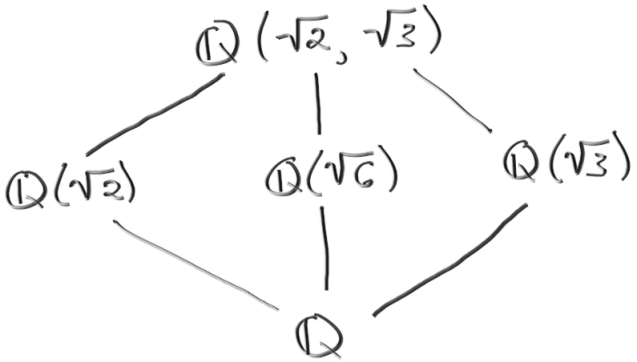
Project 3: a field extension diagram

Jennifer Brown

January 1, 2024

Note: This tutorial is currently only available as a pdf document.

In this project, we introduce Ti*k*Z's graph drawing algorithms, and demonstrate how you can use them to create "nice-looking" diagrams without having to place each element of the diagram by hand as we did in previous projects. We'll recreate this field extension diagram (such as you might see in an Abstract Algebra class):



First off, we are going to need to change LaTeX engines. (A LaTeX engine is a compiler. I'm not sure why it's called an engine in this context, but it's a common usage.) Pictures that use Ti*k*Z's graph drawing packages need to be compiled with LuaLaTeX. If you try to compile any of the pictures in this project with pdfLaTeX it may or may not throw an error, but the pictures will definitely not look right.

Switching LaTeXengines in Overleaf is just a matter of finding a dropdown menu – see this help page:

`https://www.overleaf.com/learn/how-to/Changing_compiler`

(Briefly: click on the Menu icon at the top left of the page, find the "Compiler" menu under "Settings", and set it to "LuaLaTeX".)

By the way, as far as I've been able to tell, you can just use the LuaLaTeX engine all the time; it can deal with everything that pdfLaTeX can. There might be some very subtle differences in the pdf file, but nothing very noticeable.

What you *will* notice, if you use the graph drawing packages very much, is that pictures using these packages can take a relatively long time to compile. If you are working on a document that has mostly "ordinary" LaTeX with a few pictures, you can comment out the pictures temporarily if you are working on a non-picture part of the document. For this, load the verbatim package by putting `\usepackage{verbatim}` in your preamble. Then, to comment out a block of code, enclose the stuff to be commented within `\begin{comment}` ... `\end{comment}`

We are going to draw our field extension diagram using the Layered Layouts library (see the page `https://tikz.dev/gd-layered` from the PGF/Ti*k*Z Manual for much more about this library and the algorithm behind it). To use this library, we include all of the following in our preamble:

```
\usepackage{pgfplots}
\usetikzlibrary{arrows.meta,graphs,graphdrawing}
\usegdlibrary{layered}
```

One more thing to include in the preamble:

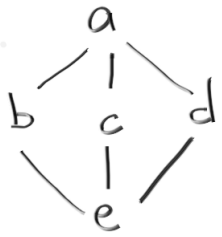
```
\pgfplotsset{compat=1.18}
```

This is for a backwards-compatibility issue that may cause a warning (though not an actual error) upon compiling.

Our field extension diagram is basically a graph, but instead of little dots for vertices, we have *nodes* where we'll be putting some text in math mode. For example, the top node in the picture will contain the text $\mathbb{Q}(\sqrt{2},\sqrt{3})$. Since we'll be using the Blackboard Bold font `mathbb`, for the rational numbers $\mathbb{Q}$, we need to include the `amssymb` package in our preamble:

```
\usepackage{amssymb}
```

Let's get drawing! First we're going to create a prototype diagram to make sure the diagram's shape is how we want it. In order to avoid a lot of clutter, we're first going to call the nodes `a` through `e`:



We start by declaring that we're going to be using one of Ti*k*Z's graph drawing algorithms by using the `\graph [⋯]` command. In the square brackets go the graph drawing library we'll be using – for this project, it's `layered layout`. Then between the curly braces we'll put our edges that will connect the vertices `a` through `e`. Note that the code below doesn't generate any output; we're just telling Ti*k*Z what kind of graph we'd like to draw.

```
\begin{tikzpicture}
\graph [layered layout]{

};
\end{tikzpicture}
```

