What is a Graph?

A graph $G$ consists of a set of nodes (also called vertices) $V$, and a set of edges $E$, each edge connecting a pair of nodes in $V$. Numerous real-life problems can be represented using graphs.

**The Seven Bridges of Königsberg**

Is it possible to walk along a route that crosses each bridge exactly once?
Problem proposed by Euler

Such a path is known as Euler path
Suppose there is such a path.

Then there is a starting point and there is an end point.
For every intermediate edge v, there must be an equal number of incoming and outgoing edges.

Is it true for the bridge graph?
Friendship in Social network

Registrar’s graph for Exam Scheduling

Each node is a course, and an edge between nodes A and B denote that some student is taking both A and B.
Classification of Graphs

Undirected vs. Directed

A multi-graph allows multiple edges between any given pair of nodes

Simple graph vs. multi-graph:

A multi-graph allows multiple edges between any given pair of nodes
Weighted graph
**Graph ADT**

*Partial list of methods*

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices(), NumberVertices()</td>
<td></td>
</tr>
<tr>
<td>Edges(), NumEdges(), Degree()</td>
<td></td>
</tr>
<tr>
<td>Outdegree(v), Indegree(v)</td>
<td>for directed graphs</td>
</tr>
<tr>
<td>OutgoingEdges(), IncomingEdges()</td>
<td></td>
</tr>
<tr>
<td>InsertVertex(x), InsertEdge(u,v,x)</td>
<td></td>
</tr>
<tr>
<td>RemoveVertex(x), RemoveEdge(e)</td>
<td></td>
</tr>
</tbody>
</table>

How efficiently the graph can be stored and these methods can be computed, will depend on the data structure used to represent the graph.
Data Structure for graphs

*Edge list* with *n nodes, m edges*

Store each list and a doubly linked list.
Takes up $O(n+m)$ space.
Takes up $O(n)$ time to report the number of vertices and
$O(m)$ time to report the number of edges
### Adjacency Matrix

A simple graph can be represented by an adjacency matrix. Consider the following graph:

![Graph Diagram](image)

The adjacency matrix for this graph is:

\[
\begin{array}{ccccc}
A & B & C & D & E \\
\hline
A & 0 & 1 & 0 & 1 & 0 \\
B & 1 & 0 & 1 & 0 & 0 \\
C & 0 & 1 & 0 & 1 & 1 \\
D & 1 & 0 & 1 & 0 & 1 \\
E & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

1 = true, 0 = false

Needs $O(n^2)$ space to store the graph.
Adjacency List

Each node maintains a linked list of its neighbors.
Graph traversal

Breadth-First-Search

Breadth-First-Search (root):

    // Initialization
    for each node n in Graph {
        n.distance = INFINITY; n.parent = NIL;
        create empty queue Q;
        root.distance = 0;
        Q.enqueue(root)
    }

    while Q is not empty{
        current = Q.dequeue();
        for each node n adjacent to current{
            if n.distance == INFINITY {
                n.distance = current.distance + 1;
                n.parent = current;
                Q.enqueue(n)
            }
        }
    }