can be represented by a diagram as shown in Fig. 1.1.

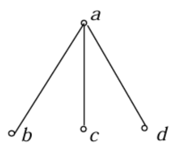


Fig. 1.1

In this graph the points a and b are adjacent whereas b and c are non – adjacent.

Example 2

Let $V = \{1, 2, 3, 4\}$ and

$$X = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}.$$

$G = (V, X)$ is a $(4, 6)$ graph.

This graph is represented by the diagram as shown in Fig. 1.2.

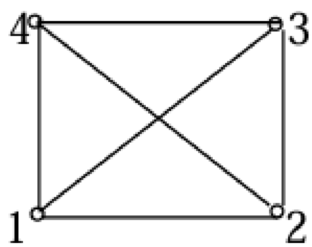


Fig. 1.2

In this graph, the lines $\{1, 3\}$ and $\{2, 4\}$ intersect in the diagram, and their intersection is not a point of the graph.

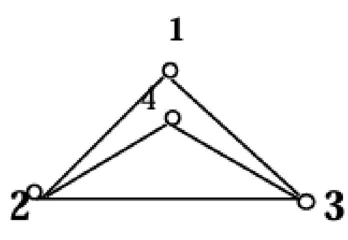


Fig. 1.3

For GATE DA Course & Test Series , visit: www.piyushwairale.com

Example 3

The $(10, 15)$ graph given in Fig. 1.4 is called a *Petersen graph.*

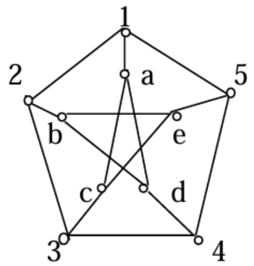


*Fig.*1.4

Remark:

The definition of a graph does not allow more than one line joining two points. Also, it does not allow any line joining a point to itself.

Line joining points to itself is called a ***loop.*** Fig.1.5 is a loop.

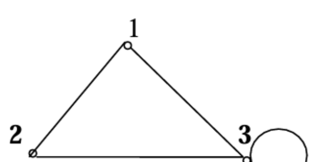


*Fig.*1.5

(2). MULTI GRAPH

Definition: If more than one line joining two vertices are allowed then the resulting object is called a ***multi graph***. Lines joining the same points are called ***multiple lines***.

Fig. 1.6 is an example of a multi graph.

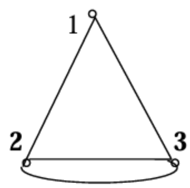


Fig. 1.6

(3). PSEUDO GRAPH

Definition: If an object contains multiple lines and loops then it is called a ***pseudo graph***.

Fig. 1.7 is a pseudo graph.

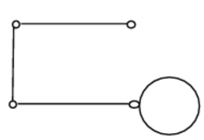


Fig. 1.7

Note: If G is a (p, q) graph then $q \leq p\,C_2$ and $q = p\,C_2$ if and only if any two distinct points are disjoint.

(4).COMPLETE GRAPH

Definition: A graph in which any two distinct points are adjacent is called a ***complete graph***.

A complete graph with p vertices is denoted by K_p.

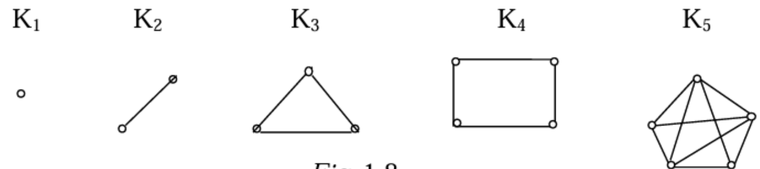


Fig. 1.8

Note: The number of edges of a complete graph K_p is $p\,C_2$.

(5). NULL GRAPH

Definition: A graph G whose edge set is empty is called a ***null graph*** or a ***totally disconnected graph.***

Example: G_1, G_2, G_3 and G_4 are null graphs.

