Graph Basics

Daryll Ko

Contents

Motivation 1
Terms 2
Representations 6
Operations 8

Motivation

Consider the following problem:

> Hoshino is friends with Shiroko.
> Serika is friends with Hoshino.
> Aris is friends with Momoi.
> Hoshino is friends with Hina.
> Yuzu is friends with Momoi.
> Hina is friends with Shiroko.
> Serika is friends with Shiroko.
> Yuzu is friends with Aris.
>
> A message can be passed from one person to another if they are friends.
> Is it possible to pass a message from Hoshino to Aris?

If we look at this problem as is, our first instinct is to, well, try to see who Hoshino can pass a message to. For example:

- Hoshino can directly pass a message to Serika, since they're friends.
- Serika can then pass that message to Shiroko, since they're friends.
- Shiroko can then pass that message to Hina, since they're friends.
- …then now we're stuck.

From this, it seems like we *can't* pass a message from Hoshino to Aris. And that would be right!

But how can we be *sure*? What if we just missed a line saying that "Hoshino is friends with Aris"?[1] What we want is to encode these *relationships* in such a way that answering questions like the one above can be done at a cursory glance.

And what better way to do that than to *draw* those relationships?

> **Tip 1.**
> If possible, draw a picture that tells you what's going on in a problem.

So let's say we draw a *line* between two students if they're friends. We'll end up with the following picture:

[1] Don't worry, there isn't one.

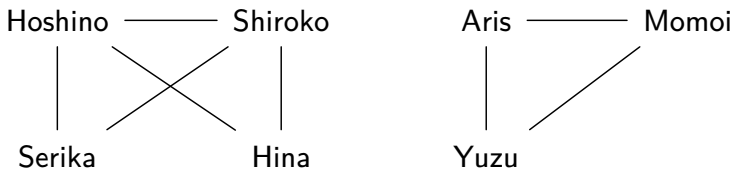


Figure 1: A picture representing the friend relationships above.

Here we can clearly see that Hoshino can't "reach" Aris! This matches our hunch from earlier.

This is the crux of **graph theory**: it gives us a way to represent *entities* and the *relationships* between those entities.[2]

Terms

> **Tip 2.**
> Understand, don't just memorize. In the assessments, we won't check if you know what `<insert term here>` means; we want to see if you can correctly *apply* the ideas to solve problems.
>
> For the exams, expect there to be a cheatsheet containing all of the terms covered.

It's definitely possible to talk about Figure 1 without introducing new terms. In fact, that's how the question is phrased!

> There isn't a series of friendships connecting Hoshino and Aris, so Hoshino can't pass a message to Aris.

But how do we describe *general* entities and relationships? We can't always use the word "friends": that would be downright wrong if our "relationships" were about *enemies* instead. Also, "series of friends" is kinda vague here: we ideally want a more *precise* way to describe this.

That's why the math community decided to agree on a set of terms to describe graphs in general.[3]

- A **vertex** (or **node**) is meant to describe an entity.
  In Figure 1, Hoshino and Hina are *vertices* (the plural of "vertex"). Koyuki is **not** a vertex.
- An **edge** is meant to describe a relationship between two entities.
  In Figure 1, there is an edge between Hoshino and Hina. There is **no** edge between Shiroko and Aris.
  Mathematically, we usually use an *unordered pair* to denote an edge. I'll write edges like this: ⟨Hoshino-Hina⟩.
- A **graph** is a set of vertices together with a set of edges. Formally,

[2] You might have heard of the term "entity-relationship diagram". The word gives me nightmares, so let's forget it exists.

[3] There's something valuable in realizing that terminology is something that is *agreed upon* and not a fundamental truth (in the mathematical sense). This is something you get to appreciate if you learn a second language: it's not the words themselves but the *meanings* that truisms are based upon. It's also why I don't exactly recommend language learning from a "direct translation" standpoint: your goal isn't to find a perfect mapping of a sentence from one language to another; it's to understand and appreciate *how* the second language conveys the meaning of the sentence. I'm sure there's a philosophy-related thing about this idea.

A **graph** G is a pair of sets (V, E), where V denotes the vertices of G, and E denotes the edges of G.

