Graph

By Mani Abedii

View Full DSA Repository

A clear, compact guide covering essential data structures and algorithms, designed for easy understanding and quick learning.

A **Graph** is a non-linear data structure. A graph G is a pair (V,E), where V is a finite set and E is a binary relation on V. The set V is called the vertex set of G and the set E is called the edge set of G. If E consists of unordered pairs of vertices the graph is **undirected**, otherwise if E consists of ordered pairs of vertices G is **directed**. In practical programming or networking a vertex is usually called a **node.**

If (u→ v) is an edge in a graph, we say that vertex v is **adjacent to** vertex u, we also say that edge (u, v) is **incident on** vertices u and v.

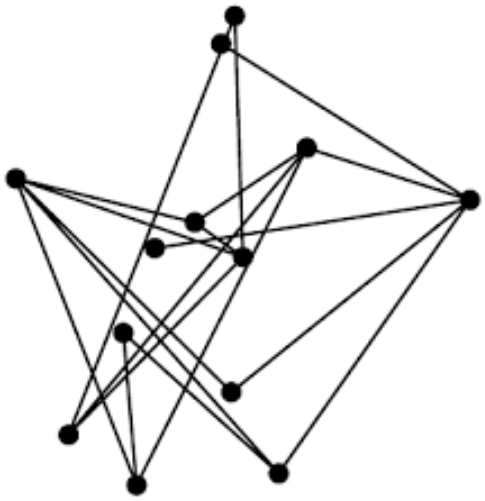
The **degree** of a vertex in an undirected graph is the number of edges incident on it. In a directed graph, the **out-degree** of a vertex is the number of edges leaving it, and the **in-degree** of a vertex is the number of edges entering it. The degree of a vertex in a directed graph is its in-degree plus its out-degree.

A **simple path** from a vertex u to a vertex u' is a distinct sequence of vertices starting with u and ending with u'. The **length** of the path is the number of edges in the path. If there is a path p from u to u', we say that u' is **reachable** from u via p. A path forms a **simple cycle** if u = u' and the path contains at least one edge. A graph with no cycles is called an **acyclic graph**.

An undirected graph is **connected** if every pair of vertices is connected by a path. A directed graph likewise is called **strongly connected** if every two vertices are reachable from each other.

Graph G' = (V', E') is a **subgraph** of G = (V, E) if V' ⊂ V and E' ⊂ E. A **complete graph** is an undirected graph in which every pair of vertices is adjacent.

An acyclic undirected graph is a **forest**, and a connected acyclic undirected graph is a **tree**. Also, **DAG** is **D**irected **A**cyclic **G**raph.

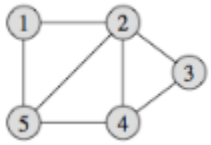


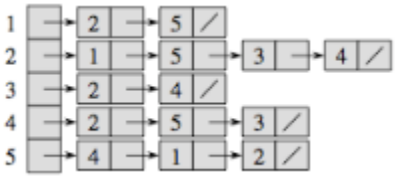
Representations of graphs

1. **Adjacency matrix:** this representation of a graph consists of a |V|×|V| matrix such that:

   element(i,j) = 1 if (i,j) ∈ E, 0 otherwise.

2. **Adjacency list:** this representation of a graph consists of an array of |V| lists, one for each vertex in V. For each u ∈ V, the adjacency list contains all the vertices adjacent to u in G.





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Traversals of graphs

Graph traversal is the process of visiting all the vertices in a graph in a systematic way. The most common graph traversal techniques are **Breadth-First Search (BFS)** and D**epth-First Search (DFS)**.

- **Breadth-First Search (BFS)** explores a graph level by level, starting from a source vertex s. It systematically visits all vertices reachable from s using a queue to track the next vertex to explore. BFS computes the shortest distance (minimum number of edges) from s to each reachable vertex and builds a breadth-first tree rooted at s.

  The **queue** ensures vertices are processed in the correct order: when a vertex is discovered, it is added to the queue, and vertices are dequeued in **FIFO order** to explore their neighbors.

- **Depth-First Search (DFS)** explores a graph by going as deep as possible along each branch before backtracking. It systematically visits all vertices reachable from a source vertex s using a stack (or recursion, which uses the call stack) to keep track of the next vertex to explore. DFS builds a depth-first tree rooted at s and can be used to explore connectivity, paths, and cycles in the graph.

  The stack ensures vertices are processed in the correct order: when a vertex is discovered, it is pushed onto the stack, and vertices are popped in **LIFO order** to continue exploring their neighbors.