Self-Assignment: Graph Theory

2024

1 Introduction

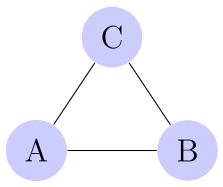
Graph Theory is a fundamental area of mathematics with applications in computer science, network analysis, optimization, and more. This document will cover essential topics including Graphs, Trees, and Networks, along with examples and key algorithms. This document is math-based and won't cover Prim's or Kruskal's algorithms.

2 Graphs

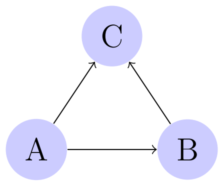
A **graph** G is an ordered pair (V, E), where V is a set of vertices (or nodes) and E is a set of edges (or links) connecting pairs of vertices.

2.1 Types of Graphs

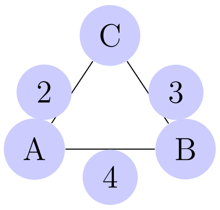
- **Undirected Graph**: In an undirected graph, edges have no direction. The edge (u, v) is identical to (v, u). This means if there is an edge between u and v, the direction does not matter.



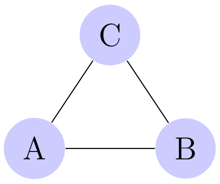
- **Directed Graph (Digraph)**: In a directed graph, edges have a direction. An edge (u, v) indicates a directed connection from u to v and is not the same as (v, u). This can model scenarios where direction matters, like traffic flows.



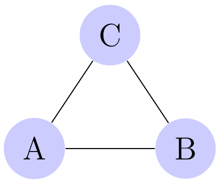
- **Weighted Graph**: In a weighted graph, each edge has a weight or cost associated with it. The weight $w(u, v)$ represents the cost or distance between vertices u and v. Weighted graphs are useful for finding the shortest path or minimum cost.



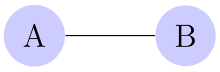
- **Unweighted Graph**: An unweighted graph has edges that do not carry any weight. All edges are considered equal, which simplifies the graph but may not be useful for applications requiring edge weights.



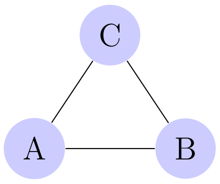
- **Simple Graph**: A simple graph has no loops (edges connecting a vertex to itself) and no multiple edges between the same pair of vertices. It is the most basic form of a graph.



- **Multi graph**: A multi graph can have multiple edges between the same pair of vertices. This allows for modeling scenarios where multiple connections exist between two nodes, such as different types of relationships.



- **Complete Graph**: A complete graph is one in which there is a unique edge connecting every pair of vertices. Each vertex is connected to every other vertex.



2.2 Graph Representations

- **Adjacency Matrix**: A $|V| \times |V|$ matrix where the entry at row i and column j indicates the presence (and weight) of an edge between vertices i and j.
- **Adjacency List**: A list where each vertex has a list of adjacent vertices. This representation is more space-efficient for sparse graphs.
- **Incidence Matrix**: A $|V| \times |E|$ matrix where the entry at row i and column j indicates whether vertex i is incident to edge j.

2.3 Example

Consider the following undirected graph G:

$$G = (V, E)$$

where $V = \{A, B, C, D\}$ and $E = \{(A, B), (B, C), (C, D), (D, A), (A, C)\}$.

The adjacency matrix for this graph is:

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

3 Trees

A **tree** is a connected acyclic graph. Trees are a special type of graph with unique properties.

3.1 Properties of Trees

- A tree with n vertices has exactly $n - 1$ edges.
- Any two vertices are connected by exactly one path.
- Adding an edge between any two vertices of a tree creates exactly one cycle.

3.2 Example

Consider the following tree:

$$T = (V, E)$$

where $V = \{A, B, C, D\}$ and $E = \{(A, B), (A, C), (B, D)\}$.

4 Networks

A **network** is a graph with weights on its edges, often used to model flow problems or optimization problems.

4.1 Example

Consider a network where each edge represents a capacity. The capacities between nodes are given by the following matrix:

$$\begin{bmatrix} - & 1 & 2 \\ 1 & - & 4 \\ 2 & 3 & - \\ - & - & - \end{bmatrix}$$

5 Algorithms

5.1 Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest paths from a source vertex to all other vertices in a weighted graph with non-negative weights.

5.1.1 Algorithm Description

1. Initialize the distance to the source vertex s to 0 and to all other vertices to infinity (∞). 2. Create a priority queue and insert the source vertex with distance 0. 3. While the priority queue is not empty:

- Extract the vertex u with the minimum distance.
- For each adjacent vertex v of u, if the distance to v through u is less than the current known distance, update the distance and insert v into the priority queue.

4. Repeat until all vertices have been processed.

5.1.2 Mathematical Representation

Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges. Each edge $(u, v) \in E$ has a non-negative weight $w(u, v)$. Define $d(u)$ as the shortest distance from the source vertex s to vertex u. Initially, $d(s) = 0$ and $d(v) = \infty$ for all $v \neq s$.

The algorithm iteratively updates $d(u)$ using the relaxation step:

$$d(v) = \min(d(v), d(u) + w(u, v))$$

5.1.3 Example

Consider a graph with vertices $\{A, B, C, D\}$ and edges with weights:

Edge	A	B	C	D
$A \to B$	1	–	–	–
$A \to C$	4	–	–	–
$B \to C$	2	1	–	–
$B \to D$	5	–	–	1
$C \to D$	1	–	–	–

Starting from vertex A, Dijkstra's algorithm will compute the shortest paths to all other vertices.

5.2 Bellman-Ford Algorithm

The Bellman-Ford algorithm finds the shortest paths from a source vertex to all other vertices in a weighted graph, and it can handle negative weights.

5.2.1 Algorithm Description

1. Initialize the distance to the source vertex s to 0 and to all other vertices to infinity (∞). 2. For $|V| - 1$ iterations:

- For each edge (u, v) with weight $w(u, v)$, if $d(u) + w(u, v) < d(v)$, update $d(v) = d(u) + w(u, v)$.

3. Check for negative-weight cycles:

- For each edge (u, v) with weight $w(u, v)$, if $d(u) + w(u, v) < d(v)$, a negative-weight cycle exists.

5.2.2 Mathematical Representation

Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges. Each edge $(u, v) \in E$ has a weight $w(u, v)$. Define $d(u)$ as the shortest distance from the source vertex s to vertex u. Initially, $d(s) = 0$ and $d(v) = \infty$ for all $v \neq s$.

For $|V| - 1$ iterations, update the distances:

$$d(v) = \min(d(v), d(u) + w(u, v))$$

After $|V| - 1$ iterations, if any distance can still be updated, the graph contains a negative-weight cycle.

5.2.3 Example

Consider a graph with vertices $\{A, B, C\}$ and edges with weights:

Edge	A	B	C
$A \to B$	1	$-$	$-$
$B \to C$	2	1	$-$
$C \to A$	-1	$-$	$-$

Starting from vertex A, the Bellman-Ford algorithm will compute the shortest paths and detect any negative-weight cycles if they exist.

6 References

- **None**