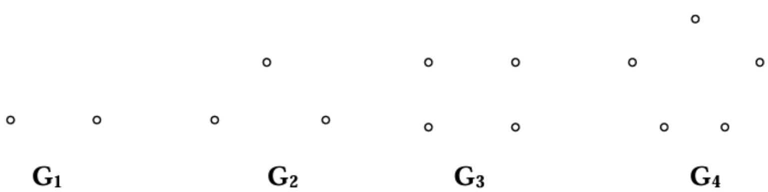


Fig. 1.9

(6).LABELLED GRAPH

Definition: A graph G is called ***labelled*** if its p points are distinguished from one another by names such as $v_1, v_2, \ldots, v_p$.

The graphs given in Fig. 1.1 and Fig.1.2 are labelled graphs and the graph in Fig. 1.8 is an unlabelled graph.

For GATE DA Course & Test Series , visit: www.piyushwairale.com

(7).BIPARTITE GRAPH

Definition: A graph G is called a ***bigraph*** or ***bipartite graph*** if the vertex set V can be partitioned into two disjoint subsets V_1 and V_2 such that every line of G joins a point of V_1 to a point of V_2. (V_1, V_2) is called a ***bipartition*** of G.

(8).COMPLETE BIPARTITE GRAPH

Definition: A graph G is called a ***complete bipartite graph*** if the vertex set V can be portioned into two disjoint subsets V_1 and V_2 such that every line joining the points of V_1 to the points of V_2. If V_1 contains *m* points V_2 contains *n* points then the complete bigraph is denoted by $K_{m, n}$.

The complete graph $K_{1, n}$ is called a ***star*** for $n \geq 1$.

2 Directed Graph

A directed graph G, also called a digraph or graph, is the same as a simple graph except that each edge e in G is assigned a direction or, in other words, each edge e is identified with an ordered pair (u, v) of nodes in G.

- Suppose G is a directed graph with a directed edge $e = (u, v)$. Then e is also called an arc. The following terminology is used with respect to directed graphs:
  - u is the origin or initial point of e and v is the destination or terminal point of e.
  - u is the predecessor of v and v is a successor or neighbor of u.
  - u is adjacent to v, but v is not adjacent to u.
- The **out-degree** of a node u in G (denoted as outdeg(u)) is the number of edges beginning at u. The **in-degree** of u (denoted as indeg(u)) is the number of edges ending at u.
- A node u is called a **source** if it has a positive out-degree but zero in-degree. Similarly, u is called a **sink** if it has a zero out-degree but a positive in-degree.
- A node v is said to be **reachable** from a node u if there is a (directed) path from u to v.
- A directed graph G is said to be **connected** or **strongly connected** if for each pair u, v of nodes in G, there is a path from u to v and there is also a path from v to u.
- A graph G is said to be **unilaterally connected** if for any pair u, v of nodes in G, there is a path from u to v or a path from v to u.

For GATE DA Course & Test Series , visit: www.piyushwairale.com

Example

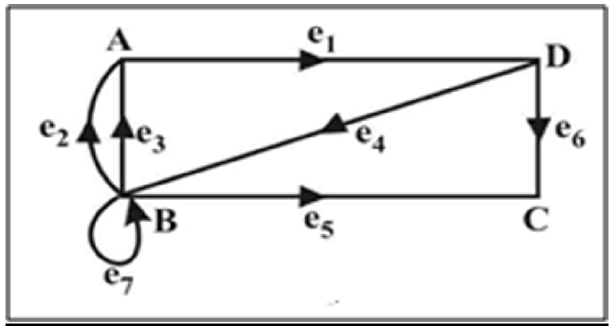


Fig. 1: Directed Graph

The graph in Fig. 1 consists of:

1. 4 nodes and 7 (directed) edges.
2. The edges e_2 and e_3 are said to be in parallel since each begins at B and ends at A.
3. The edge e_7 is a loop since it begins and ends at the same point B.
4. The sequence $P_1 = (B, C, B, A)$ is not a path since (C, B) is not an edge. However, $P_2 = (D, B, A)$ is a path from D to A, since (D, B) and (B, A) are edges. Thus, A is reachable from D.
5. There is no path from C to any other node, so G is not strongly connected. However, G is unilaterally connected. Also:
   - $\text{indeg}(D) = 1$ and $\text{outdeg}(D) = 2$.
   - Node C is a sink since $\text{indeg}(C) = 2$ but $\text{outdeg}(C) = 0$.
   - No node in G is a source.