In Figure 1,

$$V = \{\mathsf{Hoshino}, \mathsf{Shiroko}, \mathsf{Serika}, \mathsf{Hina}, \mathsf{Aris}, \mathsf{Momoi}, \mathsf{Yuzu}\}$$
$$E = \{\langle \mathsf{Hoshino\text{-}Shiroko}\rangle, \langle \mathsf{Hoshino\text{-}Hina}\rangle, \langle \mathsf{Hoshino\text{-}Serika}\rangle, \langle \mathsf{Shiroko\text{-}Serika}\rangle, \langle \mathsf{Shiroko\text{-}Hina}\rangle, \langle \mathsf{Aris\text{-}Momoi}\rangle, \langle \mathsf{Momoi\text{-}Yuzu}\rangle, \langle \mathsf{Aris\text{-}Yuzu}\rangle\}.$$

If we have a list of pairs of friends, one natural question to ask is "Who is friends with this particular student"? (e.g., "Who is friends with Hoshino?") We have some more terms to capture these ideas about what happens "locally" in the graph:

- The **neighborhood** N of a vertex v is the set of vertices that are directly connected by an edge to v.
  In Figure 1, $N(\mathsf{Hoshino}) = \{\mathsf{Shiroko}, \mathsf{Serika}, \mathsf{Hina}\}$. Note that:
  - $\mathsf{Shiroko} \in N(\mathsf{Hoshino})$, since Hoshino and Shiroko are directly connected by an edge (i.e., they are friends).
  - $\mathsf{Momoi} \notin N(\mathsf{Hoshino})$, since Hoshino and Momoi are **not** directly connected by an edge (i.e., they are **not** friends).
- The **degree** of a vertex is the size of its neighborhood.
  In Figure 1, the degree of Hoshino is 3 (that is, Hoshino has 3 friends). In other words, there are 3 students that are directly connected by an edge to Hoshino.

I don't know about you, but "directly connected by an edge" is a lot of words to say. Fortunately, there's also a term for that!

- Two vertices are **adjacent** iff there is an edge between them.
  In Figure 1, Hoshino and Hina are adjacent. Shiroko and Aris are **not** adjacent.

This gives us an alternative definition for neighborhood:

- The **neighborhood** N of a vertex v is the set of vertices adjacent to v.

Question 3.
What does it mean if a vertex has a degree of 0?

In Figure 1, the entities are represented by the vertices. But what if the *edges* also have some sense of identity? Let's take a part of the graph and give each edge its own unique color:

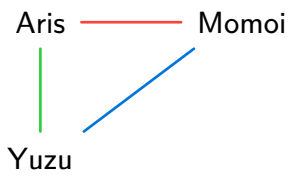


Figure 2: A part of the graph, where edges have identities.

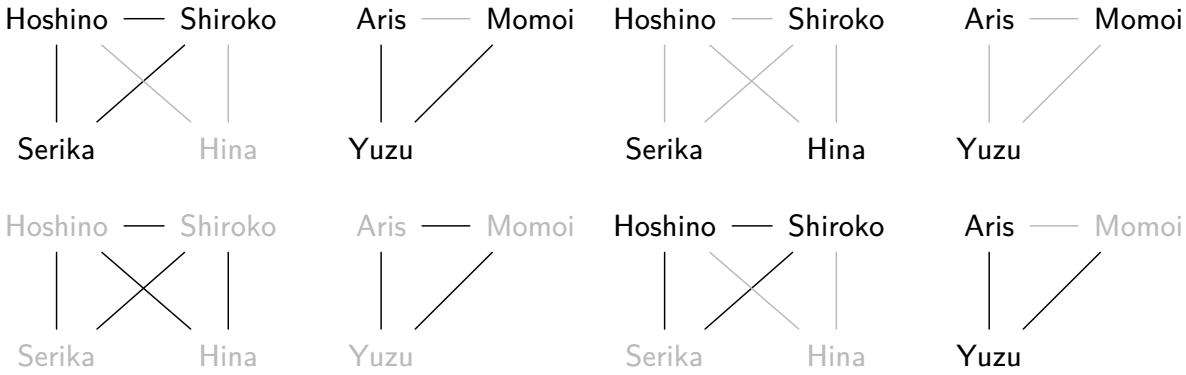
To say something like "Aris is one end of the red edge", we have the following term:

- A vertex v is **incident** to an edge e iff v is one of the endpoints of e.
  In Figure 2, Aris is incident to the red edge. Yuzu is **not** incident to the red edge.
  In general, $\langle u\text{-}v\rangle$ is incident to both u and v.

Speaking of "parts of graphs", we also have a precise way of defining that:

- A graph $G' = (V', E')$ is a **subgraph** of the graph $G = (V, E)$ iff $V' \subseteq V$ and $E' \subseteq E$.

