Ti*k*Z tutorials

Jennifer Brown, California State University Channel Islands

April 19, 2024

This is a series of tutorials to show how to use Ti*k*Z to create professional-quality mathematical diagrams, graphs, and illustrations. The drawings in the projects will be most relevant to mathematics students and researchers, but the techniques are useful in any kind of technical illustration.

These tutorials are aimed at readers who have a basic knowledge of LaTeX but are beginners at Ti*k*Z, and who want to produce drawings that incorporate more than just the very basics (lines, circles, etc.). Each tutorial is a project with code blocks that can be run in Overleaf. I am assuming that the reader has access to either an Overleaf account or to LaTeX on their own machine. If you're new to LaTeX, then I highly suggest getting yourself an Overleaf account and working through the Learn LaTeX in 30 Minutes tutorial.

If you're comfortable with the material in these tutorials and want to learn more, the definitive resource for Ti*k*Z is the PGF/TikZ Manual. For more inspiration, a quick search will produce many examples of beautiful illustrations done with Ti*k*Z.

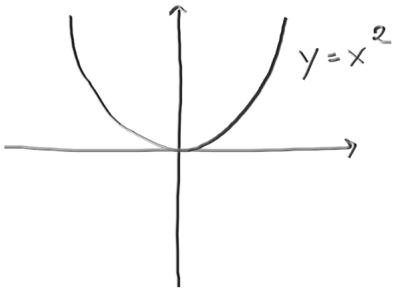
> **An important preliminary:** in your own Overleaf LaTeX document (where you'll be copying and modifying the code chunks in these projects), remember to put the command `\usepackage{tikz}` in your preamble – after the `\documentclass{article}` and before the `\begin{document}`.

The projects, and some things you'll learn by doing them

1. **Project 1: parabola.** The `tikzpicture` environment; the `draw` command for lines and arrows; the `plot` command; basic colors; fancy colors and the `xcolor` package; commenting your Ti*k*Z code.
2. **Project 2: bipartite graph.** Planning pictures on paper; helplines; using nodes for vertices of graphs and for text; the `shapes.geometric` library; Ti*k*Z styles.
3. **Project 3: a field extension diagram.** Ti*k*Z's layered layout graph drawing algorithm and the LuaLaTeX engine; placing text in nodes when using the graph drawing algorithms; "by-hand" adjustments to graphs.
4. **Project 4: inclined plane with apple.** Including, re-sizing, and rotating images in Ti*k*Z pictures; the `arrows.meta` library; coordinate transformations with `PGF`.

Project 1: parabola

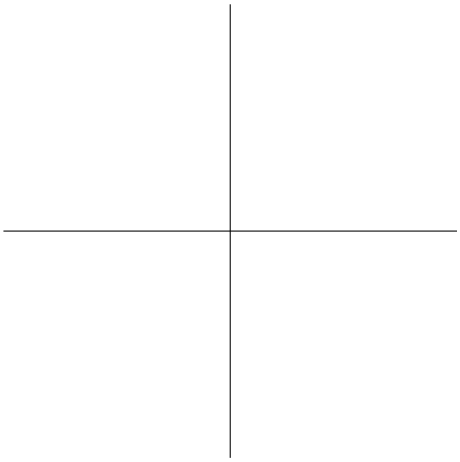
For our first project, we'll be using Ti*k*Z to make a nice version of this hand-drawn sketch of a parabola:



Let's start by making the x- and y-axes. The code block

```
\begin{tikzpicture}
\draw (-3,0) -- (3,0);
\draw (0,-3) -- (0,3);
\end{tikzpicture}
```

generates the picture below:



(The format in these tutorials will be to first display, verbatim, the code used to generate a picture, and then to display the resulting picture. You should copy and paste the code blocks into your own document to see how they work!)

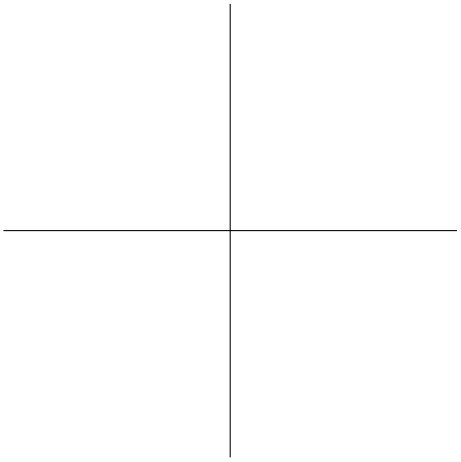
A few things to notice about that first block of code above:

- Ti*k*Z pictures need to be made inside the Ti*k*Z environment: `\begin{tikzpicture}` …`\end{tikzpicture}`.
- Commands in Ti*k*Z, such as the `\draw` command, are like commands in LaTeX – they begin with backslashes.
- The two dashes, `--`, mean that we are using the straight line path operation with the `\draw` command. (There are other path operations too; we'll see others later.)
- We specify the starting and ending points for each straight line by listing the points as ordered pairs.
- At the end of every line of Ti*k*Z commands, we have to put a semicolon.