Advantages of Directed Graphs

- Let T be any nonempty tree graph. Suppose we choose any node R in T. Then T with this designated node R is called a **rooted tree**, and R is called its **root**. This defines a direction to the edges in T, so the rooted tree T may be viewed as a directed graph.
- Suppose we also order the successors of each node v in T. Then T is called an **ordered rooted tree**. Ordered rooted trees are almost the same as general trees.
- A directed graph G is said to be simple if G has no parallel edges. A simple graph G may have loops, but it cannot have more than one loop at a given node.

3 In-Degrees and Out-Degrees of Vertices of a Digraph

Consider the following graph:

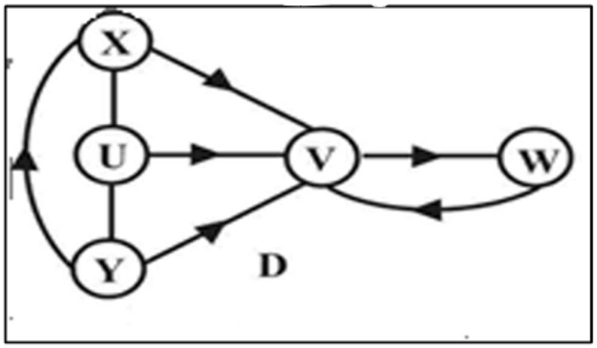


Fig. 2: In-degree and Out-degree

Let us consider a vertex U of a digraph D. The **in-degree** of U is defined as the number of arcs for which U is the head, and the **out-degree** is the number of arcs for which U is the tail.

Note:

$$\text{indeg}(U) \iff d_D^+(U) \iff \text{in-degree of } U \text{ in graph } D$$
$$\text{outdeg}(U) \iff d_D^-(U) \iff \text{out-degree of } U \text{ in graph } D$$

Null Graph

A graph is said to be **null** if all its vertices are isolated.

Finite Graphs

A multi-graph is said to be **finite** if it has a finite number of vertices and a finite number of edges.
Note: A simple graph with a finite number of vertices must automatically have a finite number of edges and so must be finite.

Trivial Graph

The finite graph with one vertex and no edges, i.e., a single point, is called the **trivial graph**.

Subgraphs

Consider a graph $G = G(V, E)$. A graph $H = H(V', E')$ is called a **subgraph** of G if the vertices and edges of H are contained in the vertices and edges of G, i.e., if $V' \subseteq V$ and $E' \subseteq E$.

Advantages of Subgraphs

1. A sub-graph $H(V', E')$ of $G(V, E)$ is called the **sub-graph induced by its vertices** V' if its edge set E' contains all edges in G whose endpoints belong to vertices in H.

2. If v is a vertex in G, then $G - v$ is the sub-graph of G obtained by deleting v from G and deleting all edges in G which contain v.

3. If e is an edge in G, then $G - e$ is the sub-graph of G obtained by simply deleting the edge from G.

4 Topological Sorting in Graph Theory

Topological sorting is a linear ordering of vertices in a directed acyclic graph (DAG) such that for every directed edge (u, v), vertex u appears before v in the ordering. Topological sorting is used in scenarios where there is a dependency among items, such as task scheduling, course prerequisites, and build systems.

Properties of Topological Sorting

- Topological sorting is only possible for directed acyclic graphs (DAGs).
- A graph can have more than one valid topological ordering.
- If a graph contains a cycle, topological sorting is not defined for it.

Algorithm for Topological Sorting

Kahn's Algorithm

Kahn's algorithm is an iterative approach that uses in-degree information for vertices.