For example, each of the non-greyed parts below is a subgraph of Figure 1:



It might feel weird to have "dangling" edges, as can be seen in the third and fourth graphs. It *is* weird, so there's a related term that avoids this:

- Let $G = (V, E)$ be a graph, and let $V' \subseteq V$. The **subgraph induced by V'** is the graph $G' = (V', E')$, where $\langle u\text{-}v \rangle \in E'$ iff $u \in V'$, $v \in V'$, and $\langle u\text{-}v \rangle \in E$.

Exercise 4.
Let V' be a subset of the vertices of a graph G. Show that the subgraph induced by V' really is a subgraph.

Okay, so now we have *some* grasp on the different ideas related to the graph. We can now take this statement…

> There isn't a series of friendships connecting Hoshino and Aris, so Hoshino can't pass a message to Aris.

…and rewrite it like this:

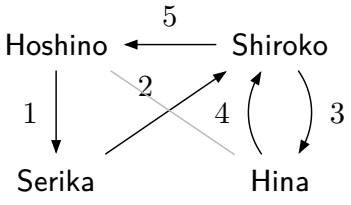
> There isn't a series of **edges** connecting the **vertices** Hoshino and Aris, so Hoshino can't pass a message to Aris.

That's…not a lot of progress.

What we still need here is a way to represent, as the statement above says, a *series* of edges. And so we also have terms for those!

- A **walk** is a sequence of vertices $v_1 \to v_2 \to \ldots \to v_\ell$ such that the edge $\langle v_i\text{-}v_{i+1} \rangle$ exists for all $1 \leq i \leq \ell - 1$.
  A walk with e edges has a **length** of e.

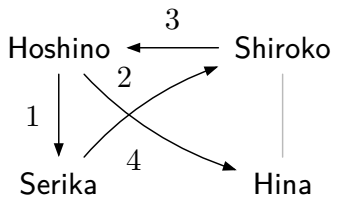
Here's the walk Hoshino → Serika → Shiroko → Hina → Shiroko → Hoshino, with length 5:



- A **trail** is a walk that doesn't repeat edges.

  The walk above isn't a trail (because the edge ⟨Shiroko-Hina⟩ is used twice).

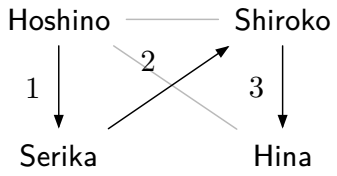
Here's the trail Hoshino → Serika → Shiroko → Hoshino → Hina:



- A **path** is a walk that doesn't repeat vertices.

  The trail above isn't a path (since Hoshino is visited twice).

Here's the path Hoshino → Serika → Shiroko → Hina:



Exercise 5.
Show that every path is also a trail.

So why do we care about walks, trails, and paths? Well, they're what allow us to establish *indirect* relations between vertices:

- We say that two vertices u and v are **connected** iff there is a walk that starts at u and ends at v.
  In Figure 1, Serika and Hina are connected. Hoshino and Aris are **not** connected.

This notion of connected-ness has some nice properties:

- It's *reflexive*; for any vertex v, there's always a walk (with length 0) that starts and ends at v.
- It's *symmetric*; for any two vertices u and v, there's a walk that starts at u and ends at v iff there's a walk that starts at v and ends at u.

Exercise 6.
Prove this.

- It's *transitive*; for any three vertices u, v, and w, if there's a walk that starts at u and ends at v and a walk that starts at v and ends at w, then there's a walk that starts at u and ends at w.

Exercise 7.
Prove this.

Because connected-ness satisfies these three properties, it's an equivalence relation, so we can establish equivalence classes for the vertices!

- A **connected component** is a *maximal* set of vertices C such that for any two vertices u and v in C, u and v are connected.

  By maximal, we mean that if C is a connected component, then for all vertices $u, v \in V$, $u \in C$ and $v \in C$ iff u and v are connected.

  The connected components of Figure 1 are {Hoshino, Serika, Shiroko, Hina} and {Aris, Momoi, Yuzu}.

We can now rewrite

> There isn't a series of **edges** connecting the **vertices** Hoshino and Aris, so Hoshino can't pass a message to Aris.

as

> The **vertices** Hoshino and Aris are not in the same **connected component**, so Hoshino can't pass a message to Aris.

Question 8.
Now, using the past few terms we just introduced, what does it mean if a vertex has a degree of 0?

Representations

Let's consider the list of friend relationships that we have:

> Hoshino is friends with Shiroko.
> Serika is friends with Hoshino.
> Aris is friends with Momoi.
> Hoshino is friends with Hina.
> Yuzu is friends with Momoi.
> Hina is friends with Shiroko.
> Serika is friends with Shiroko.
> Yuzu is friends with Aris.

