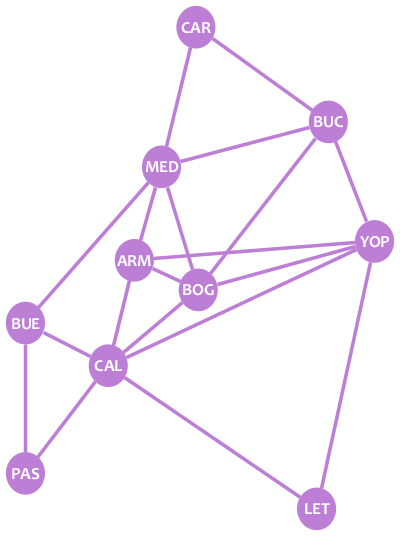
**How graphs are represented**

You now hopefully know what graphs are. Now, let's see how we can represent graphs in ways other than drawing. This will be useful if we want to work with graphs with a computer.

Adjacency List

The image below represents some Colombian cities: Cartagena, Bucaramanga, Medellín, Bogotá, among others. Each circle is a city. The lines connecting the circles represent commercial flights that you can take between cities. For example, you can fly from Bucaramanga (BUC) to Bogotá (BOG), as those cities are connected with a line, but you can't fly from Pasto (PAS) to Leticia (LET) because they aren't connected.



Easy, right? Well, that's a graph, and they are indeed easy to understand. **Graphs** are structures that allow us to model relationships between elements. In this case, we used a graph to model flight connections between cities. Graphs are composed of two kinds of elements:

1. **Vertices or nodes**, which represent elements. In the image above, the circles are the nodes, each node representing a city.
2. **Edges**, which represent relationships between elements. In the image, the lines are the edges, each edge representing a flight connection between cities.

Each edge connects exactly two nodes. This means that you'll never find an edge with one side pointing to nowhere.

We say that two nodes are neighbors if there is an edge connecting them. In the example, Bucaramanga (BUC) and Bogotá (BOG) are neighbors, but Pasto (PAS) and Leticia (LET) are not neighbors.

Let's make a list of each node and that node's neighbors:

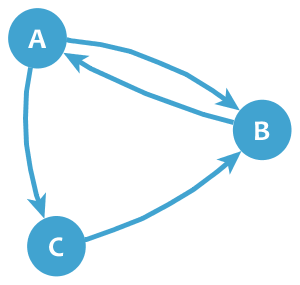
* **CAR:** MED, BUC
* **MED:** CAR, BUC, BUE, ARM, BOG
* **BUC:** CAR, MED, BOG, YOP
* **BUE:** MED, CAL, PAS
* **ARM:** MED, BOG, CAL
* **BOG:** MED, BUC, YOP, ARM, CAL
* **YOP:** BUC, ARM, BOG, LET
* **CAL:** BUE, ARM, BOG, YOP, LET, PAS
* **PAS:** BUE, CAL
* **LET:** CAL, YOP

That's the ***adjacency list*** of the graph: a list of lists describing the nodes and the neighbors of each node. If the graph is *directed*, a node B only appears in the list of a node A if there's an edge from A to B.

Let's suppose we want to model some Twitter users with a graph. This is the information about the users:

* Alice (A) follows Bob and Carol.
* Bob (B) follows Alice.
* Carol (C) follows Bob.

Our directed Twitter graph would look like this:



The adjacency list would look like this:

* **A:** B, C
* **B:** A
* **C:** B

Adjacency Matrix

Graphs can also be represented with **adjacency matrices**. Here's the adjacency matrix of our cities graph:

|  | **CAR** | **BUC** | **YOP** | **BOG** | **LET** | **CAL** | **ARM** | **MED** | **BUE** | **PAS** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CAR** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **BUC** | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **YOP** | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **BOG** | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| **LET** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **CAL** | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| **ARM** | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **MED** | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| **BUE** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| **PAS** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Adjacency matrices have the graph nodes in both their rows and columns. The value on each cell shows if there exists an edge between the pair of nodes of the corresponding row and column. For example, the cell in row **BOG** and column **BUC** has a value of **1** because there is an edge between BOG and BUC, whereas the cell in row **PAS** and column **YOP** has a **0** because there's no edge between those two nodes.