Note that the picture above isn't centered on the page; it's put just at the place where a new paragraph would be put. If you want your picture to be centered (and you don't need to keep track of lots of pictures that might all need to be numbered as figures), you can just put it inside a displayed math environment (that is, inside `\[ …\]`), like this:

```
\[
\begin{tikzpicture}
\draw (-3,0) -- (3,0);
```

```
\draw (0,-3) -- (0,3);
\end{tikzpicture}
\]
```

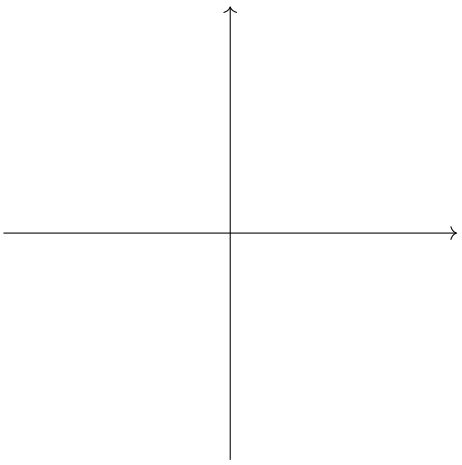


Notice that the `tikzpicture` environment works inside or outside of math mode. I'll do it outside of math mode for now.

> **An important aside:** Play around with these drawings and this code! Copy and paste the code blocks into your own LaTeX document, and then try messing with it. Change the colors, make a solid line dashed, scale your picture up or down, … If you don't know how to do something, check out the Ti*k*Z manual at `tikz.dev`, or see if someone has posted a similar question on StackOverflow or StackExchange. The very best way to learn Ti*k*Z (and many other things besides) is to take a basic model and mess around with it to see how it works. If you're worried that you might mess up an important document that you might have to turn in for a class, make yourself a "scratch" document and use it for experimentation.

Next, let's add some arrows to our axes. Ti*k*Z has lots of fancy arrows, but to do basic ones, just put `[->]` (or `[<-]`, or `[<->]`) after the basic `\draw` command. Let's say we just want arrows pointing up from the positive y-axis and right from the positive x-axis.

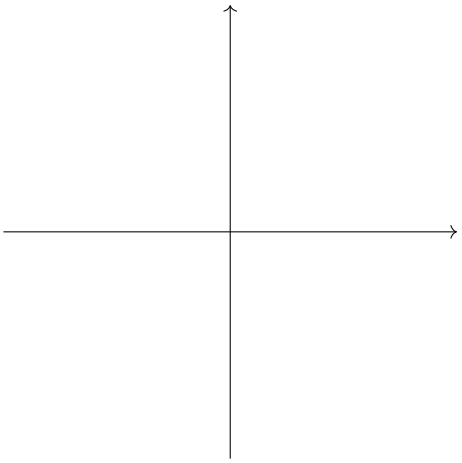
```
\begin{tikzpicture}
\draw[->] (-3,0) -- (3,0);
\draw[->] (0,-3) -- (0,3);
\end{tikzpicture}
```



Ti*k*Z is not very picky about spaces; the following code produces the same picture. (I'll usually use spacing in the example code just to make things easier to read, though.)

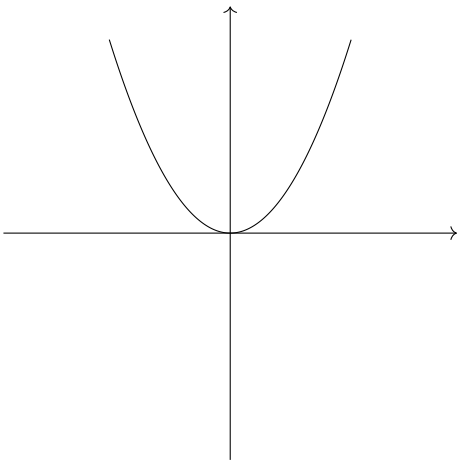
```
\begin{tikzpicture}
\draw[->](-3,0)--(3,0);
```

```
\draw[->](0,-3)--(0,3);
\end{tikzpicture}
```



Next, let's add the parabola $y = x^2$ to our picture.

```
\begin{tikzpicture}
\draw [->] (-3,0) -- (3,0);
\draw [->] (0,-3) -- (0,3);
\draw [smooth, samples=100, domain=-1.5:1.5] plot(\x, {(\x)^2)});
\end{tikzpicture}
```



There are a few new options in the `\draw` command here; let's look at each of them.

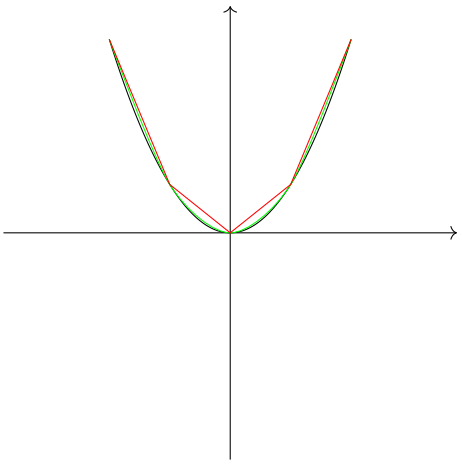
- In order to draw this parabola, Ti*k*Z is plotting a bunch of points (100, in this case) and joining them together. If you set the `sample` size to be small enough, you'll start to notice the graph getting choppy-looking.
- the `smooth` option means that the points get joined together with smooth curves, rather than straight lines.
- The `domain` option lets you set the smallest and largest x-value of plotted points. I set them to be $\pm$ 1.6 because that would make the y-values come nearly to the top of our axes (since $1.6^2 = 2.56$ and our y-axis goes to 3), but this is simply personal preference.