Suppose I ask you to put these relationships in a text file. What would you do?

Well, the straightforward solution is just to, well, literally put those relationships in the text file:

```
Hoshino is friends with Shiroko.
Serika is friends with Hoshino.
Aris is friends with Momoi.
Hoshino is friends with Hina.
Yuzu is friends with Momoi.
Hina is friends with Shiroko.
Serika is friends with Shiroko.
Yuzu is friends with Aris.
```

If we're going to process this text file (using a Python script, for example), it would be nice to have a uniform format for the data. So we can simplify it like this:

```
Hoshino Shiroko
Serika Hoshino
Aris Momoi
Hoshino Hina
```

```
Yuzu Momoi
Hina Shiroko
Serika Shiroko
Yuzu Aris
```

What we're essentially doing here is listing out the *edges* of the graph. This is the **edge list** representation.

So that's it, right? We now have a way to represent graphs for computers to process. Yay! Yay?

Well…not so fast. This is certainly *one* way to represent a graph, but are there alternatives?

If you think about it, since these are *edges*, it's easy to do things related to the edges of the graph. For example, getting the edges in lexicographical order is pretty straightforward[4]:

```
Aris Momoi
Hina Shiroko
Hoshino Hina
Hoshino Shiroko
Serika Hoshino
Serika Shiroko
Yuzu Aris
Yuzu Momoi
```

Additionally, if the relationships are given to you as a list anyway, you don't have to spend too much brainpower encoding those relationships (just copy-and-paste them into the file)!

But what if we want information not about the edges, but the *vertices* and their relationships?

Consider what we have to do to answer the following question:

> Are Aris and Yuzu friends?

Using the text file above, we'd have to check if each of the lines is either `Aris Yuzu` or `Yuzu Aris`. For a very long text file, that doesn't seem very fast.[5]

So what if we *directly* encode that information? Like, literally put it in the text file:

```
"Are they friends?"

        Hoshino Shiroko Serika Hina Yuzu Aris Momoi
Hoshino            ✓      ✓     ✓
Shiroko    ✓               ✓     ✓
Serika     ✓       ✓
Hina       ✓       ✓
Yuzu                                     ✓     ✓
Aris                                ✓            ✓
Momoi                               ✓     ✓
```

Now, all you have to do to check if Aris and Yuzu are friends is to check if there's a ✓ in the cell in both Aris' row and Yuzu's column.

[4]For example, you can pipe the text file through the `sort` command. You remember this from CS 11, right? …right?

[5]It's $\mathcal{O}(e)$, where e is the number of edges of the graph. You remember $\mathcal{O}$ from CS 11, right? …right?

This is a *matrix* that encodes the adjacency information between the vertices, so we call it the **adjacency matrix** representation.

Now, what if you want to answer the following question?

> How many friends does Hoshino have?

We'd encounter the same problem as with the previous question if we still used the edge list representation, so what about the adjacency matrix representation?

It doesn't seem so bad: all we have to do is count the number of ✓s in Hoshino's row. But what would happen if the graph was really big? You'd essentially have a *spreadsheet* containing the relationship (or absence of one) between *every single pair* of students. In other words, this would take up quite a lot of *space*.[6]

So to work around that, let's try to "compress" each of the rows in the matrix into an ordinary list. That would look like this:

```
Hoshino: Shiroko, Serika, Hina
Shiroko: Hoshino, Serika, Hina
Serika: Hoshino, Shiroko
Hina: Hoshino, Shiroko
Yuzu: Aris, Momoi
Aris: Yuzu, Momoi
Momoi: Yuzu, Aris
```

Now we can (easily) see that Hoshino has 3 friends: Shiroko, Serika, and Hina.

This time, we're storing the adjacency information for each vertex as a *list*, so we call this the **adjacency list** representation.

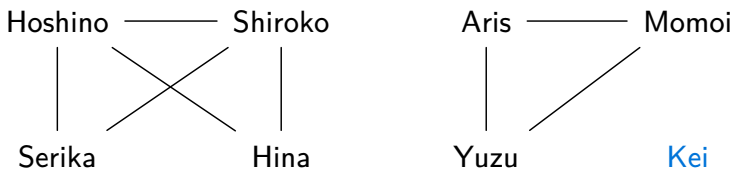
Operations

So we know some things about graphs, and we know some ways to represent them. What now happens if we *change* them? Relationships can change, after all.

