Introduction to Graphs

Step 1

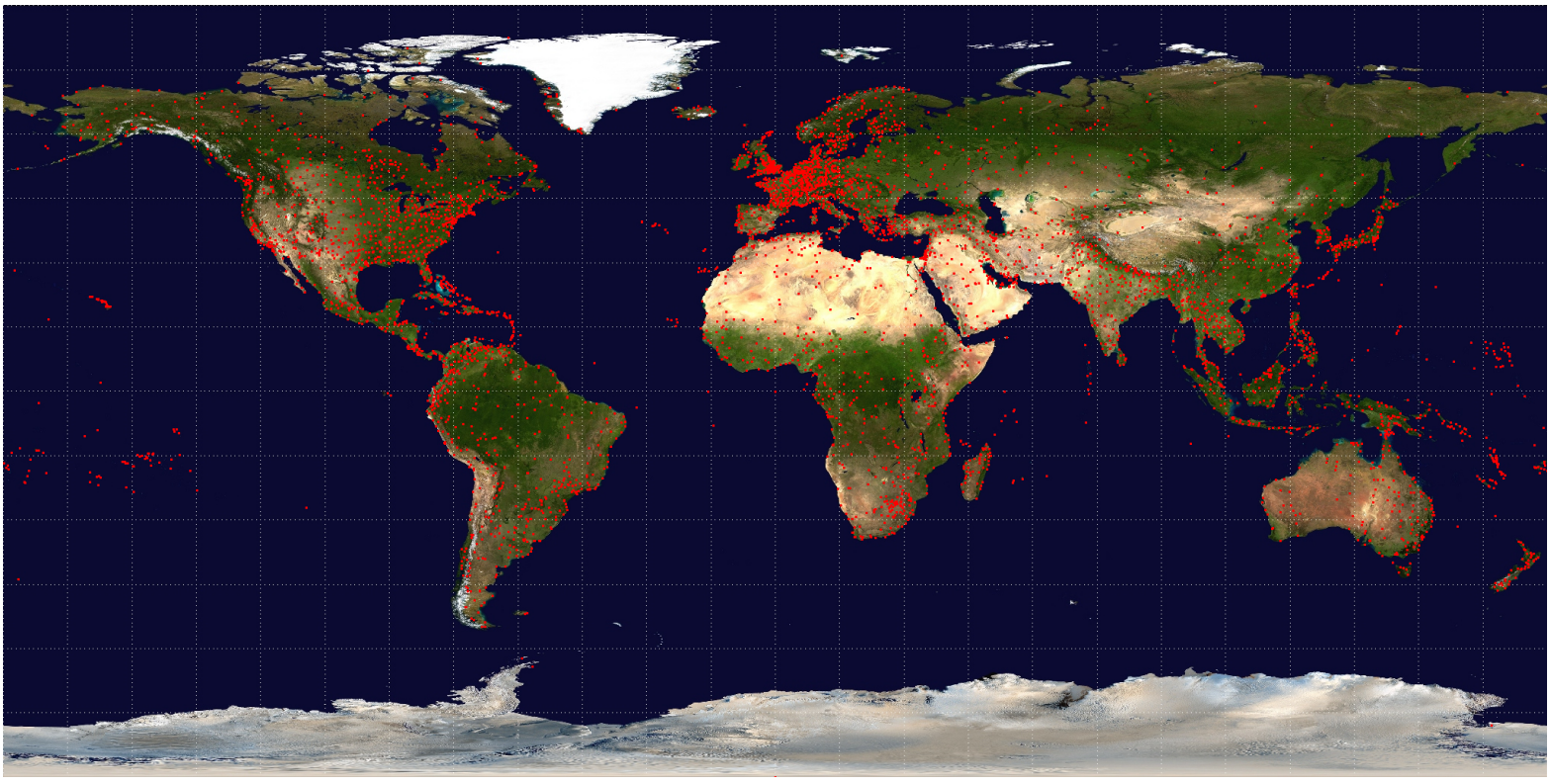
Long ago, before computers were staple items in all households, people had to use physical maps to navigate the streets of the world. As computers became more and more commonplace, services like MapQuest and later Google Maps were developed, which allowed individuals to obtain and print out turn-by-turn directions to reach their destinations. However, the moment you veered off the exact path laid out for you, it was often difficult to figure out how to get back on track. Technology advanced even further, leading to the development of portable GPS devices, which allowed for turn-by-turn directions that could be generated in real-time, and if you veered off the initial path, they would be able to generate updated directions for you on the fly. Now, we've reached a point where even portable GPS devices are obsolete, as most people are able to navigate themselves using their cell phones.