Next, the parabola: `plot(\x, {(\x)^2)})` tells Ti*k*Z that the x-values, denoted `\x`, should get squared in order to produce the y-values. The symbol `\x` should go inside round parentheses when you specify how y-values should get computed, and the entire y-value should go inside curly braces. Another example: if, instead of $y = x^2$, we wanted to plot $y = 3x^3 - 4x$, we would replace `plot(\x, {(\x)^2)})` with `plot(\x, {3*(\x)^3 - 4*(\x)})`

This next graph isn't really part of the picture we set out to draw in this first project, but it shows

the effect of shrinking the `sample` size and using the default (straight-line) connections between points. It also demonstrates how to plot multiple graphs in the same picture, and how to change colors as an option to the `draw` command.

```
\begin{tikzpicture}
\draw [->] (-3,0) -- (3,0);
\draw [->] (0,-3) -- (0,3);
\draw [smooth, samples=100, domain=-1.6:1.6] plot(\x, {(\x)^2});
\draw [green, smooth, samples=5, domain=-1.6:1.6] plot(\x, {(\x)^2});
\draw [red, samples=5, domain=-1.6:1.6] plot(\x, {(\x)^2});
\end{tikzpicture}
```



An aside about colors: base LaTeX as used in Overleaf has basic colors, like red and brown, built-in. If you want access to a bunch of additional fancy colors, you can use the `xcolor` package (together with an option). For example, by putting `\usepackage[dvipsnames]{xcolor}` in the preamble, you get 68 more named colors, such as BrickRed and RawSienna.

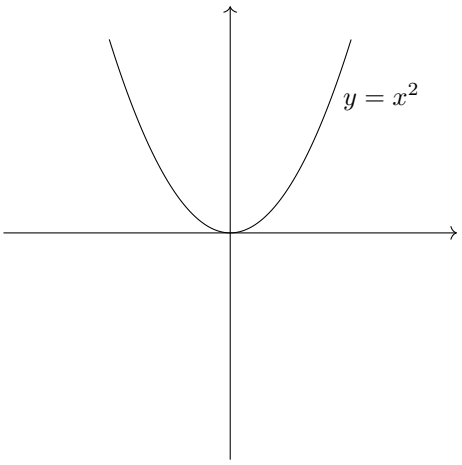
Important: you need to load the `xcolor` package (with its option) *before* you load `tikz`! For example, to load `xcolor` with the `dvipsnames` option, you need to write

```
\usepackage[dvipsnames]{xcolor}
\usepackage{tikz}
```

in that order. See Overleaf's page on "Using colours in LaTeX" for details about the `xcolor` package. For even fancier colors and more control over shading (using varying saturation, for example), you can use the `xxcolor` package. See the PGF/Ti*k*Z Manual's page on "Extended Color Support" for details.

Getting back to our basic picture of the parabola $y = x^2$: the next thing we'd like to do is to add a label to it. We'll do this by adding a node to our picture. A node, in Ti*k*Z, is a specified location where things like labels or shapes can go. In this case, I put the text $y = x^2$, in math mode, centered at the point (2.0,1.8) in the plane. (Again, this was just personal preference – that location looked about right to me. You should play around with the location of the label until it suits you.)

```
\begin{tikzpicture}
\draw [->] (-3,0) -- (3,0);
\draw [->] (0,-3) -- (0,3);
\draw [smooth, samples=100, domain=-1.6:1.6] plot(\x, {(\x)^2});
\node at (2.0,1.8) {$y=x^2$};
\end{tikzpicture}
```



Note that the math mode text in the node needs to be enclosed in dollar signs as usual, and also that the entire text at the node needs to be enclosed in curly braces.

Almost done! There is one more thing you should do, and it's not going to show up in your picture, but it is nonetheless very important: *Comment your TikZ code.* This is part of best practice when you're doing any kind of coding. You need to explain what you're doing with each piece of code so that your collaborators, or you six months from now, know what's going on in case they, or you, need to modify or debug something.

Commenting in the `tikzpicture` environment is done the way it's usually done in LaTeX, with a `%` sign. Here is how I might comment our parabola picture:

```
% a picture of the parabola y=x^2
\begin{tikzpicture}
% the x-axis, with an arrow pointing right
\draw [->] (-3,0) -- (3,0);

% the y-axis, with an arrow pointing up
\draw [->] (0,-3) -- (0,3);

% the parabola y=x^2, from -1.6 to 1.6,
% using sample size = 100 points
% and joining points smoothly
\draw [smooth, samples=100, domain=-1.6:1.6] plot(\x, {(\x)^2)});

% the label y=x^2
\node at (2.0,1.8) {$y=x^2$};
\end{tikzpicture}
```

If you've made it this far, congratulations! Now go back, if you haven't already, and play around with the code blocks (copy and paste them into your own Overleaf document) to get an idea of how the commands you've seen in this first project work.

Then, when you're ready, test your understanding by completing the following exercises. **NOTE:** There is no one "correct" solution for any of these exercises. The idea is to make a Ti*k*Z picture that does what the exercise says it should, and you need to be the judge of whether or not your picture "works". The exercises for a particular project will always be doable using the commands and options introduced in that project or previous ones, but there might be other (possibly better) ways to do them too. If you're not satisfied with a picture, it's OK to set it aside for now, and come back to it as you learn more.

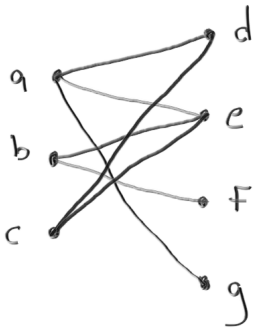
Exercises for Project 1

1. Make a square with its corners at $(-1, 1)$, $(1, 1)$, $(1, -1)$, and $(-1, -1)$.

2. Make a yellow diamond with its corners at $(0, 1)$, $(1, 0)$, $(0, -1)$, and $(-1, -0)$. Center the picture on the page horizontally.
3. Make a picture similar to our final parabola picture, except change the function to $y = -x^3 + 2x - 5$. Adjust the axes as needed. Include a label with the equation of the function.
4. Make a picture showing the graphs of the functions $y = x^2 - 2$ and $y = -x^2 + 1$ on the same set of axes. Color the $y = x^2 - 2$ graph red and the $y = -x^2 + 1$ graph blue. Include a label for each graph showing its equation.

