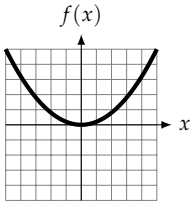
Learning objectives:

- ☐ describe basic graphs as a pair of a set of edges and a set of vertices,
- ☐ describe why graphs are useful and where they come up in computer science,
- ☐ represent graphs using adjacency matrices and adjacency lists,
- ☐ sneakpeek into types of graphs,
- ☐ sneakpeek into some graph algorithms.

One of the most important structures you will encounter in computer science is that of a *graph*. The central concept of graphs is the notion of *connections* between objects (we'll define this formally soon). Can you come up with some everyday examples in which you may want to study connections between objects?

object	connection
cities	roads
people	friends on Facebook
programs	can run concurrently
web pages	links

Graph?



Not this kind of graph, Piper, although you might argue there are connections. For example, you might consider the objects to be values in $\mathbb{R}$. There is a connection between any x and $f(x)$.

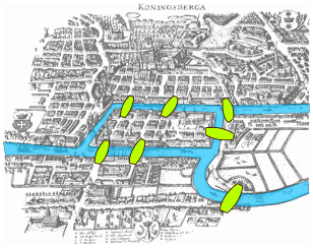
Example 1:

Suppose you are at a party with n people. If n is really big, then it's impossible for everyone to speak to each other during the party. Instead, every person p_i speaks to m_i other people ($0 \leq m_i \leq n$). After leaving the party, everyone writes down how many people they spoke with. Now that you have all the m_i data ($1 \leq i \leq n$), how many *total* conversations occurred at the party?

Click for the solution.

Example 2:

In 1736 Konigsburg, Prussia, you decide to go for a stroll. The beautiful Pregel River passes right through the city and you want to maximize the amount of time you spend looking at the river. However, you need to get back to your cs200 quizlet, so you need to keep your walk on the shorter side. You, therefore, decide that you should only cross each of the seven bridges *exactly once*. Can you cross each of the seven bridges *exactly once*?



Source

This is, in fact, the problem that gave birth to study of graphs. Leonhard Euler proved that there is no solution. :(

It turns out that you need to have either zero or two nodes of odd degree to cross every bridge once. We call this type of path through a graph as an *Eulerian path*.

1 Graph definitions

Before we can do some fancy things with graphs, we need to make a few definitions.

Definition 1. *A **graph** G is a pair of sets $G = (V, E)$ where V is a* nonempty *set of items called **vertices** (or **nodes**) and E is a set of $2-$item subsets of V called **edges**.*

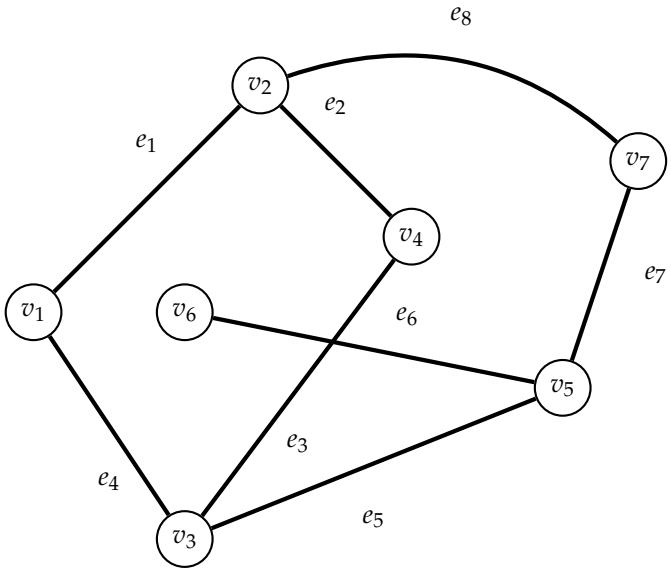


Figure 1: Example of a graph with 7 vertices and 8 edges.

Your first graph!

For example, consider the graph of Figure 1 with 7 vertices and 8 edges. The set of vertices V is

$$V = \{v_1,\ v_2,\ v_3,\ v_4,\ v_5,\ v_6,\ v_7\},$$

and the edges are

$$\begin{aligned} E &= \{e_1,\ e_2,\ e_3,\ e_4,\ e_5,\ e_6,\ e_7,\ e_8\} \\ &= \{\{v_1, v_2\},\ \{v_2, v_4\},\ \{v_3, v_4\},\ \{v_1, v_3\},\ \{v_3, v_5\},\ \{v_5, v_6\},\ \{v_5, v_7\},\ \{v_2, v_7\}\}. \end{aligned}$$

We say that two vertices $v_i,\ v_j$ are **adjacent** if $\{v_i, v_j\} \in E$. We also say that an edge $e = \{v_i, v_j\}$ is **incident** to endpoints v_i and v_j.

Let's return to the party example from the beginning of the lecture. In order to determine the total number of conversations that occurred, we need to make an important definition.