Clearly, navigation systems are essential for transportation, but how do these systems actually work under the hood? You enter a starting location and a destination, and you are magically given turn-by-turn directions for the shortest path connecting these two locations; so what's really going on behind the scenes? In this chapter, we will discuss **Graphs**, the data structures that allow for our beloved navigation systems to function.

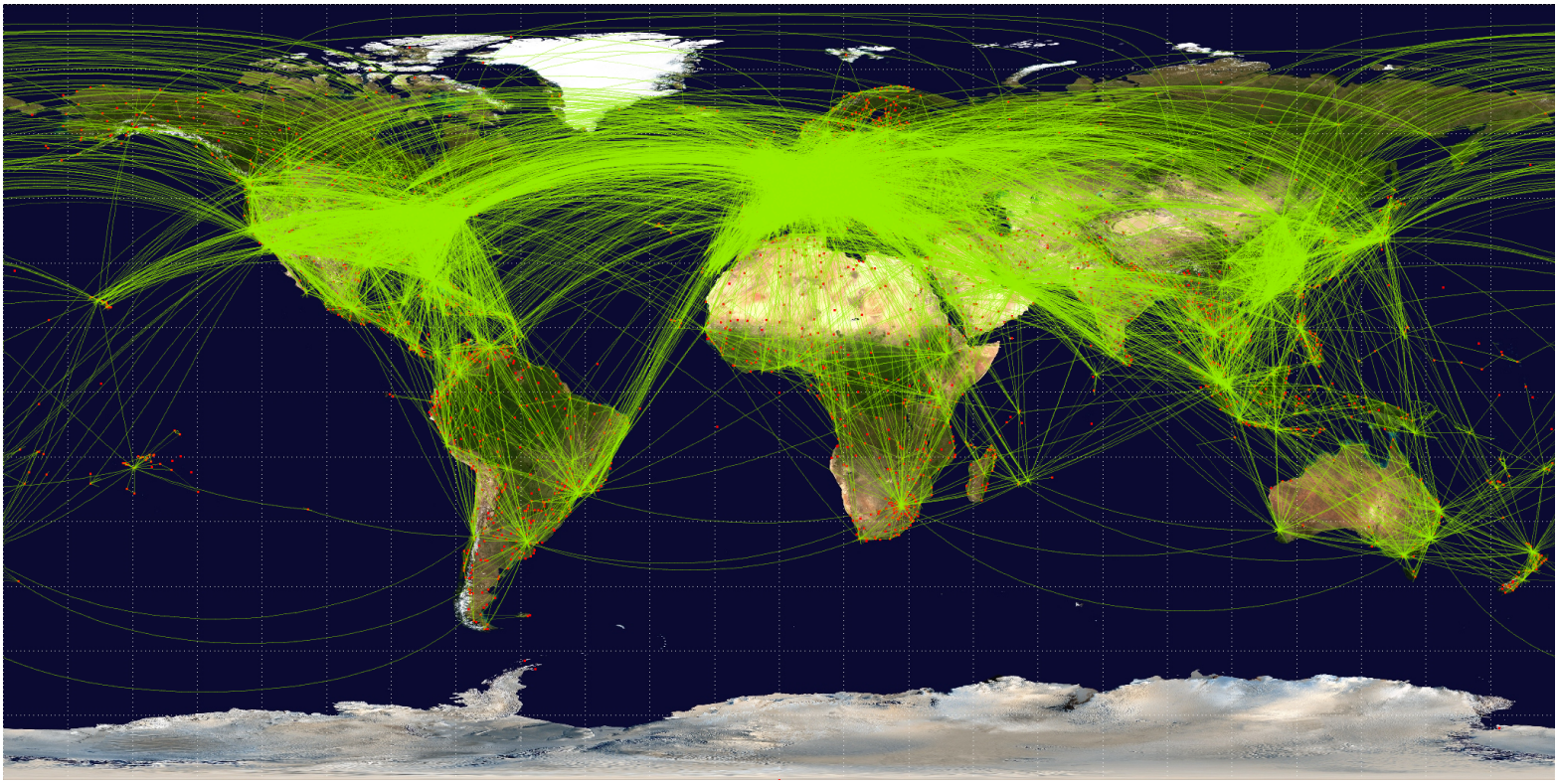
Step 2

Let's take our original idea about navigation and generalize it somewhat: let's imagine that we're given a set of "locations" (which we will refer to as **nodes**) and a set of "connections" between these locations (which we will refer to as **edges**). If we are looking at the original navigation idea, the **nodes** would be building addresses as well as street intersections, and the **edges** would be streets connecting the **nodes**. If we are looking instead at flights to various places in the world, where we want to find the shortest series of flights that will get us from one airport to another, the **nodes** would be airports, and the **edges** would be flights that connect airports. If we are instead looking at data packets that we need to send between ports in a network, the **nodes** would be ports and the **edges** would be network connections between ports. With this generalization of **nodes** and **edges**, the applications of our navigation idea are endless.

The following images were obtained from OpenFlights. Below is an image of the globe, where airports are denoted as red dots:



As we mentioned previously, we can think of our airports as **nodes**, and we can think of flights as **edges** that connect pairs of **nodes** (airports). Below is the same image of the globe, but now with flights overlain as green curves:



It should hopefully be clear now that, if we are clever with how we represent our data as a set of **nodes** and a set of **edges**, both the navigation problem and the flight problem mentioned previously can be transformed into the same computational problem: given a **node** *start* and a **node** *end*, what is the shortest path (i.e., series of **edges**) to take us from *start* to *end*?

Step 3

Formally, a graph consists of the following:

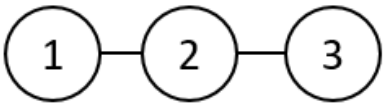
- A set of elements called **nodes** (or **vertices**)
- A set of connections between pairs of nodes called **edges**

In general, graphs do not *need* to have a sense of sequential or hierarchical order. So, unlike in trees, nodes in a graph will not necessarily have a "parent," "successor," etc. As a result, a graph can take up many different shapes and forms.

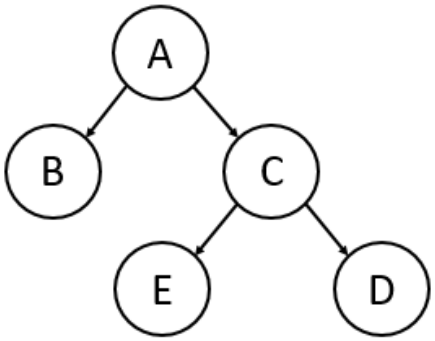
A graph can be any of the following:

- An **unstructured** set, such as a **disconnected** group of nodes:

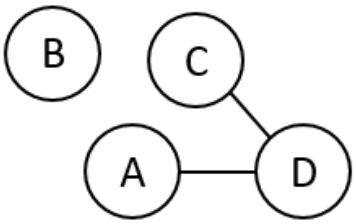
- A **sequential** set - where order matters - of **connected** nodes and edges:



- A **hierarchical** set - where rank exists (i.e. "parents", "successors", etc) - of **connected** nodes and edges:

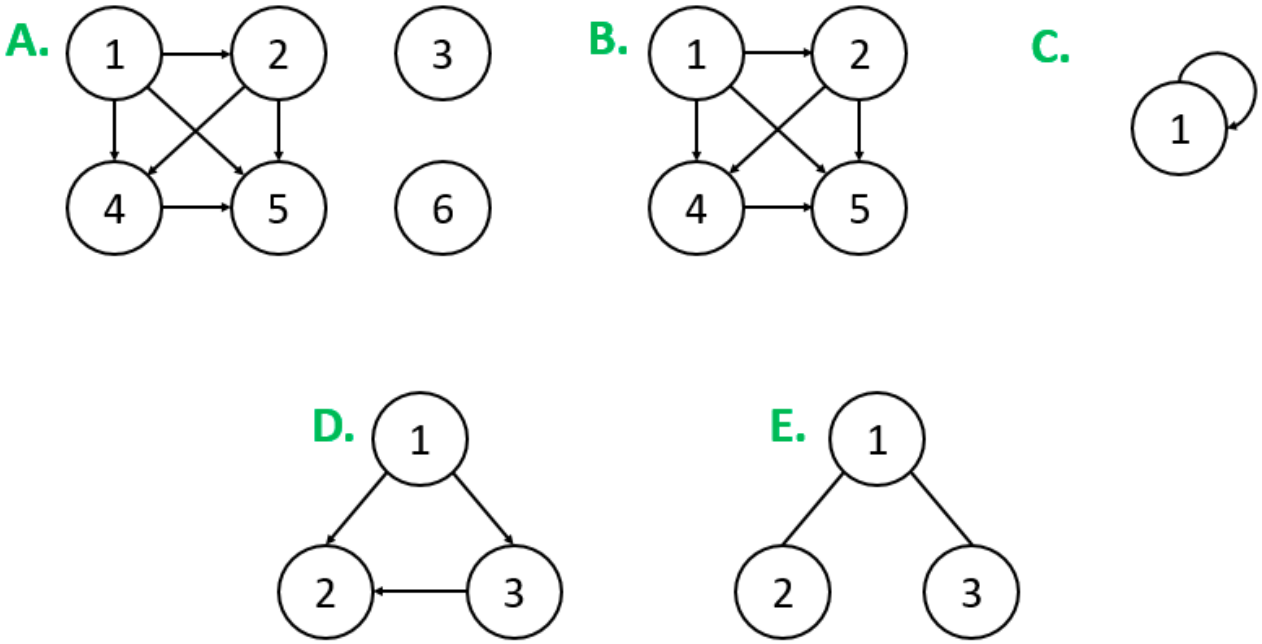


- A **structured** set of both **connected** and **disconnected** nodes and edges:



Step 4

EXERCISE BREAK: Which of the following would be defined as a graph? (Select all that apply)

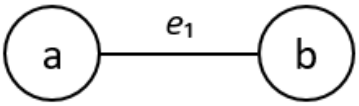


To solve this problem please visit https://stepik.org/lesson/28701/step/4

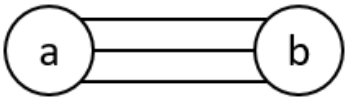
Step 5

The example graphs shown previously are quite simple, but in reality, graphs can be incredibly massive, to the point that drawing out a picture of nodes and edges will no longer feasible. As a result, we need a formal mathematical definition of a "graph". Specifically, we choose to formally represent a graph ***G* = (*V*, *E*)**, where ***V*** is the set of **vertices** in *G* and ***E*** is the set of **edges** in *G*.

Formally, we represent edge *e* as a pair (*v*, *w*) such that both *v* and *w* are vertices. For example, e_1 in the graph below would be written as (*a*, *b*).



For an arbitrary graph *G*, the set *V* follows the intuitive notion of a "set" in that each vertex in *V* is unique. However, note that the set *E* is not necessarily as intuitive: there can exist multiple equivalent edges connecting the same two vertices *v* and *w*. An example of such graph, known as a **multigraph**, can be found below, in which we have three edges connecting vertices *a* and *b*:



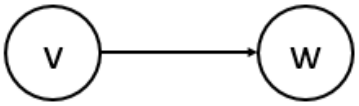
However, for the purpose of this text, we will ignore the possibility of multigraphs and operate under the assumption that we can have at most one edge connecting two given vertices.

It is also useful to know that, when talking about the sizes of graphs, |*V*| represents the total number of vertices and |*E*| represents the total number of edges. In the first example graph above, |*V*| = 2 (because there are two vertices: *a* and *b*) and |*E*| = 1 (because there is only a single edge, one that connects *a* and *b*).

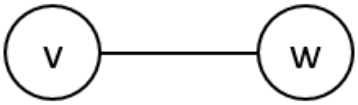
Step 6

Based on the behavior of a graph's edges, a graph is often classified as being either **directed** or **undirected**.

A **directed** graph is a graph in which each edge has a direction associated with it. For example, in a **directed** graph, *e* = (*v*, *w*) would mean that there is a connection from node *v* to *w* **but not necessarily** a connection from node *w* to *v*. As a result, (*v*, *w*) is considered an *ordered pair* (i.e., the edge will always be leaving the first node and entering the second node).