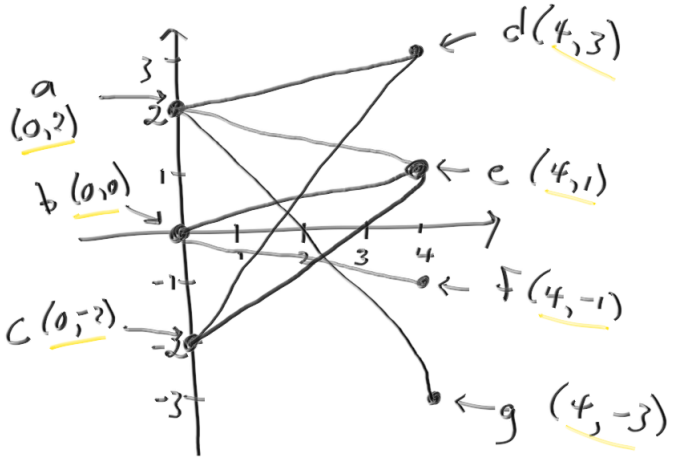
Project 2: bipartite graph

For this project, when I mention "graphs", I'm talking about the sort you'd see in a discrete math class, with vertices and edges, rather than the sort you'd see in a calculus class. We're going to use Ti*k*Z to draw the following bipartite graph:



Ti*k*Z has several different ways of making graphs. The one you will see in this project is the most intuitive: we will simply tell Ti*k*Z where to put vertex dots and edge lines by specifying coordinates in the plane. (In a later project, we'll see another way to draw graphs, using Ti*k*Z's graph-drawing algorithms and a different LaTeX engine; it's more powerful but less intuitive.)

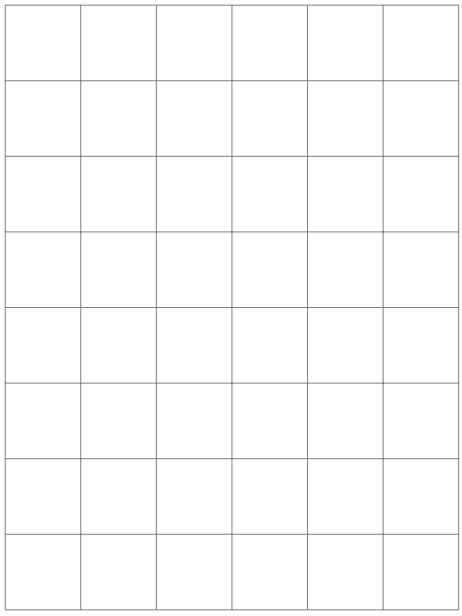
In this method, you need to some paper-and-pencil work up front in order to figure out where, on the plane, the vertices and edges of your graph should be located. As a rule of thumb, try to make your scale such that you're dealing with integer values fairly near the origin, like $(4, 2)$, rather than decimal values far from the origin, like $(572.0043, -82.009)$. This is just to make life easier for yourself; it's not a hard-and-fast requirement. You can always re-scale your picture to fit it where it needs to go, and you can also fine-tune the placement of elements of your picture after you've got the basic picture done.



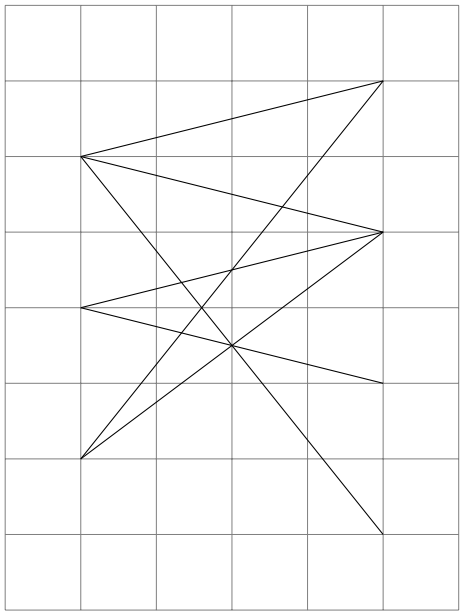
When you are making pictures by specifying coordinates for where things go, it's helpful to have Ti*k*Z draw some gridlines so that you can see if you're placing elements as you specified in your hand-drawn

picture. To do this, you use `help lines` as an argument to the `\draw` function, and use the `grid` path operation. Ti*k*Z will draw a grid (at integer values) in a rectangle whose lower left corner is the first point and whose upper right corner is the second point. I chose to have a grid containing all of the x- and y-values that I will need in my picture, plus a little wiggle room for adding labels.

```
\begin{tikzpicture}
\draw [help lines] (-1,-4) grid (5,4);
\end{tikzpicture}
```



Next, let's add the edges, since we've already seen how to do that in Project 1: we use the `\draw` command. For example, to draw the edge that goes from $(0, 2)$ to $(4, 3)$, we'd write `\draw (0,2) -- (4,3);`.