Let's look at some ways to modify graphs:

- Add a vertex

  We can add a new student, Kei:



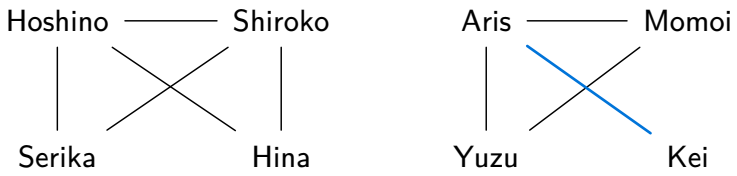
In other words, we add Kei to the set V.

Question 9.
How does this affect the edge list, adjacency matrix, and adjacency list representations of the graph?[7]

[6] It takes $\mathcal{O}(v^2)$ space, where v is the number of vertices.
[7] You'll be seeing this question a lot!

- Add an edge

  Kei and Aris can become friends:

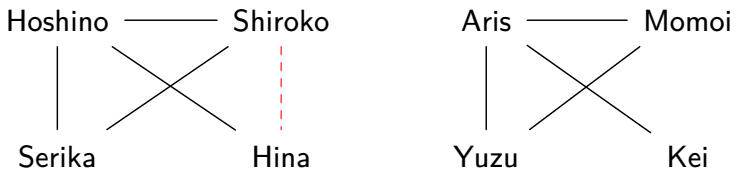


In other words, we add the edge ⟨Aris-Kei⟩ to the set E.

Question 10.
How does this affect the edge list, adjacency matrix, and adjacency list representations of the graph?

- Remove an edge

  Shiroko and Hina can become enemies:

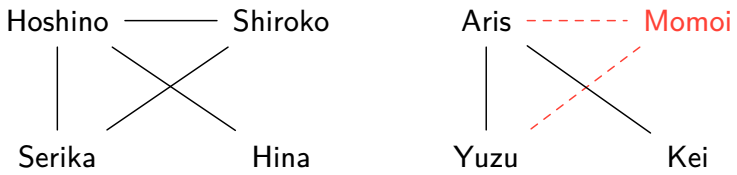


In other words, we *remove* the edge ⟨Shiroko-Hina⟩ from the set E.

Question 11.
How does this affect the edge list, adjacency matrix, and adjacency list representations of the graph?

- Remove a vertex

  Let's say Momoi suddenly disappears:



As you can see, when removing a vertex, the edges incident to that vertex also disappear.

Here, we remove Momoi from V and the edges ⟨Aris-Momoi⟩ and ⟨Yuzu-Momoi⟩ from E.

Question 12.
How does this affect the edge list, adjacency matrix, and adjacency list representations of the graph?

Being able to modify graphs is a pretty powerful thing: you're adding and removing objects and the *relationships* between them. As such, things can also get pretty weird.

Let's say we receive two additional pieces of information:

Hoshino is friends with Hina.
Aris is friends with Aris.

Technically, these are two new edges in our graph. So let's add them!

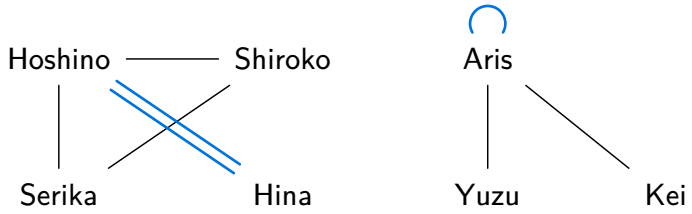


Figure 3: The same graph with the new info.

Oh, our graph now looks a bit…peculiar. What's happening here?

- There are now *two* edges between Hoshino and Hina. These edges are what we call **parallel edges**.

  Note that the set of edges E of the graph must now be a *multiset* (i.e., a set that allows for duplicates) if we're supporting parallel edges.

Question 13.
How does this affect the edge list, adjacency matrix, and adjacency list representations of the graph?

- There is now an edge connecting Aris and itself. This is what we call a **self-loop**.

  At first glance, self-loops seem innocent, but they actually affect some of the definitions from a while ago.

Question 14.
Using our earlier definition, what is the degree of a vertex with a self-loop (and no other edges connected to it?)

Once self-loops come into the picture, we need to tweak the definition of "degree" a bit:[8]

- The **degree** of a vertex is the number of times it appears as the endpoint of an edge. In particular, a self-loop at v contributes 2 to the degree of v.

As you can see, life is a lot simpler if we didn't have to deal with self-loops and parallel-edges.
Like, literally a lot simpler.

- A graph is **simple** iff it has no self-loops and parallel edges.

  Figure 1 is a simple graph.
  Figure 3 is **not** a simple graph.

[8]The reason we're doing this will be evident soon. That's a surprise move that can help us later.