CMSC 351: Graphs

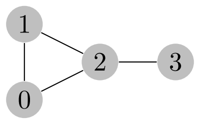
Justin Wyss-Gallifent

April 7, 2024

1 Introduction

Graphs are essentially diagrams (or data, at an abstract level) which represent networks and in general connectivity between objects.

For example the following diagram could represent the connectivity between four computers:



Here we see that computers 0,1,2 are connected to one another and computer 3 is only connected to computer 2.

2 Summary of Definitions

This is just here to help organize. We define:

1. Graph, edge, vertex (node).
2. Adjacent edges.
3. Degree.
4. Loop.
5. Multiple edge(s).
6. Simple.
7. Weighted and unweighted.
8. Directed and undirected.
9. Walk, trail, path.
10. Connected graph.
11. Cycle.
12. Tree.
13. Adjacency matrix for an unweighted undirected simple graph.
14. Adjacency matrix for a weighted directed simple graph.
15. Adjacency list for an unweighted undirected simple graph.
16. Degree matrix for an unweighted undirected simple graph.
17. Laplacian matrix for an unweighted undirected simple graph.
18. These final five definitions are not comprehensive nor completely uniform.

3 Definitions

We have the following formal definitions:

Definition 3.0.1. A *graph* consists of *vertices* or *nodes* connected by *edges*.

Example 3.1. The diagram above is a graph consisting of four vertices and four edges.

Definition 3.0.2. Two vertices are *adjacent* if they are connected by an edge.

Definition 3.0.3. The *degree* of a vertex is the number of edge connections incident at that vertex.

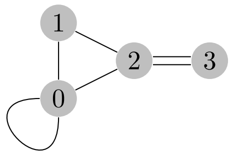
Example 3.2. In the above graph vertices 0,1 have degree 2, vertex 2 has degree 3 and vertex 3 has degree 1.

Note 3.0.1. If the graph has loops and multiple edges (see below) then these must be accounted for.

Definition 3.0.4. A *loop* is an edge which joins a vertex to itself.

Definition 3.0.5. Two vertices may be joined by more than one edge, in which case we say there are *multiple edges* between the vertices.

Example 3.3. This graph has a loop (vertex 0 to itself) and multiple edges (vertex 2 to vertex 3):



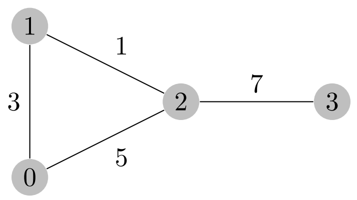
Definition 3.0.6. A graph is *simple* if it has no loops and no multiple edges.

Example 3.4. The first graph is simple.

Definition 3.0.7. A graph is *weighted* if each edge has a numerical weight associated to it.

Most, if not all, of the graphs we will look at will be weighted simple graphs where each weight yields some sort of cost of connection. A simple example of a road network between cities in which each weight is the distance.

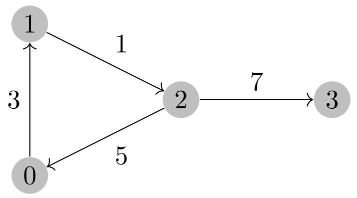
Example 3.5. Here is a weighted simple graph:



Definition 3.0.8. A graph is *directed* if the edges have directions, usually indicated by arrows.

Working with directed graphs is much more challenging than working with undirected graphs.

Example 3.6. Here is a weighted simple directed graph:



4 Counting Notation

It's common to use V for the number of vertices and E for the number of edges. Sometimes V and E are also used for the sets of vertices and edges but I'll try to avoid that here.

5 Walks, Trails, Paths, Cycles, Trees

Definition 5.0.1. A *walk* of length k from vertex u to vertex v is a sequence of k edges which start at vertex u and end at vertex v.

Intuitively a walk is simply a way of getting from vertex u to vertex v. It's permissible to repeat edges and vertices.

Definition 5.0.2. A *trail* is a walk in which all edges are distinct. Note that vertices may be repeated.

Note 5.0.1. Walks and trails of length 0 are permitted. Specifically there is a walk/trail of length 0 from each vertex to itself.