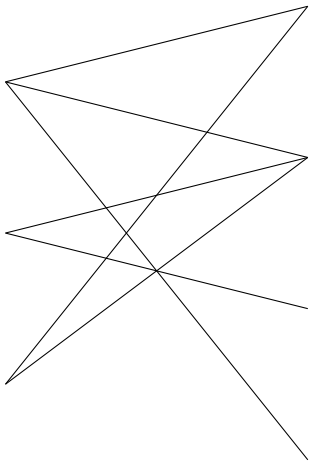
Remember to comment your code! The little note

```
% help lines (comment out when done)
```

reminds me that I should put a `%` sign in front of the help lines command when I'm satisfied with my picture. If we do that right now, the picture looks like this:

```
\begin{tikzpicture}
```

```
% help lines (comment out when done)
% \draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

\end{tikzpicture}
```



We'll put the help lines back for now. The next element we need is little dots at the vertices of our graph. The little dots will be drawn at a *node*. A node in Ti*k*Z is a thing placed at a particular place. That's a pretty broad description, but a node is a very flexible object: it can contain some text, an equation in math mode, a shape, or some complicated combination of these. For our purposes, we want to put nodes at each location where a vertex should be, and put a little solid black circle at each of these nodes:

```
\node [circle, fill, inner sep=2pt] at (0,2) {};
```

This would place a little dot at $(0, 2)$, where vertex a is. We'll see a bit later in this project what the various arguments to the node mean, and how they can be modified to adjust the appearance of the vertices in the picture.

(A Technical Aside: we're using a little trick here. Strictly speaking, a node in Ti*k*Z is a specification of a place to put text, but you can also specify a *background* for the text. If your "background" is the shape that you actually want to position at the node, you can make the text empty so that only the background – that is, the shape – shows up. This is what we'll do to draw the vertices.)

Before we start placing the vertices in the graph, let's see how to modify the size of the little dots. In the picture below, the first dot is the one whose code I write above:

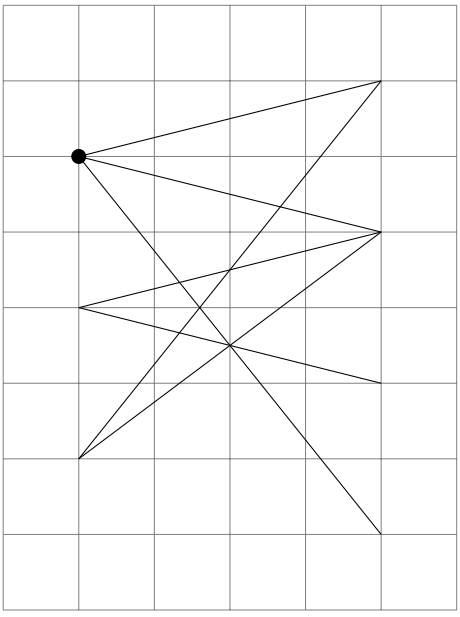
```
\node [circle, fill, inner sep=2pt] at (0,2) {};
```

The first two arguments to the `node` command, `circle` and `fill`, tell Ti*k*Z to make a filled-in circle. The default color is black. The next argument, `inner sep=2pt`, tells Ti*k*Z how big to make the dot. (To be precise, it tells it how much space to put between the text – which is empty – and the edge of the dot.) Finally, the ordered pair (this is $(0, 2)$ in our example) tells Ti*k*Z where to put the node; and the empty brackets `{}` tells it that the spot that would ordinarily be occupied by text is to be empty (see the Technical Aside above).

For comparison, below are some dots and dot-like things that can be drawn by varying the basic dot shape above. Ti*k*Z has two built-in shapes for nodes: `circle` and `rectangle`. To get additional shapes, we can load a library called `shapes.geometric` using the command `\usepgflibrary{shapes.geometric}`. Some of our dots are in fancy colors. Back in the preamble, we put `\usepackage[dvipsnames]{xcolor}` to load the `dvipsnames` colors from the `xcolor` package; and we loaded this package *before* we loaded Ti*k*Z, as discussed in Project 1.

Once we're satisfied with our vertex shape, we could place it at vertex a, like this:

```
\begin{tikzpicture}

% help lines (comment out when done)
\draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [circle, fill, inner sep=2pt] at (0,2) {};

\end{tikzpicture}
```



Then we could repeat the command

```
\node [circle, fill, inner sep=2pt] at (0,2) {};
```

for the six other vertices in our graph (changing $(0, 2)$ to the appropriate coordinate). – But then what if, later, we decide that we wanted that little circle to be a bit bigger? or smaller? or purple? We would

then need to go back and alter seven different occurrences of this command in order to make the desired changes. This is a recipe for frustration and error.

Enter Ti*k*Z styles. A *style* is a series of arguments – in this case, to a node – that you name and declare either in the preamble, or just at the beginning of your current picture. In this project, the style for the vertices is going to be declared at the beginning of each picture – right after the beginning of the `tikzpicture` environment. However, if you're going to be using a style in multiple places throughout a document, it's best to declare it in the preamble (and to make a comment at the beginning of each picture where it's used, saying that the style is used and what it's called). I'm going to declare a style called `vertex`:

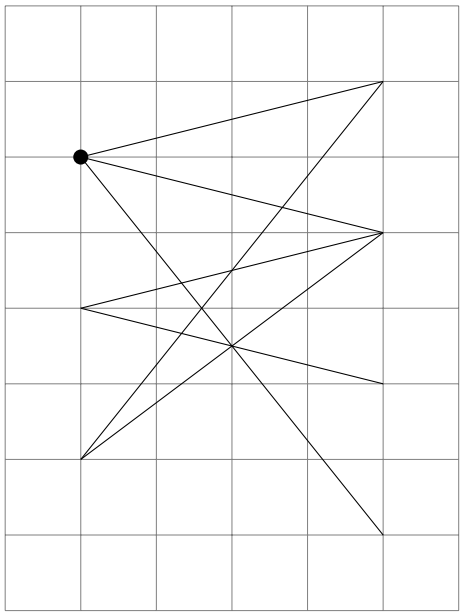
```
\begin{center}
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]
\end{center}
```

(Note that declaring a style, just by itself, doesn't produce any output.) This command is simply taking all of the arguments we had inside the `node` command and packaging them inside a new argument called `vertex`. Now we're going to declare our `vertex` style at the beginning of the picture, delete the

```
\node [circle, fill, inner sep=2pt] at (0,2) {};
```

command, and use the `vertex` style instead to put a vertex at a.