The Layered Layout algorithm, as the name suggests, will arrange nodes into layers. The default is that the layers will be horizontal. Layers are kind of like levels in a tree diagram, but there is no requirement that the graphs be trees. For our picture, the first level is the node `a` = $\mathbb{Q}(\sqrt{2}, \sqrt{3})$. This node is connected by edges to three nodes on the second level, `b`, `c`, and `d`. Also, the edges are just plain edges, without any arrows. We tell Ti*k*Z all of this with the command

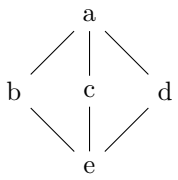
```
a -- {b, c, d};
```

Each of the nodes `b`, `c`, and `d` on the second level is connected by an edge to the single vertex on the third level, `e`:

```
{b, c, d} -- e;
```

That's it – there are no more levels, and we've told Ti*k*Z about all of the nodes and edges. Let's put these two commands, `a -- {b, c, d};` and `{b, c, d} -- e;`, into our graph:

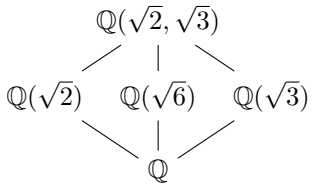
```
\begin{tikzpicture}
\graph [layered layout]{
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```



This graph is isomorphic to the one we want; only the text in the nodes needs to be changed. Placing text in a node when using Ti*k*Z's graph drawing algorithms is done in a different way from what we've seen before: you don't place text

between curly braces; instead, you place it in double quotes " $\cdots$ ". You need to tell Ti*k*Z explicitly that you're in math mode, too. In the next picture, we've replaced our temporary labels `a` through `e` with the fields $\mathbb{Q}(\sqrt{2}, \sqrt{3})$ etc. *Pro tip:* type out all of your complicated math or other node text in a separate place – a plain text document in another tab, for example – and copy/replace the temporary labels one by one.

```
\begin{tikzpicture}
\graph [layered layout] {
"$\mathbb{Q}(\sqrt{2}, \sqrt{3})$" --
{"$\mathbb{Q}(\sqrt{2})$", "$\mathbb{Q}(\sqrt{6})$",
"$\mathbb{Q}(\sqrt{3})$"};
{"$\mathbb{Q}(\sqrt{2})$", "$\mathbb{Q}(\sqrt{6})$",
"$\mathbb{Q}(\sqrt{3})$"}
-- "$\mathbb{Q}$";
};
\end{tikzpicture}
```



At this point, we could say we're done with this project: we have a Ti*k*Z picture that looks a lot like our hand-drawn field extension diagram.

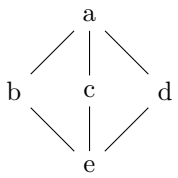
However, there are some more things to talk about. First off: although in this case Ti*k*Z made its picture just like we imagined it should be – with all of the layers like in our picture, and all of the nodes within each layer in the same order – this doesn't always happen. The algorithm behind Ti*k*Z's Layered Layout library might have its own ideas about what the "best" way to draw a diagram is. Sometimes it creates a graph that, while isomorphic to the one you drew by hand, looks quite different from what you had in mind. If this happens, sometimes there are tweaks you can make to "fix" Ti*k*Z's version of the graph. Sometimes, though, you basically just have to accept Ti*k*Z's version, or resign yourself to placing all of the vertices and edges by hand.

(Note: if you (1) are running LaTeX on your own machine, not through Overleaf, and (2) are a fairly confident programmer, then it's actually possible to alter or add to the Lua files that underlie Ti*k*Z's graph drawing algorithms. I don't think this is possible on Overleaf, and anyway it's not for the faint of heart, so I won't discuss this option here.)

To give an example of what I'm talking about, let's go back to our prototype diagram with its letters `a` through `e` at the nodes.

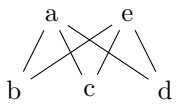
```
\begin{tikzpicture}
```

```
\graph [layered layout]{
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```



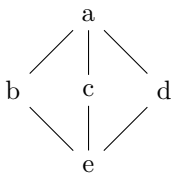
What if we'd written `e -- {b, c, d};` instead of `{b, c, d} -- e;`? This would still specify the same graph, up to isomorphism, but it makes a big difference in the appearance of our graph:

```
\begin{tikzpicture}
\graph [layered layout]{
a -- {b, c, d};
e -- {b, c, d};
};
\end{tikzpicture}
```