Definition 5.0.3. A *path* is a walk in which all vertices are distinct. Note that if vertices are distinct then so are edges.

Note 5.0.2. We can't have a path of length 0 because we can't repeat vertices.

We will primarily focus on paths because these make the most sense when we're discussing optimization. For example if you wanted the shortest distance between two cities you would certainly not repeat any of the intermediate cities or roads!

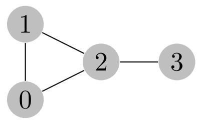
Definition 5.0.4. A *cycle* is a trail in which all vertices are distinct except we insist that the starting and ending vertices are the same.

Note 5.0.3. In a graph in which loops are allowed a loop is a cycle consisting of the point and the edge.

Definition 5.0.5. A graph is *connected* if for any two vertices there is at least one path joining those vertices.

Definition 5.0.6. Formally a *tree* is an undirected simple connected graph with the property that every two vertices are connected by exatly one path.

Example 5.1. In our original graph:



Here are some observations:

- Formally we have $VS = \{0, 1, 2, 3\}$ and $ES = \{(0, 1), (0, 2), (1, 2), (2, 3)\}$.
- The sequence $\langle 3, 2, 0, 2, 1\rangle$ is a walk of length 4 from vertex 3 to vertex 1. It is not a trail since the edge $(0, 2)$ is repeated and so automatically it is not a path either.
- The sequence $\langle 2, 0, 1, 2, 3\rangle$ is a walk of length 4 from vertex 2 to vertex 3. It is also a trail since no edges are repeated. It is not a path though since vertex 2 is reapeated.
- The sequence $\langle 3, 2, 1, 0\rangle$ is a walk of length 3 from vertex 3 to vertex 0. It is also a trail and a path.
- This graph is connected.

6 Adjacency Matrices

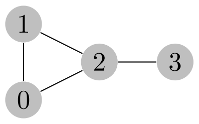
A graph can be represented by an adjacency matrix. For a graph G with V vertices the adjacency matrix A will be $V \times V$ as explained below for various types of graphs.

Note 6.0.1. When we work with matrices in mathematics it's traditional to index from 1 upwards but in computer science it's traditional to index from 0 upwards. I'm going to stick with the computer science approach here so when we discuss an $m \times n$ matrix we really mean an $m \times n$ array A indexed as $AM[i][j]$ with $0 \leq i \leq m - 1$ and $0 \leq j \leq n - 1$. This may seem a little weird from a mathematics perspective but it makes the coding significantly easier.

1. If the graph is simple, unweighted and undirected then:

$$am_{ij} = \begin{cases} 1 & \text{if vertices } i \text{ and } j \text{ are connected by an edge} \\ 0 & \text{if not} \end{cases}$$

Example 6.1. The graph from the beginning:



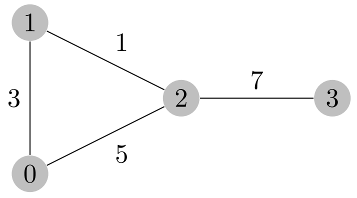
has adjacency matrix:

$$AM = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

2. If the graph is simple and weighted, but not directed, then:

$$am_{ij} = \begin{cases} w & \text{if vertices } i \text{ and } j \text{ are connected by an edge with weight } w \\ 0 & \text{if not} \end{cases}$$

Example 6.2. The following graph:



has adjacency matrix:

$$AM = \begin{bmatrix} 0 & 3 & 5 & 0 \\ 3 & 0 & 1 & 0 \\ 5 & 1 & 0 & 7 \\ 0 & 0 & 7 & 0 \end{bmatrix}$$

Note 6.0.2. This is not comprehensive.

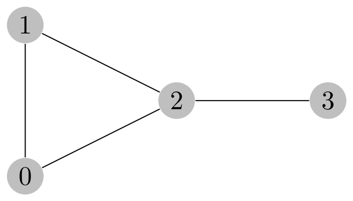
7 Adjacency Lists

While using the adjacency matrix AM is convenient it is not always the best option. Another option is to store an adjacency list.