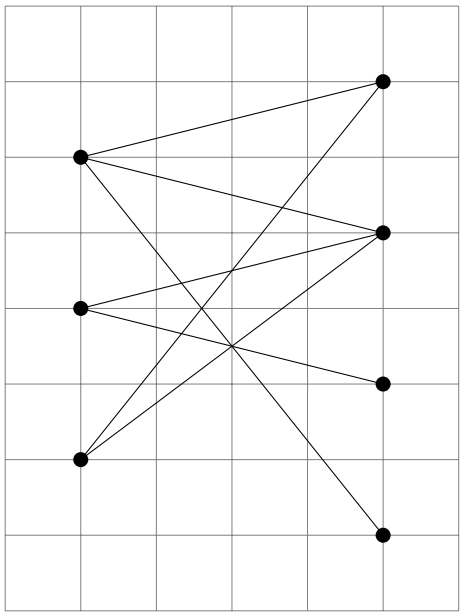
```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
\draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};

\end{tikzpicture}
```



Now we just put a node in the `vertex` style at all of the other vertices, adjusting the coordinates as appropriate.

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
\draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};

% dot at vertex b
\node [vertex] at (0,0) {};

% dot at vertex c
\node [vertex] at (0,-2) {};

% dot at vertex d
\node [vertex] at (4,3) {};

% dot at vertex e
\node [vertex] at (4,1) {};
```

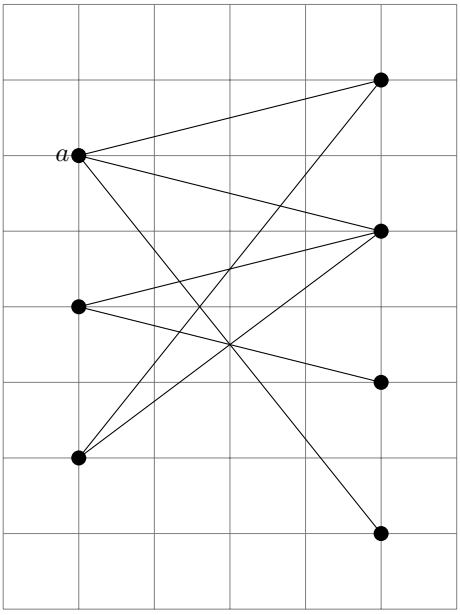
```
% dot at vertex f
\node [vertex] at (4,-1) {};

% dot at vertex g
\node [vertex] at (4,-3) {};

\end{tikzpicture}
```



Our graph is looking almost complete! The only thing we're missing is the labels a, b, etc. on the vertices. The simplest way to do this is to put a node with the vertex label at the same coordinates where the little dots went – only offset, either to the left or the right. For example, to get the letter a (in math mode) to appear just to the left of where the vertex dot for a is, we can add the command `\node [left] at (0,2) {$a$};`. (To make it easy for me to find, I've put that command right below where we put the command for the vertex at a.)

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
\draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};
```

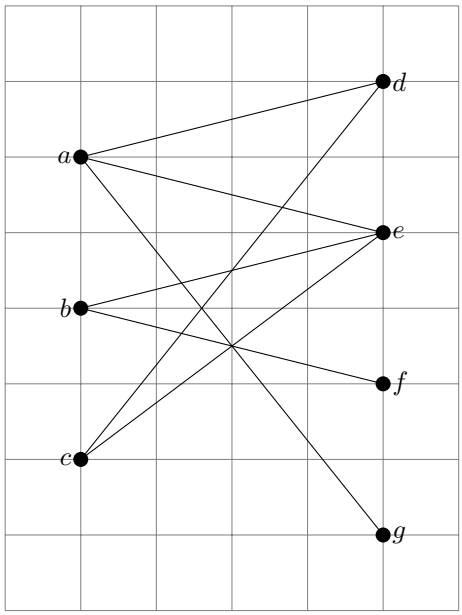
```
% label at vertex x
\node [left] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};

% dot at vertex c
\node [vertex] at (0,-2) {};

% dot at vertex d
\node [vertex] at (4,3) {};

% dot at vertex e
\node [vertex] at (4,1) {};

% dot at vertex f
\node [vertex] at (4,-1) {};

% dot at vertex g
\node [vertex] at (4,-3) {};

\end{tikzpicture}
```



If we are satisfied about where the label is going, we can do the same thing for all of the vertex labels – that is, place them at the same node where the vertex dot went, only offset to the right or the left. In order that the labels don't get mixed up with the edges, I'm putting the labels a, b, and c to the left of their respective vertices; and I'm putting the labels d, e, f, and g to the right.