Definition 2. *The **degree** of a vertex v is equal to the number of edges incident to that vertex, and is denoted by $deg(v)$.*

In the graph of Figure 1, $\deg(v_1) = \deg(v_4) = \deg(v_7) = 2$. Also, $\deg(v_2) = \deg(v_3) = \deg(v_5) = 3$ and $\deg(v_6) = 1$.

Now, consider what happens when you add up the degrees of every vertex in a graph. This leads us to an important property of graphs, known as the *handshaking lemma*.

Lemma 1. *The total sum of the vertex degrees is equal to twice the number of edges in a graph.*

Proof. We use a direct proof. Let $G = (V, E)$ be a graph. Traverse every edge $e \in E$ and add the contribution of that edge $e = \{v_i, v_j\}$ to the total degree of the endpoint vertices v_i and v_j.

$$\sum_{e \in E} (1 + 1) = 2|E| = \sum_{v \in V} \deg(v)$$

since we have added the contribution of every edge incident to every vertex. □

Therefore, we just need to add up the number of conversations reported by every person and divide by 2 to count the total number of conversations at the party.

2 Properties of graphs

The following are some important properties of graphs we will consider in this course. Note that multiple properties can be combined for a single graph.

2.1 Weighted

As we will see in future lectures, it is often useful to assign **weights** to the edges or vertices of a graph. For example, the original graph of Figure 1 has been redrawn with weights on the edges in Figure 3. We may want to extract a subset of the edges of the graph and solve a shortest path problem or a matching problem.

2.2 Simple

A graph is **simple** if it has *no loops* or *multiple edges* (often *multiedges*). The graphs of Figure 1 and 3 are simple, but the graphs of Figure 2 are not simple.

2.3 Directed

A **directed graph** (or *digraph*) is a graph $G = (V, E)$ in which every edge $e \in E$ has an **orientation**. That is, an edge $e = \{v_i, v_j\}$ is *directed* from v_i to v_j, and v_i may be called the **tail**, whereas v_j is called the **head**. The graph in Figure 3 is an example of a directed graph.

Other properties?

This is not an exhaustive list! We will see other properties such as *acyclic* and *connected* graphs, which further requires us to define things like *walks*, *paths*, *cycles*.

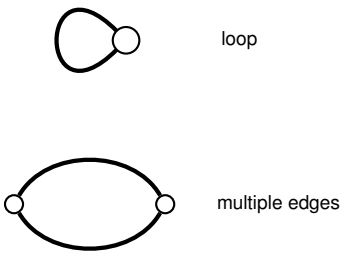


Figure 2: Examples of graphs that are not simple.

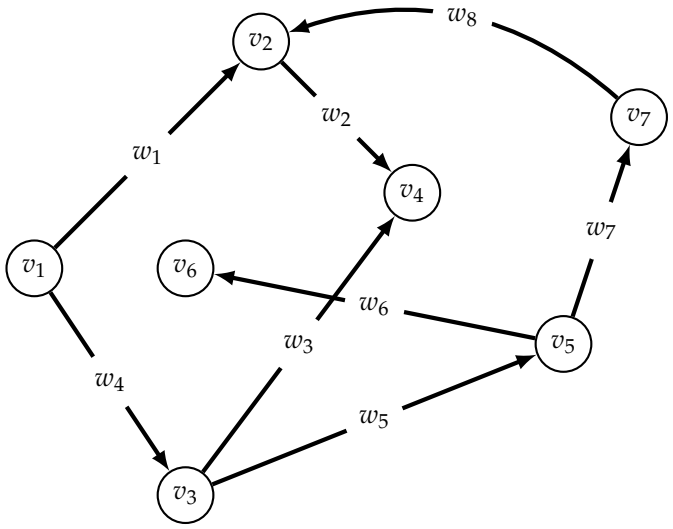


Figure 3: Example of a directed graph with weights on the edges.

3 Representing graphs

Because graphs are so useful in computer science, we need a way to represent them in a computer!

3.1 Adjacency matrix

One way to represent a graph is using an **adjacency matrix**. This is a matrix A of size $|V| \times |V|$ in which the entry a_{ij} is nonzero if there is an edge connecting vertex v_i and v_j. In the case of a directed graph, entry a_{ij} is nonzero if the edge is directed from v_i to v_j. For example,

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	1	1	0	0	0	0
v_2	1	0	0	0	0	0	1
v_3	1	0	0	1	1	0	0
v_4	0	1	1	0	0	0	0
v_5	0	0	1	0	0	1	1
v_6	0	0	0	0	1	0	0
v_7	0	1	0	0	1	0	0

Adjacency matrix for the graph of Figure 1.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	w_1	w_4	0	0	0	0
v_2	0	0	0	w_2	0	0	0
v_3	0	0	0	w_3	w_5	0	0
v_4	0	0	0	0	0	0	0
v_5	0	0	0	0	0	w_6	w_7
v_6	0	0	0	0	0	0	0
v_7	0	w_2	0	0	0	0	0

Adjacency matrix for the graph of Figure 3.