Definition 7.0.1. The *adjacency list* of a graph G is the list AL such that $AL[i]$ contains a list of vertices adjacent to i.

Note 7.0.1. An adjacency list is a list of lists.

Example 7.1. The following graph:



has adjacency list:

$$AL = [[1,2],[0,2],[0,1,3],[2]]$$

Note 7.0.2. We can generalize the notion of an adjacency list to a weighted graph by storing $(j,w) \in AL[i]$ if there is an edge from i to j with weight w.

8 Considerations

8.1 Storage

Observe the following:

- The adjacency matrix AM will always require V^2 values.
- The adjacency list AL will vary in accordance with the number of edges.

8.2 Access

Observe the following. If we are focusing on a particular vertex x:

- The adjency matrix AM will require a scan of V iterations (either row x or column x) to figure out which other vertices x is connected to.
- The adjacency list AL allows us to immediately access a list of the other vertices x is connected to.

8.3 Math

Observe the following. If we are focusing on a particular vertex x:

- The adjacency matrix AM is a convenient structure which we can work with mathematically.
- The adjacency list isn't.

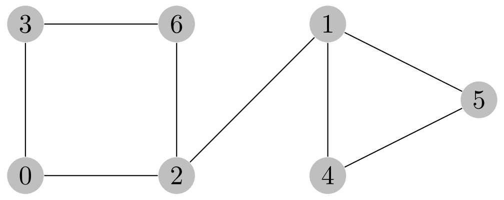
9 The Degree and Other Matrices for a Graph

Definition 9.0.1. For a graph G with n vertices the degree matrix for G is the $V \times V$ diagonal matrix D such that d_{ii} equals the degree of vertex i.

There are many other matrices associated to a graph. One of the most useful is the Laplacian Matrix $L = D - AM$ which is used to study the connectivity of a graph.

10 Thoughts, Problems, Ideas

1. Provide the adjacency matrix and adjacency list for the graph shown here:



2. Suppose a graph G has adjacency list:

$$AL = [[7], [2, 4], [1, 4], [6, 7], [1, 2, 6, 7], [], [3, 4], [0, 3, 5]]$$

Draw a picture of G and write down its adjacency matrix.

3. For a graph G with n vertices write the pseudocode for an algorithm which takes the adjacency matrix AM for G and produces the corresponding adjacency list AL. What is the $\Theta(n)$ time complexity of this? Assume it takes $\Theta(1)$ time to allocate any fixed size list and array full of zeros.

4. For a graph G with n vertices write the pseudocode for an algorithm which will takes the adjacency list AL for G and produce the corresponding adjacency matrix AM. What is the Θ time complexity of this? Assume it takes $\Theta(1)$ time to allocate any fixed size list and array full of zeros.

5. Given the adjacency matrix AM for a graph with n vertices and a list V of k vertices, write the pseudocode for an algorithm which would determine whether V specifies a walk, returning either `TRUE` or `FALSE`. What is the best- and worst-case Θ time complexity for this?

6. Given the adjacency matrix AM for a graph with n vertices and a list V of k vertices, write the pseudocode for an algorithm which would determine whether V specifies a trail, returning either `TRUE` or `FALSE`. What is the best- and worst-case Θ time complexity for this?

7. Given the adjacency matrix AM for a graph with n vertices and a list V of k vertices, write the pseudocode for an algorithm which would determine whether V specifies a path, returning either `TRUE` or `FALSE`. What is the best- and worst-case Θ time complexity for this?

8. Given the adjacency list AL for a graph with n vertices and a list V of k vertices, write the pseudocode for an algorithm which would determine whether V specifies a walk, returning either `TRUE` or `FALSE`. What is the best- and worst-case Θ time complexity for this?

9. Given the adjacency list AL for a graph with n vertices and a list V of k vertices, write the pseudocode for an algorithm which would determine whether V specifies a trail, returning either `TRUE` or `FALSE`. What is the best- and worst-case Θ time complexity for this?

10. Given the adjacency list AL for a graph with n vertices and a list V of

k vertices, write the pseudocode for an algorithm which would determine whether V specifies a path, returning either `TRUE` or `FALSE`. What is the best- and worst-case Θ time complexity for this?