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
\draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
```

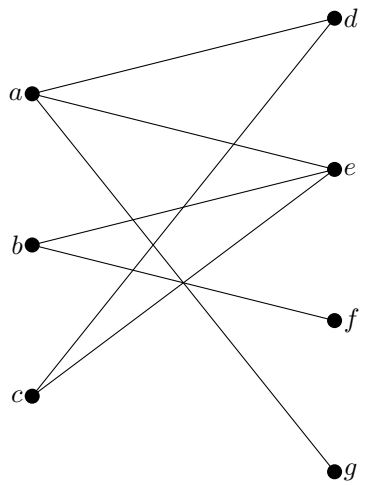
```
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a
\node [left] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};
% label at vertex b
\node [left] at (0,0) {$b$};

% dot at vertex c
\node [vertex] at (0,-2) {};
% label at vertex a
\node [left] at (0,-2) {$c$};

% dot at vertex d
\node [vertex] at (4,3) {};
% label at vertex d
\node [right] at (4,3) {$d$};

% dot at vertex e
\node [vertex] at (4,1) {};
% label at vertex e
\node [right] at (4,1) {$e$};

% dot at vertex f
\node [vertex] at (4,-1) {};
% label at vertex f
\node [right] at (4,-1) {$f$};

% dot at vertex g
\node [vertex] at (4,-3) {};
% label at vertex g
\node [right] at (4,-3) {$g$};

\end{tikzpicture}
```



Let's see what our picture looks like without the help lines. (Remember the comment `% help lines (comment out when done)`? Let's do that.)

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
% \draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a
\node [left] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};
% label at vertex b
\node [left] at (0,0) {$b$};

% dot at vertex c
\node [vertex] at (0,-2) {};
% label at vertex a
\node [left] at (0,-2) {$c$};
```

```
% dot at vertex d
\node [vertex] at (4,3) {};
% label at vertex d
\node [right] at (4,3) {$d$};

% dot at vertex e
\node [vertex] at (4,1) {};
% label at vertex e
\node [right] at (4,1) {$e$};

% dot at vertex f
\node [vertex] at (4,-1) {};
% label at vertex f
\node [right] at (4,-1) {$f$};

% dot at vertex g
\node [vertex] at (4,-3) {};
% label at vertex g
\node [right] at (4,-3) {$g$};

\end{tikzpicture}
```



Now for a little fine-tuning. I don't know about you, but to me, the labels seem just a bit too close to the vertices they're labeling. (You might disagree, but bear with me.) There is an easy way to adjust "by hand" the placement of text around nodes: we add an *offset*. When we write

```
\node [left] at (0,2) {$a$};
```

the offset of the `left` position is 0pt by default; it would be the same if we wrote

```
\node [left=0pt] at (0,2) {$a$};
```

Now let's do a few little Ti*k*Z pictures to see how just this one vertex with its one label would look if we adjust the offset a little. Don't forget to include our `vertex` style, or else the code blocks below won't compile.

Offset = 0pt:

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]
```

```
% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a, offset = 0
\node [left=0pt] at (0,2) {$a$};

\end{tikzpicture}
```


Offset = 1pt:

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a, offset = 1
\node [left=1pt] at (0,2) {$a$};

\end{tikzpicture}
```


Offset = 2pt:

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a, offset = 2
\node [left=2pt] at (0,2) {$a$};

\end{tikzpicture}
```

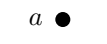

Offset = 3pt:

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a, offset = 3
\node [left=3pt] at (0,2) {$a$};

\end{tikzpicture}
```


Offset = 4pt:

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]
```

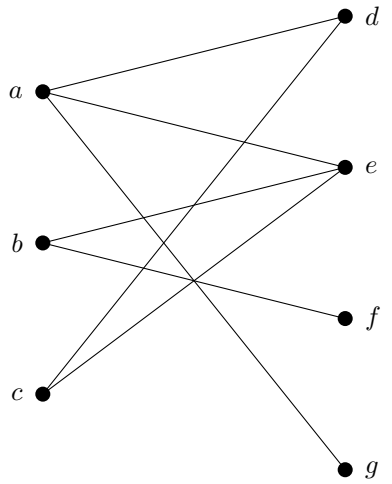
```
% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a, offset = 4
\node [left=4pt] at (0,2) {$a$};

\end{tikzpicture}
```



To me, the offset by 4pt looks the best. If you disagree, feel free to alter it. Now just insert that `=4pt` offset after each `left` or `right` argument to the vertex label nodes. The offset is not an absolute direction; it's just a distance that modifies (in this case) either the `left` or the `right` argument to the node. This means that we don't have to do anything different with our labels that are to the right of their vertices than we do for those that are placed to the left of their vertices.

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
% \draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a
\node [left=4pt] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};
% label at vertex b
\node [left=4pt] at (0,0) {$b$};

% dot at vertex c
\node [vertex] at (0,-2) {};
% label at vertex a
\node [left=4pt] at (0,-2) {$c$};

% dot at vertex d
\node [vertex] at (4,3) {};
% label at vertex d
\node [right=4pt] at (4,3) {$d$};

% dot at vertex e
\node [vertex] at (4,1) {};
```

```
% label at vertex e
\node [right=4pt] at (4,1) {$e$};

% dot at vertex f
\node [vertex] at (4,-1) {};
% label at vertex f
\node [right=4pt] at (4,-1) {$f$};

% dot at vertex g
\node [vertex] at (4,-3) {};
% label at vertex g
\node [right=4pt] at (4,-3) {$g$};

\end{tikzpicture}
```



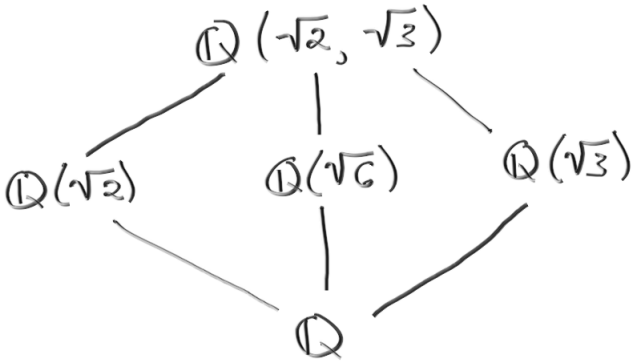
I think this looks like our original hand-drawn picture! Feel free to go back and do some of your own fine-tuning. When you're ready, test your understanding with these exercises.