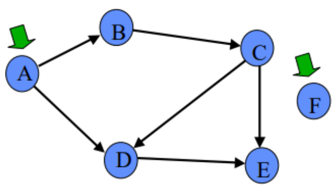
1. Compute the in-degree of each vertex in the graph.
2. Initialize a queue with all vertices that have an in-degree of 0.
3. While the queue is not empty:
   - Remove a vertex u from the queue and add it to the topological order.
   - For each neighbor v of u, reduce the in-degree of v by 1.
   - If the in-degree of v becomes 0, add v to the queue.
4. If all vertices are processed, the topological order is complete. If there are unprocessed vertices, the graph contains a cycle.

5 Example:

Topological Sort Algorithm

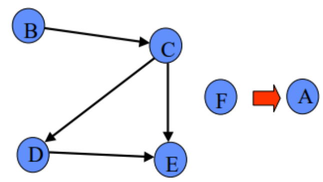
Step 1: Identify vertices that have no incoming edge

- The "**in-degree**" of these vertices is zero



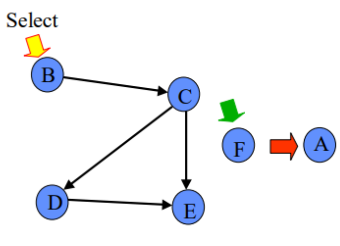
Topological Sort Algorithm

Step 2: Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.



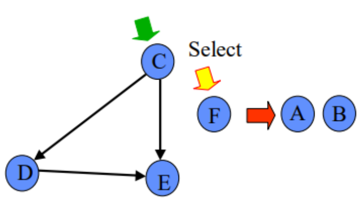
Topological Sort Algorithm

Repeat Steps 1 and Step 2 until graph is empty

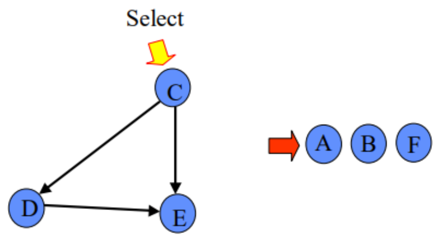


Topological Sort Algorithm

Repeat Steps 1 and Step 2 until graph is empty



Repeat Steps 1 and Step 2 until graph is empty



Repeat Steps 1 and Step 2 until graph is empty

Final Result:

6 Graph Traversal: DFS and BFS

Graph traversal refers to the process of visiting all the vertices and edges of a graph systematically. Traversals are used in various applications such as searching, shortest path finding, and connectivity checking. The two most common graph traversal techniques are Depth-First Search (DFS) and Breadth-First Search (BFS).

6.1 Depth-First Search (DFS)

Overview

Depth-First Search (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It uses a stack (explicitly or implicitly via recursion) to keep track of vertices to visit next.

Algorithm

1. Start from a source vertex v.
2. Mark v as visited.
3. Recursively visit all unvisited neighbors of v.
4. If all neighbors are visited, backtrack to the previous vertex.

Implementation

Recursive DFS:

```
DFS(vertex v):
    Mark v as visited
    For each neighbor u of v:
        If u is not visited:
            DFS(u)
```

Iterative DFS:

```
DFS(start):
    Create a stack S and push start onto it
    While S is not empty:
        Pop a vertex v from S
```

```
        If v is not visited:
            Mark v as visited
            Push all unvisited neighbors of v onto S
```

Properties

- **Time Complexity:** $O(V + E)$, where V is the number of vertices and E is the number of edges.
- **Space Complexity:** $O(V)$ for the recursion stack or explicit stack.
- Suitable for exploring all components of a graph.

6.2 Breadth-First Search (BFS)

Overview

Breadth-First Search (BFS) is a graph traversal algorithm that explores all neighbors of a vertex before moving on to their neighbors. It uses a queue to keep track of vertices to visit next.

Algorithm

1. Start from a source vertex v.
2. Mark v as visited and enqueue it.
3. While the queue is not empty:
    - Dequeue a vertex u.
    - Visit all unvisited neighbors of u and enqueue them.