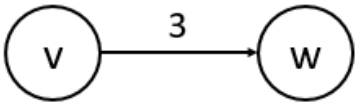
An **undirected** graph is a graph in which each edge has no particular direction associated with it. For example, in an **undirected** graph, *e* = (*v*, *w*) would mean that there is a connection from node *v* to *w* **and** from *w* to *v*. As a result, (*v*, *w*) is considered an *unordered pair* (i.e., the order in which we write the vertices doesn't matter because we are guaranteed that edge *e* connects both *v* to *w* and *w* to *v*).



STOP and Think: How could you transform an **undirected** graph into a **directed** graph?

Step 7

A **weighted** graph is a graph in which each edge has a "weight" (or "cost") associated with it. As a result, we formally represent an edge *e* = (*v*, *w*, *c*), where *c* is the "edge weight" (or "cost"). In the example below, the edge can be formally defined as (*v*, *w*, 3).



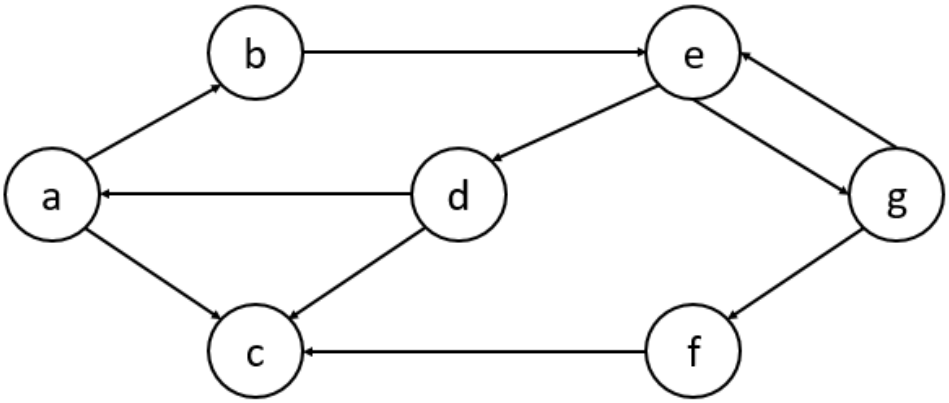
It is important to note that a "weight" can be any number (positive, negative, decimal, etc.). A weighted graph can also be either directed or undirected.

An **unweighted** graph is a graph in which each edge doesn't have a "weight" associated with it. To "transform" an unweighted graph into a weighted graph, it is thus common to think of each edge as just having a weight of 1.

STOP and Think: Could we transform an unweighted graph into a weighted graph by giving each edge a weight of 2 (as opposed to the previously mentioned weight of 1)?

Step 8

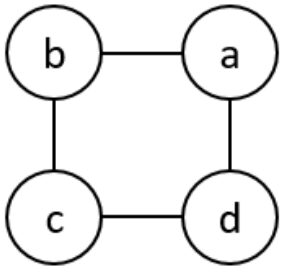
EXERCISE BREAK: Which of the following edges appear in the **directed unweighted** graph shown below? (Select all that apply)



To solve this problem please visit https://stepik.org/lesson/28701/step/8

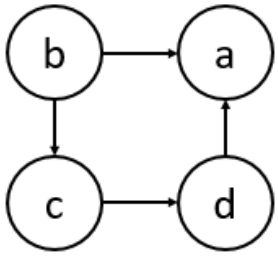
Step 9

By definition, a **cycle** in a graph is a valid path that starts and ends at the same node. For example, in the graph below, one cycle would be defined by the path $a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$.



It is important to note that, just because a directed graph may have an "undirected cycle" (i.e., a cycle that is formed by removing direction from the directed edges in a graph), there need not be a cycle in the directed graph.

STOP and Think: Does the graph below contain a cycle?



Step 10

We explored graphs and discussed some of their properties, and based on the previous exercises, we saw that it can be easy to interpret graphs when we can see them in their entirety. Also, we saw that we can solve some interesting real-world problems by simply exploring a graph. However, what happens when the number of **nodes** and/or **edges** becomes absurdly large, to the point that we can

no longer just "eyeball" solutions?

Because real-world graphs can blow up in size, before we can even begin to worry about graph exploration algorithms, we need to think about how we can represent graphs on a computer so that any graph exploration algorithms we come up with can be implemented and executed on these real-world datasets. In the next section, we will discuss various methods to represent graphs and we will analyze the trade-offs of each type of representation.