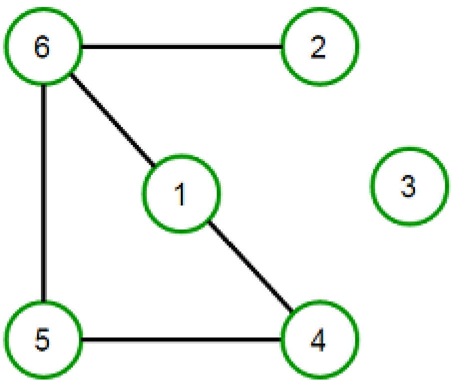
Terminology and Representations of Graphs

This post discusses the basic definitions in terminologies associated with graphs and covers the adjacency list and adjacency matrix representations of the graph data structure.

What is a Graph?

A graph is an ordered pair `G = (V, E)` comprising a set `V` of vertices or nodes and a collection of pairs of vertices from `V`, known as edges of a graph. For example, for the graph below.

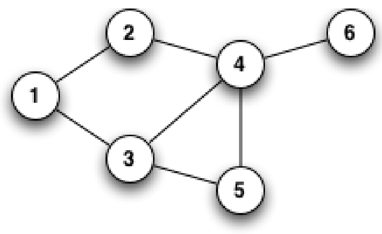
```
V = { 1, 2, 3, 4, 5, 6 }
E = { (1, 4), (1, 6), (2, 6), (4, 5), (5, 6) }
```



Types of Graph

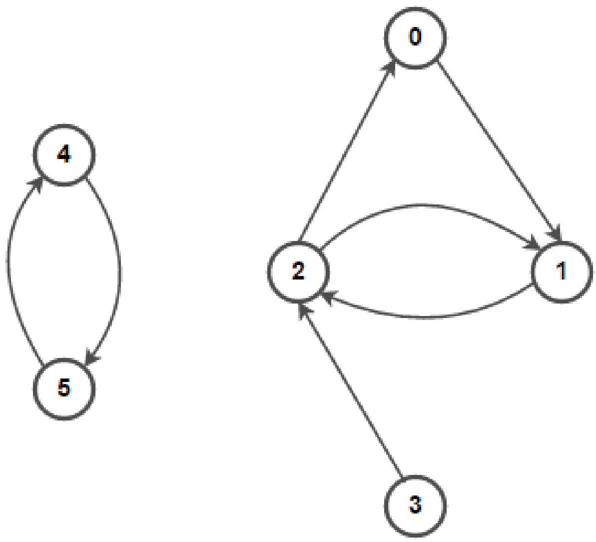
1. Undirected graph

An undirected graph (graph) is a graph in which edges have no orientation. The edge `(x, y)` is identical to edge `(y, x)`, i.e., they are not ordered pairs. The maximum number of edges possible in an undirected graph without a loop is `n×(n-1)/2`.



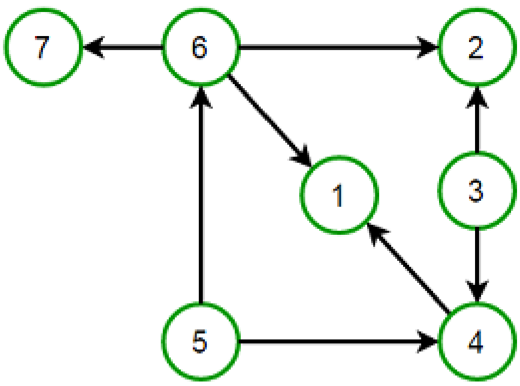
2. Directed graph

A Directed graph (digraph) is a graph in which edges have orientations, i.e., The edge `(x, y)` is not identical to edge `(y, x)`.



3. Directed Acyclic Graph (DAG)

A Directed Acyclic Graph (DAG) is a directed graph that contains no cycles.