A graph may have several adjacency matrices. In the example above, if you put the nodes in a different order, you'll have a new adjacency matrix of the same graph.

If the graph was *weighted* (there was a cost associated with the edge, e.g. the length of the trip), you could put the weight in the adjacency matrix (instead of just 0 or 1).

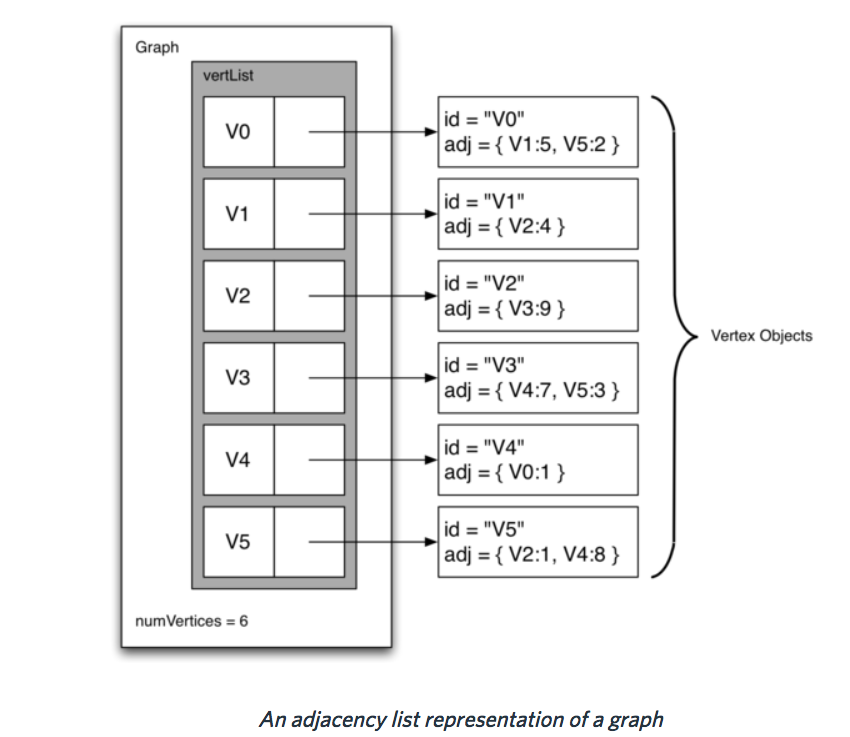
As you can see, if a graph is undirected, its adjacency matrices are symmetrical. On the other hand, if the graph is directed, the matrices are not symmetrical, as a 1 only appears in a cell if there's an edge from the node of the row to the node of the column. For example, the following is the adjacency matrix of our Twitter graph:

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **A** | 0 | 1 | 1 |
| **B** | 1 | 0 | 0 |
| **C** | 0 | 1 | 0 |

Adjacency Map

An adjacency list is actually a bit of a misnomer; a list of linked lists is really more of a table. That said, sometimes it is convenient to maintain maps, rather than lists, for quicker / easier lookup. An adjacency map is essentially an adjacency list, with a map instead of a list.

For example, you could have a Graph class which maintains a master map of all the vertices in the graph. Each vertex would be mapped to a list of its edges. In the case of a weighted graph (a graph where there is a cost associated with moving from one node to another), the weight of the connection could *also* be stored in a map, like this:



*Borrowed with minor modifications from:*

[*https://tech.io/playgrounds/5470/graph-theory-basics/representations*](https://tech.io/playgrounds/5470/graph-theory-basics/representations)