Adjacency Matrix

One of the easiest ways to implement a graph is to use a two-dimensional matrix. In this matrix implementation, each of the rows and columns represent a vertex in the graph. The value that is stored in the cell at the intersection of row v and column w indicates if there is an edge from vertex v to vertex w. When two vertices are connected by an edge, we say that they are **adjacent**. The example below illustrates the adjacency matrix for the graph in **the previous section (https://maryville.instructure.com/courses/43640/pages/an-introduction-to-graphs)** . A value in a cell represents the weight of the edge from vertex v to vertex w.

	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

The advantage of the adjacency matrix is that it is simple, and for small graphs it is easy to see which nodes are connected to other nodes. However, notice that most of the cells in the matrix are empty. Because most of the cells are empty we say that this matrix is "sparse." A matrix is not a very efficient way to store sparse data. In fact, in Python you must go out of your way to even create a matrix structure like the one in example.

The adjacency matrix is a good implementation for a graph when the number of edges is large. But what do we mean by large? How many edges would be needed to fill the matrix? Since there is one row and one column for every vertex in the graph, the number of edges required to fill the

matrix is $|V|^2$. A matrix is full when every vertex is connected to every other vertex. There are few real problems that approach this sort of connectivity.

Source: **Problem Solving and Algorithms in Python**   **_(http://interactivepython.org/runestone/static/pythonds/index.html#)_** from Bradley Miller on **www.interactivepython.org**   **_(http://interactivepython.org/runestone/static/pythonds/index.html#)_** .