Implementation

```
BFS(start):
    Create a queue Q and enqueue start
    Mark start as visited
    While Q is not empty:
        Dequeue a vertex v from Q
        For each neighbor u of v:
            If u is not visited:
                Mark u as visited
                Enqueue u
```

Properties

- **Time Complexity:** $O(V + E)$, where V is the number of vertices and E is the number of edges.
- **Space Complexity:** $O(V)$ for the queue.
- Finds the shortest path (in terms of the number of edges) in an unweighted graph.

For GATE DA Course & Test Series , visit: www.piyushwairale.com

S. No	BFS	DFS
1	BFS stands for Breadth-First Search.	DFS stands for Depth-First Search.
2	BFS uses Queue data structure for finding the shortest path.	DFS uses Stack data structure.
3	BFS can be used to find the single-source shortest path in an unweighted graph, as it reaches a vertex with the minimum number of edges from a source vertex.	DFS might traverse through more edges to reach a destination vertex from a source.
4	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions far from the source.
5	BFS considers all neighbors first and is therefore not suitable for decision-making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems where we make a decision, then explore all paths through it. If this leads to a solution, we stop.
6	The time complexity of BFS is $O(V + E)$, where V stands for vertices and E stands for edges.	The time complexity of DFS is also $O(V + E)$, where V stands for vertices and E stands for edges.

7 Spanning Tree

A spanning tree of a graph is a subgraph that includes all the vertices of the original graph and is a tree. In other words, a spanning tree is a subset of the graph that:
1. Contains all the vertices of the original graph.
2. Is connected.
3. Has no cycles.

For an undirected graph with n vertices, a spanning tree will have $n-1$ edges. If there are more edges, the subgraph will contain cycles, and if there are fewer edges, the subgraph will not be connected.

7.1 Minimum Spanning Tree (MST)

A minimum spanning tree (MST) is a spanning tree of a weighted graph such that the sum of the weights of its edges is as small as possible. In other words, an MST is a spanning tree with the minimum possible total edge weight.
Given a connected weighted graph G, it is often desired to create a spanning tree T for G such that sum of the weights of the tree edges in T is as small as possible. Such a tree is called a minimum spanning tree.

Properties of MST

1. **Uniqueness:** If all edge weights in the graph are distinct, then there will be exactly one unique MST.
2. **Number of MSTs:** If some edge weights are equal, there may be multiple MSTs for the same graph.

Algorithms to Find MST

(Do watch the lectures for better understanding)

7.2 Kruskal's Algorithm:

- Like Prim's algorithm, Kruskal's algorithm also constructs the minimum spanning tree of a graph by adding edges to the spanning tree one by one. At all points during its execution, the set of edges selected by Prim's algorithm forms exactly one tree.
- The set T of edges is initially empty. As the algorithm progresses, edges are added to T. So long as it has not found a solution, the partial graph formed by nodes of G and edges in T consists of several connected components.
- The elements of T included in a given connected component form a minimum spanning tree for the nodes in the component. At the end of the algorithm, only one connected component remains, so T is then a minimum spanning tree for all the nodes of G.
- We observe the edges of G in order of increasing length. If an edge joins two nodes in different connected components, we add it to T. The two connected components now form only one component. Otherwise, the edge is rejected as it joins two nodes in the same connected component and therefore cannot be added to T without forming a cycle (because the edges in T form a tree for each component). The algorithm stops when only one connected component remains.

Algorithms

- Sort all edges in the graph by their weights.
- Initialize an empty subgraph.
- Add edges one by one to the subgraph, starting from the edge with the smallest weight.
- Only add edges that do not form a cycle in the subgraph.
- Stop when there are $n-1$ edges in the subgraph (where n is the number of vertices).

7.3 Prim's Algorithm:

- An arbitrary node is chosen initially as the tree root.
  Note: In an undirected graph and its spanning tree, any node can be considered the tree root, and the nodes adjacent to it are treated as its sons.
- The nodes of the graph are then appended to the tree one at a time until all nodes of the graph are included.

