**Graphs: Introduction**

Graphs are the most generalized data structure in this course. Many seemingly unrelated real-world problems can be expressed as graphs, and therefore solved.

Graphs are generalized trees; alternatively, trees are graphs with restrictions on how the nodes are connected. Trees are well-understood but there are graph theorems still waiting to be discovered.



In mathematics and computer science, a *graph* is a set of *vertices* (or nodes) which are connected by links (or *edges*). The graphs may be *directed*, meaning the edges go in one direction, as shown by the arrows above. Otherwise, if the edges connect in both directions, drawn without arrows, the graph is *undirected*. One of the fundamental questions about graphs is the question of *reachability*. For example:

1. If you start at 0, is 1 reachable? Y/N
2. If you start at 0, is 2 reachable? Y/N
3. If you start at 2, is 0 reachable? Y/N

More vocabulary:

* *Loop*: a loop is an edge that connects a vertex to itself. In the diagram above, 2 has a loop.
* *Path*: a path is a sequence of vertices connected by edges. In a directed graph, the path goes from the *source* vertex to the *target* vertex.
* *Cycle*: a cycle in a directed graph is a path that begins and ends at the same vertex. The *length* of the cycle is the number of edges.
* *Neighbors*: the neighbors of a source vertex are all the vertices reachable by traveling along one (1) edge.

More questions:

1. How many vertices are in the graph above?
2. How many edges are in the graph above?
3. How many loops are in the graph above?
4. Write the path for the cycle in the graph above.
5. How long is that cycle?
6. Is there a path from source 2 to target 0?
7. List the neighbors of 0.
8. List the neighbors of 3.

**Graphs 0: AdjMat**

Two common representations of graphs are an *adjacency matrix* and an *adjacency list* (coming later). Compare the diagram on the left with its adjacency matrix representation on the right, in which a 0 indicates no edge and 1 indicates an edge. The diagram and the matrix contain the same information, but the computer can process the adjacency matrix.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0** | 0 | 0 | 1 | 0 |
| **1** | 1 | 0 | 0 | 0 |
| **2** | 0 | 0 | 1 | 1 |
| **3** | 1 | 0 | 1 | 0 |



An alternative implementation has the adjacency matrix storing booleans instead of 0’s and 1’s. In our case, using 0 and 1 generalizes nicely when we reach *weighted* graphs (coming later).

Looking only at the adjacency matrix, answer the same questions as before:

1. If you start at 0, is 1 reachable? Y/N
2. If you start at 0, is 2 reachable? Y/N
3. If you start at 2, is 0 reachable? Y/N
4. How many vertices are in the graph above?
5. How many edges are in the graph above?
6. How many loops are in the graph above?
7. Write the path for the cycle in the graph above.
8. How long is that cycle?
9. Is there a path from source 2 to target 0?

Now we need to write a class that implements the *adjacency matrix representation of a graph*. We will call it AdjMat. We will start simply and add to AdjMat in later labs.

1. What data structure will hold the 0's and 1's in the grid? Declare it here:
2. Do we need a constructor? Do we need a constructor with arguments?
3. There is no file to read. We will hard-code the edges. In that case, how do we add edges?
4. How do we remove edges?

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 |

1. We will need to display the matrix. Let's return a simple toString of the grid, showing only the 0's and 1's in the grid, without labeling rows and columns.
2. Let’s do isEdge and edgeCount. How?
3. Let's do List<Integer> getNeighbors(int source). How?

**Assignment**: In AdjMat, implement the AdjacencyMatrix interface. Test AdjMat using AdjMat\_0\_Driver, which has been written for you. You will submit AdjMat.