3.2 Adjacency list

An alternative method for representing graphs is using an **adjacency list**. In an adjacency list for an unweighted graph, a list of neigh-

boring vertices is stored for every vertex. In an adjacency list for an edge-weighted graph, a list of pairs is stored for every vertex. The first entry in the pair is the neighboring vertex, whereas the second entry is the weight. For example,

vertex	list
v_1	(2,3)
v_2	(1,4,7)
v_3	(1,4,5)
v_4	(2,3)
v_5	(3,6,7)
v_6	(5)
v_7	(2,5)

Adjacency list for the graph of Figure 1.

vertex	list
v_1	$(v_2, w_1), (v_3, w_4)$
v_2	(v_4, w_2)
v_3	$(v_4, w_3), (v_5, w_5)$
v_4	-
v_5	$(v_6, w_6), (v_7, w_7)$
v_6	-
v_7	(v_2, w_8)

Adjacency list for the graph of Figure 3.

Example 3:
What is the adjacency matrix and list for the graph of Figure 4?

Solution:

	v_1	v_2	v_3	v_4
v_1	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
v_2	$\frac{1}{2}$	0	0	$\frac{1}{2}$
v_3	0	0	0	1
v_4	0	0	0	1

vertex	list
v_1	$(v_2, \frac{1}{3}), (v_3, \frac{1}{3}), (v_4, \frac{1}{3})$
v_2	$(v_1, \frac{1}{2}), (v_4, \frac{1}{2})$
v_3	$(v_4, 1)$
v_4	$(v_4, 1)$

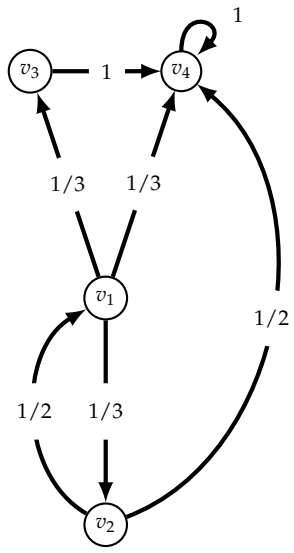


Figure 4: Graph used in the adjacency matrix & list examples.

Example 4:
Write pseudocode to determine the degree of vertex v in a graph using (a) the adjacency matrix and (b) the adjacency list. Which one do you think is faster?

Solution:
The algorithms are provided in Algorithms 1 and 2. Using the adjacency list is faster because it requires only one operation (calculating the length of the list). With the adjacency matrix, about $|V|$ operations are used.

4 Graph problems

Graphs allow us to solve some really interesting problems. In some cases, we will be able to solve them in polynomial time. In others, we may only be able to *verify* a solution in polynomial time, if such

```
degree_matrix(v_i, A)

    input:
    v_i: vertex,
    A: adjacency matrix for G = (V, E),

    output: degree of vertex v_i
1   degree = 0
2   for v_j ∈ V
3       degree = degree + A(v_i, v_j)
4   return degree
```

Algorithm 1: Calculating the degree of a vertex using the adjacency matrix.

```
degree_list(v_i, A)

    input:
    v_i: vertex,
    A: adjacency list for G = (V, E),

    output: degree of vertex v_i
1   return length(A[v])
```

Algorithm 2: Calculating the degree of a vertex using the adjacency list.

a solution exists. This brings us to the distinction between problems that are in P, NP, NP-complete and NP-hard.

- P: problems which can be *solved* in polynomial time.
- NP: problems for which an algorithm can be *verified* in polynomial time.
- NP-complete: problems which, if you find a solution, then all problems in NP can be solved in polynomial time.

I want to solve an NP-complete problem!

I hope you do, Piper! There are more than 3000 known NP-complete problems. If you solve one, you solve them all! Anyone who solves an NP-complete problem wins a million dollars.

4.1 Coloring

Given a graph G and k colors, assign a color to each vertex so that adjacent vertices get different colors. Certain problems, such as determining the minimum value of k (called the *chromatic number* of G) is NP-complete. Graph coloring has applications to course exam scheduling. For example, we may want to assign a time slot (color) for the exam of each course in the college such that no students (who may possibly be in the same course) have exam conflicts.

4.2 Matching

Matching is an important problem that has applications to dating, doctor-hospital assignments and minimum course requirements. Given a graph $G = (V, E)$, a matching M is a subgraph of G where every node has degree 1. We won't talk too much about matching in this course, but if you are interested, please see the stable marriage problem. Some matching problems are in P, but some are in NP.

4.3 Searching

Given a graph $G = (V, E)$, there are a few ways we can search for a node that satisfies a particular property. The two methods we will study are *breadth-first search* and *depth-first-search*. Searching is in P.

4.4 Shortest path

Determining the shortest path between vertices in a graph is a very important problem! We won't talk too much about shortest path algorithms (Dijkstra, Bellman-Ford, etc.) in this course but you will see these in CS 302. Determining the shortest path between two vertices in a graph is in P. However, determining whether there is a shortest path in a graph G from a starting vertex, that visits *every* vertex in G, is NP-complete. See the Traveling Salesman Problem.