Algorithms

- Start from any vertex and initialize an empty subgraph.
- Grow the MST one edge at a time by adding the smallest edge that connects a vertex in the MST to a vertex outside the MST.
- Repeat until all vertices are included in the MST.

Complexity and Optimization

- If the graph is represented by an adjacency matrix, each `for` loop in Prim's algorithm must examine $O(n)$ nodes. Since the algorithm contains a nested `for` loop, its time complexity is $O(n^2)$.
- Prim's algorithm can be made more efficient by maintaining the graph using adjacency lists and keeping a priority queue of the nodes not in the partial tree.

8 Shortest Path

A **shortest path** from u to v is a path of minimum weight from u to v. The **shortest-path weight** from u to v is defined as

$$\delta(u, v) = \min w(p) : p \text{ is a path from } u \text{ to } v.$$

Note: $\delta(u, v) = \infty$ if no path from u to v exists.

The **single-source shortest path problem** (SSSP) is to find the shortest path from a single source vertex s to every other vertex in the graph.

Output of the Algorithm

The output of this algorithm can either be the $n - 1$ numbers giving the weights of the $n - 1$ shortest paths or (some compact representation of) these paths.

We first consider Dijkstra's algorithm for the case of non-negative edge weights, and then give the Bellman-Ford algorithm that handles negative weights as well.

8.1 Dijkstra's Algorithm for Non-Negative Weights

Dijkstra's algorithm keeps an estimate `dist` of the distance from s to every other vertex. Initially:

- The estimate of the distance from s to itself is set to 0 (which is correct).
- The estimate is set to ∞ for all other vertices (which is typically an over-estimate).

All vertices are unmarked. Then repeatedly, the algorithm:

1. Finds an unmarked vertex u with the smallest current estimate.
2. Marks this vertex (thereby indicating that this estimate is correct).
3. Updates the estimates for all vertices v reachable by arcs uv as:

   $$\text{dist}(v) \leftarrow \min \text{dist}(v), \text{dist}(u) + w_{uv},$$

   where w_{uv} is the weight of the edge from u to v.

Implementation Details

We keep all the vertices that are not marked and their estimated distances in a priority queue, and extract the minimum in each iteration.

Algorithm : Dijkstra's Algorithm

Input: Digraph $G = (V, E)$ with edge-weights $w_e \geq 0$ and source vertex $s \in G$

Output: The shortest-path distances from s to each vertex

```
add s to heap with key 0
for v ∈ V \ {s} do
    add v to heap with key ∞
while heap not empty do
    u ← deletemin
    for v a neighbor of u do
        key(v) ← min{ key(v), key(u) + w_uv }        // relax uv
```

For GATE DA Course & Test Series , visit: www.piyushwairale.com

9 Adjacency Matrix

Given a graph G with n vertices, the adjacency matrix A is an n
$times n$ matrix where the entry $A[i][j]$ indicates whether there is an edge from vertex i to vertex j.

Types of Graphs and Their Adjacency Matrices

1. Undirected Graphs

- If the graph is undirected, the adjacency matrix is symmetric. This means that $A[i][j] = A[j][i]$.
- For example, if there is an edge between vertices i and j, then both $A[i][j]$ and $A[j][i]$ will be 1 (assuming 1 represents the presence of an edge).

2. Directed Graphs (Digraphs)

- In a directed graph, the adjacency matrix is not necessarily symmetric.
- $A[i][j] = 1$ indicates there is a directed edge from vertex i to vertex j, and $A[j][i]$ may be 0 if there is no edge from j to i.

3. Weighted Graphs

- If the graph is weighted, the entries in the adjacency matrix can be the weights of the edges instead of 1.
- For instance, $A[i][j]$ could be the weight of the edge from i to j, and if there is no edge, $A[i][j]$ could be 0 or another value indicating no connection (often ∞ or some large number).

For GATE DA Course & Test Series , visit: www.piyushwairale.com