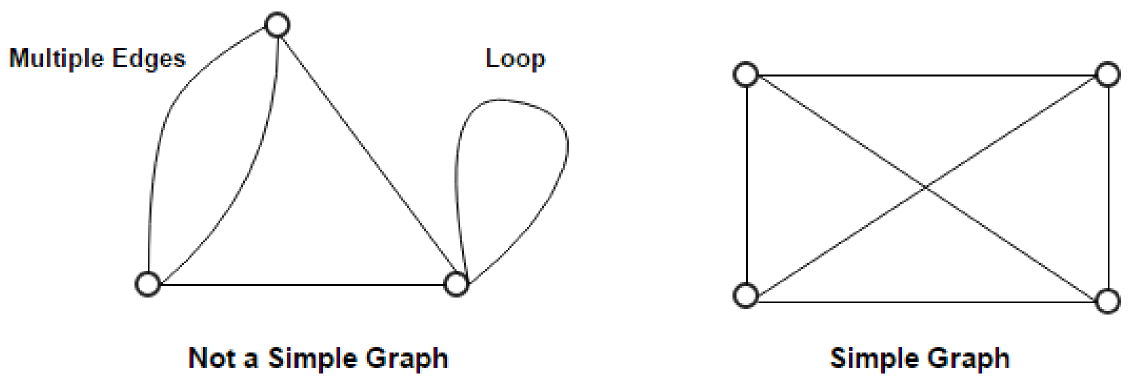


4. Multi graph

A multigraph is an undirected graph in which multiple edges (and sometimes loops) are allowed. Multiple edges are two or more edges that connect the same two vertices. A loop is an edge (directed or undirected) that connects a vertex to itself; it may be permitted or not.

5. Simple graph

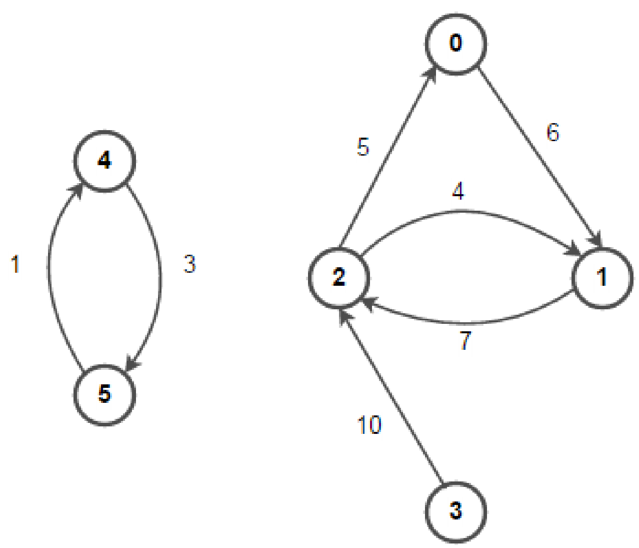
A simple graph is an undirected graph in which both multiple edges and loops are disallowed as opposed to a multigraph. In a simple graph with `n` vertices, every vertex's degree is at most `n-1`.

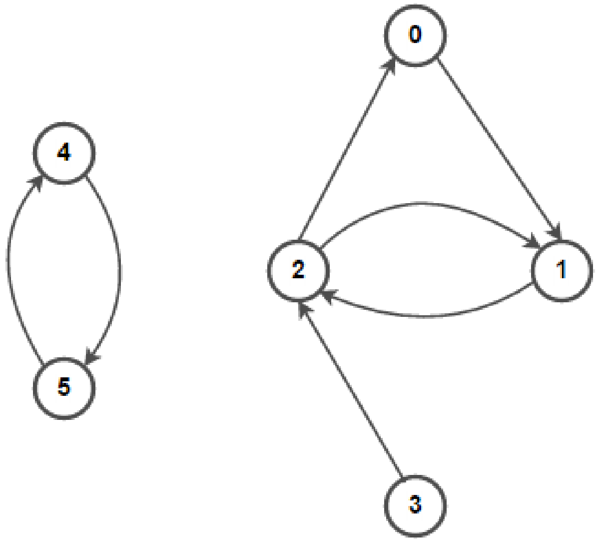


6. Weighted and Unweighted graph

A weighted graph associates a value (weight) with every edge in the graph. We can also use words cost or length instead of weight.

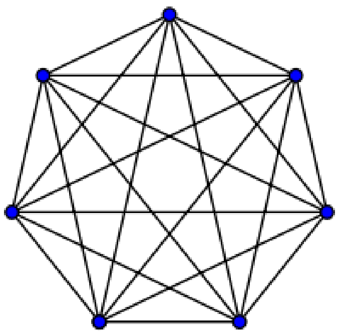
An unweighted graph does not have any value (weight) associated with every edge in the graph. In other words, an unweighted graph is a weighted graph with all edge weight as 1. Unless specified otherwise, all graphs are assumed to be unweighted by default.





