What a Graph Is

A graph is a set of points (called **vertices** or **nodes**), of which some may be connected by lines (called **edges**).

In some graphs, these edges go only one way (e.g. vertex one may be connected by an edge to vertex two, but vertex two may not be connected to vertex one) - any graph like this is called a **directed graph**. When all edges go both ways (if vertex one is connected to vertex two, vertex two is connected to vertex one), the graph is called an **undirected graph**.

Edges can have lengths (called **weights**). If no weights are specified, then it should be taken to mean that every weight is exactly 1. A **path** between vertices *a* and *b* is a sequence of edges that leads from *a* to *b*. The **length of the path** is the sum of the edge weights on every edge in the sequence.

A path is a **cycle** if the start and end vertices are the same. A path is **simple path** if every vertex that appears on the path, appears at most once.

An undirected graph is **connected** if there exists a path between any given pair of vertices. The connected parts of a graph are called **components**.

A **tree** is a connected graph of *N* vertices that contains exactly *N-1* edges. There are 0 cycles in a tree, and every simple path between any given pair of vertices in a tree is unique.

A graph is a **simple graph** if no edge starts and ends at the same vertex.

Storing Graphs

Graphs are generally stored one of two ways:

2D array (adjacency matrix)

Declare an array *G[N][N]* where *N* is the number of vertices; *G[a][b]* will represent the weight of the edge connecting *a* to *b* (if there is no edge connecting *a* to *b*, or *a* == *b*, the array entry will be 0).

Primitive array of vectors

Declare an array of vectors `vector<pair<int,int>> G[N]` where `N` is the number of vertices. `G[i]` will contain all the edges emanating from vertex `i`, each one stored as a pair containing both the vertex it connects to and the edge weight.

The adjacency matrix is better for when individual edges should be queried; it takes a guaranteed one operation to check the edge between any two vertices, while in an array of vectors up to each edge connecting to one of the two vertices must be checked.

When operations involve interactions with each edge connecting to a given vertex (e.g. shortest path problems), then the primitive array of vectors is more useful; attempting the same with an adjacency matrix causes operations to be wasted as the program checks edges that may not exist.