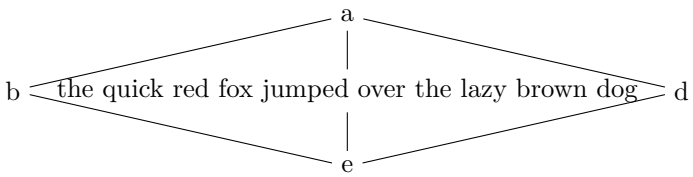
We've confused Ti*k*Z into thinking that nodes `a` and `e` should be on the same level. In general, if you put an edge `x -- y;` in your graph, it should be the case that `x` is on a higher level than `y`. However, it is possible to tell Ti*k*Z explicitly which nodes go together on a level using the command `{ [same layer]` $\cdots$ `};`

```
\begin{tikzpicture}
\graph [layered layout]{
a -- {b, c, d};
e -- {b, c, d};
{ [same layer] a };
{ [same layer] b, c, d };
{ [same layer] e };
};
\end{tikzpicture}
```



Ti*k*Z will automatically adjust horizontal and vertical spacing within the graph to accommodate especially long node text:

```
\begin{tikzpicture}
\graph [layered layout]{
a -- {b, "the quick red fox jumped over the lazy brown dog", d};
{b, "the quick red fox jumped over the lazy brown dog", d} -- e;
};
\end{tikzpicture}
```



However, it's also possible to make adjustments to the spacing by hand. *Note:* the following commands won't change how the Layered Layout groups and orders the layers. Also, in general, Ti*k*Z's graph drawing algorithms treat such "by-hand" adjustments as *suggestions*; they might not get honored in the graph that is produced. Finally, in general, Ti*k*Z's graph drawing algorithms already do a very good job of setting up graphs and diagrams to be nicely spaced and readable. It's best not to mess with its choices much unless you have a good reason.

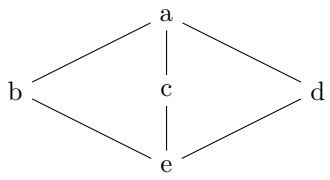
Adjustments

1. You can control how close together nodes on the same level are using the `sibling distance` argument to the `graph` command. (Nodes on the same level are called "sibliings".)
2. You can control the vertical space between levels using the `level distance` argument to the `graph` command.
3. You can orient the graph vertically using the command `vertical = x to y` where `x` and `y` are nodes that you want to place vertically relative to each other. The rest of the graph will then orient itself relative to `x` and `y`.
4. You can alter the position of a node using the `nudge` command.

The following variations on our prototype diagram illustrate how these adjustments work.

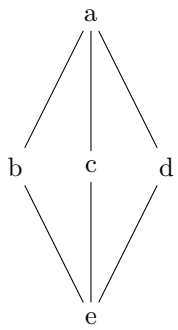
1. Change sibling distance:

```
\begin{tikzpicture}
% nodes on the same level are spaced farther apart
\graph [layered layout, sibling distance=2cm] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```



2. Change level distance:

```
\begin{tikzpicture}
% layers are spaced farther apart
\graph [layered layout, level distance=2cm] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```

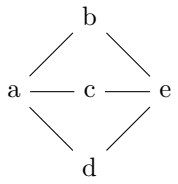


3. Orient the graph vertically …

   (a) … so that the orientation of the original graph flips:

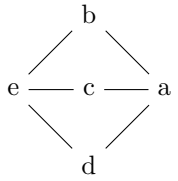
```
\begin{tikzpicture}
% graph is oriented vertically
% (levels are vertical)
\graph [layered layout, vertical = b to d] {
a -- {b, c, d};
{b, c, d} -- e;
};
```

```
\end{tikzpicture}
```



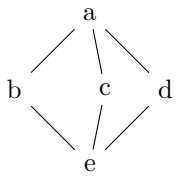
(b) … so that the orientation of the original graph is preserved:

```
\begin{tikzpicture}
% graph is oriented vertically
% (levels are vertical)
\graph [layered layout, vertical' = b to d] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```



4. Nudge a node:

```
\begin{tikzpicture}
% node e is nudged a bit off-center
\graph [layered layout]{
a -- {b, c, d};
{b, c, d} -- e[nudge=(left:3mm)];
};
\end{tikzpicture}
```

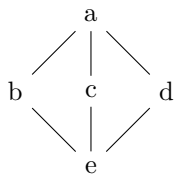


(There are more options for tweaking the graph produced by Ti*k*Z's graph drawing algorithms; see the PGF/TikZ Manual's Chapter 28: "Using Graph Drawing in TikZ" for details.)

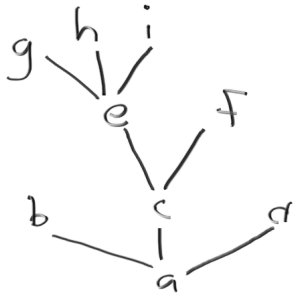
Exercises for Project 3

1. Alter our basic prototype diagram by increasing the spacing between nodes on the same level and decreasing the spacing between levels (by how much is up to you). Here, for reference, is that basic prototype diagram and the code that produced it:

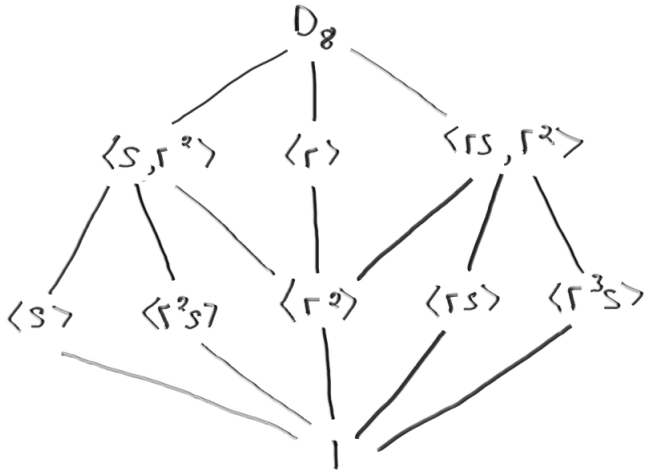
```
\begin{tikzpicture}
\graph [layered layout] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```



2. Create the following diagram in Ti*k*Z:

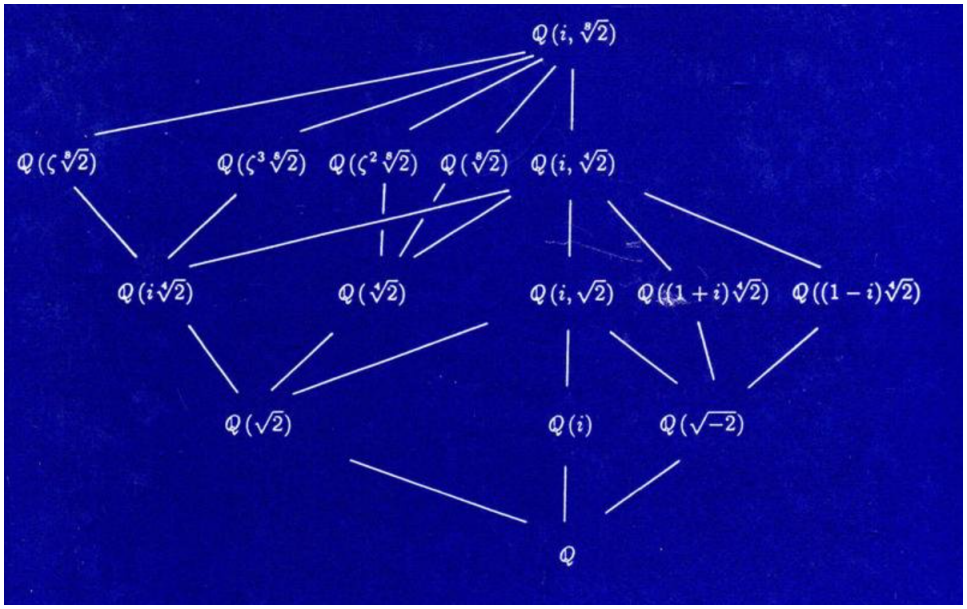


3. Create the following diagram in Ti*k*Z:



(This is the subgroup lattice of D_4, the dihedral group of symmetries of a square. r denotes a reflection, and s is a rotation by $\frac{\pi}{2}$.)

4. (Could be challenging) Recreate the following diagram, to the extent possible:



This is the diagram on the cover of the first edition of Dummit and Foote's *Abstract Algebra*. You may find it tricky to make Ti*k*Z reproduce the diagram exactly, but you should at least reproduce an isomorphic graph that preserves levels horizontally (though not necessarily the order of the

nodes within levels) and has no more than a few edge-crossings.

Note: Ti*k*Z's Layered Layout algorithm uses a heuristic to try and minimize edge crossings while keeping to the rule that edges go more or less directly to their destinations, without becoming too long or winding. It doesn't always produce a graph with the minimum possible number of edge crossings subject to this constraint, though.