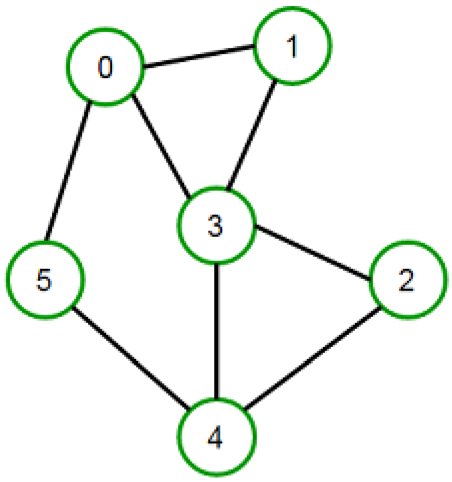
7. Complete graph

A complete graph is one in which every two vertices are adjacent: all edges that could exist are present.



8. Connected graph

A Connected graph has a path between every pair of vertices. In other words, there are no unreachable vertices. A disconnected graph is a graph that is not connected.



Most commonly used terms in Graphs

- An edge is (together with vertices) one of the two basic units out of which graphs are constructed. Each edge has two vertices to which it is attached, called its `endpoints`.
- Two vertices are called `adjacent` if they are endpoints of the same edge.
- `Outgoing edges` of a vertex are directed edges that the vertex is the origin.
- `Incoming edges` of a vertex are directed edges that the vertex is the destination.
- The `degree` of a vertex in a graph is the total number of edges incident to it.
- In a directed graph, the `out-degree` of a vertex is the total number of outgoing edges, and the `in-degree` is the total number of incoming edges.
- A vertex with in-degree zero is called a source vertex, while a vertex with out-degree zero is called a sink vertex.
- An isolated vertex is a vertex with degree zero, which is not an endpoint of an edge.
- `Path` is a sequence of alternating vertices and edges such that the edge connects each successive vertex.
- `Cycle` is a path that starts and ends at the same vertex.
- `Simple path` is a path with distinct vertices.
- A graph is `Strongly Connected` if it contains a directed path from `u` to `v` and a directed path from `v` to `u` for every pair of vertices `u`, `v`.
- A directed graph is called `Weakly Connected` if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. The vertices in a weakly connected graph have either out-degree or in-degree of at least 1.
- `Connected component` is the maximal connected subgraph of an unconnected graph.
- A `bridge` is an edge whose removal would disconnect the graph.
- `Forest` is a graph without cycles.
- `Tree` is a connected graph with no cycles. If we remove all the cycles from DAG (Directed Acyclic Graph), it becomes a tree, and if we remove any edge in a tree, it becomes a forest.
- `Spanning tree` of an undirected graph is a subgraph that is a tree that includes all the vertices of the graph.

Relationship between number of edges and vertices

For a simple graph with `m` edges and `n` vertices, if the graph is

- directed, then `m = n×(n-1)`
- undirected, then `m = n×(n-1)/2`
- connected, then `m = n-1`
- a tree, then `m = n-1`
- a forest, then `m = n-1`
- complete, then `m = n×(n-1)/2`

Therefore, $O(m)$ may vary between $O(1)$ and $O(n^2)$, depending on how dense the graph is.

Graph Representation

1. Adjacency Matrix Representation:

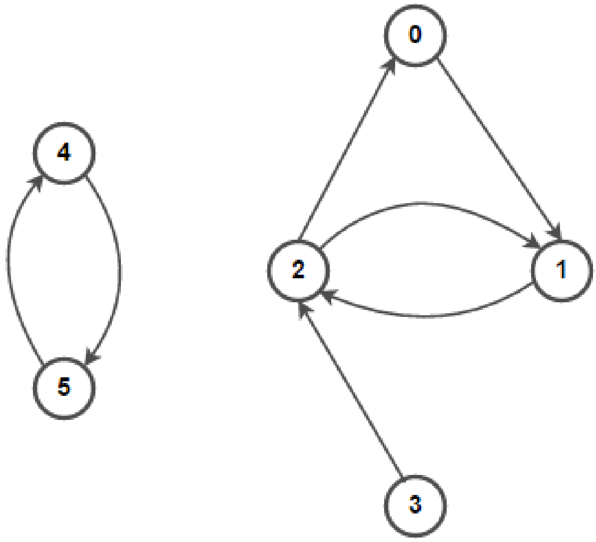
An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

Definition:

For a simple unweighted graph with vertex set `V`, the adjacency matrix is a square `|V| × |V|` matrix `A` such that its element:

$A_{ij} = 1$, when there is an edge from vertex `i` to vertex `j`, and
$A_{ij} = 0$, when there is no edge.

Each row in the matrix represents source vertices, and each column represents destination vertices. The diagonal elements of the matrix are all zero since edges from a vertex to itself, i.e., loops are not allowed in simple graphs. If the graph is undirected, the adjacency matrix will be symmetric. Also, for a weighted graph, A_{ij} can represent edge weights.

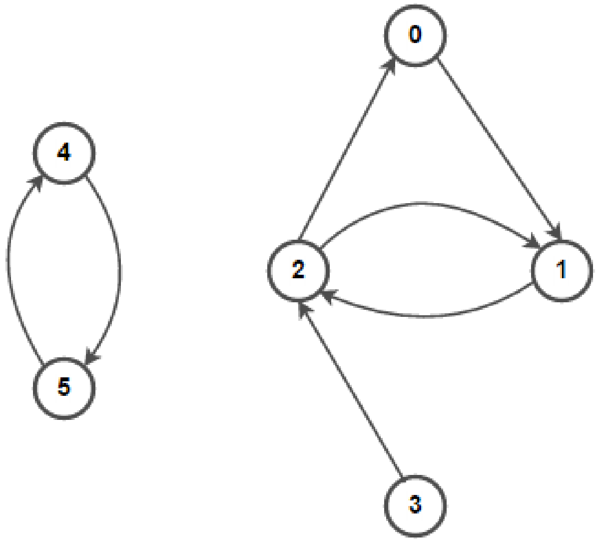


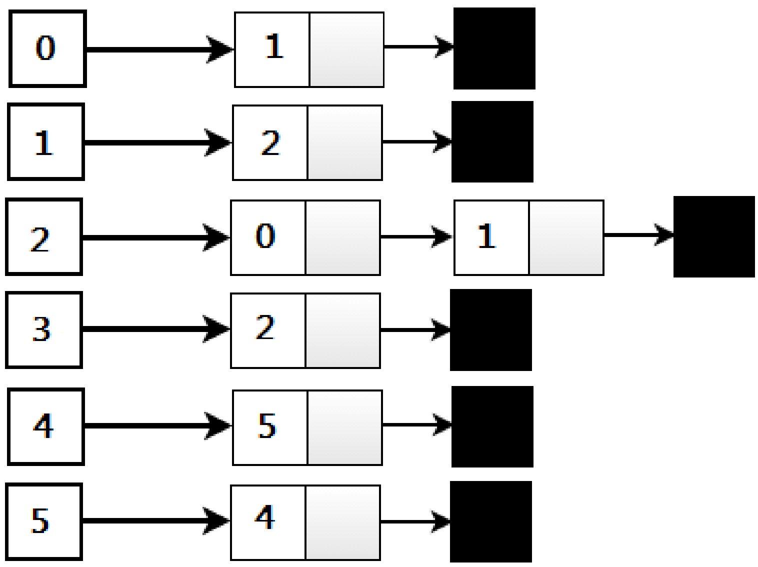
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

An adjacency matrix keeps a value `(1/0/edge-weight)` for every pair of vertices, whether the edge exists or not, so it requires `n2` space. They can be efficiently used only when the graph is dense.

2. Adjacency List Representation:

An adjacency list representation for the graph associates each vertex in the graph with the collection of its neighboring vertices or edges, i.e., every vertex stores a list of adjacent vertices. There are many variations of adjacency list representation depending upon the implementation. This data structure allows the storage of additional data on the vertices but is practically very efficient when the graph contains only a few edges. i.e. the graph is sparse.





Also See:

Implement Graph Data Structure in C

Graph Implementation in C++ (without using STL)

Graph Implementation in C++ using STL

Graph Implementation in Java using Collections

References:

1. **http://www.csl.mtu.edu/cs2321/www/newLectures/24_Graph_Terminology.html**

2. **https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)**

Graph

Beginner, Must Know