Exercises for Project 2

1. Make a grid of help lines with the lower left corner of the rectangular grid at $(-2, -2)$ and the upper right corner at $(2, 2)$. Using the `vertex` style developed in this project (modified if you like), put a dot at each non-origin intersection point of the help lines. Put a noticeably different sort of dot at the origin.
2. Modify the final graph picture in this project so that there is an additional edge from a to f, but no edge from b to e.
3. Modify the final graph picture in this project by adding an additional vertex h at $(4, -5)$, and give it edges to c and b.
4. Modify the final graph picture in this project by defining and using a new vertex style, called `vertex2`, that makes dots that are about half the size as the ones in the `vertex` style and are also green. (If you'd like to use something fancier than `green`, see the "Using colors in LaTeX" page in the Overleaf help documentation for a list of available `xcolor` colors. Be sure you've loaded the appropriate `xcolor` option before loading Ti*k*Z in your preamble.)
5. Draw a picture of $K_{3,3}$, the complete bipartite graph from 3 vertices to 3 vertices, using ingredients from the code in the block for our final graph picture in this project.

Project 3: field extension diagram

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 the LuaLaTeX engine. If you try to compile any of the pictures in this project with the pdfLaTeX engine, it may or may not throw an error, but the pictures will definitely not look right.

Switching LaTeX engines 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".)

If you are using a LaTeX editor on your own machine, you will also need to switch compilers, but there will be a different way to do it. For example, in TeXstudio, you put into your preamble the following "magic comment" (*with the % sign!*):

```
% !TeX program = lualatex
```

If you're using a different editor, there may be a different magic comment.

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:

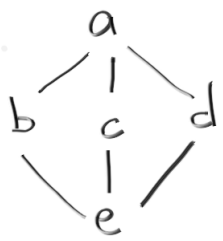
```
\pgfplotsset{compat=1.18}
```

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

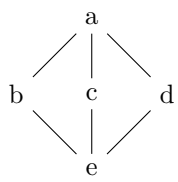
```
a -- {b, c, d};
```

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;
```

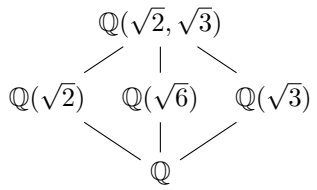
That's it – there are no more levels, and we've told Ti*k*Z about all of the nodes and edges. Let's put these two commands, `a -- {b, c, d};` and `{b, c, d} -- e;`, into our graph:

```
\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 `" ··· "`. 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}
```



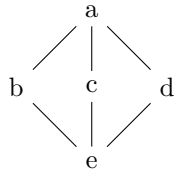
At this point, we could say we're done with this project: we have a Ti*k*Z picture that looks a lot like our hand-drawn field extension diagram.

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.)

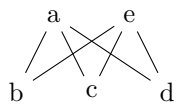
To give an example of what I'm talking about, let's go back to our prototype diagram with its letters `a` through `e` at the nodes.

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



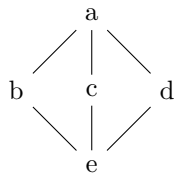
What if we'd written `e -- {b, c, d};` instead of `{b, c, d} -- e;`? This would still specify the same graph, up to isomorphism, but it makes a big difference in the appearance of our graph:

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



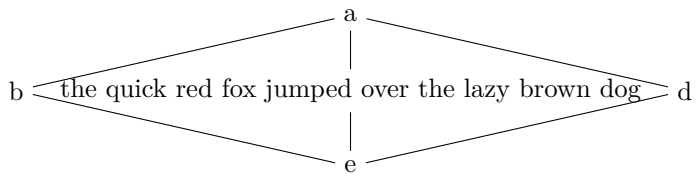
We've confused Ti*k*Z into thinking that nodes `a` and `e` should be on the same level. In general, if you put an edge `x -- y;` in your graph, it should be the case that `x` is on a higher level than `y`. However, it is possible to tell Ti*k*Z explicitly which nodes go together on a level using the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

```
\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

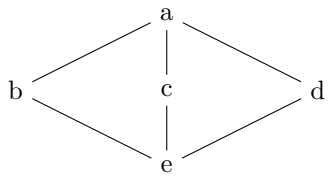
1. You can control how close together nodes on the same level are using the `sibling distance` argument to the `graph` command. (Nodes on the same level are called "sibliings".)

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.

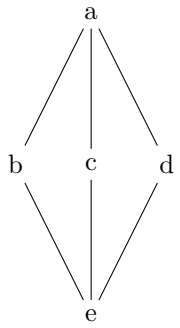
1. Change sibling distance:

```
\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}
```



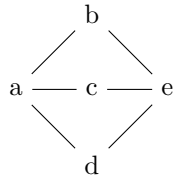
2. Change level distance:

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



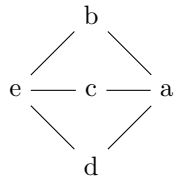
3. Orient the graph vertically …
   (a) … so that the orientation of the original graph flips:

```
\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}
```



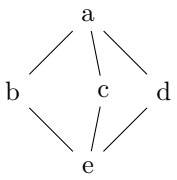
(b) … so that the orientation of the original graph is preserved:

```
\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}
```

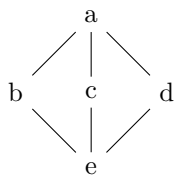


(There are more options for tweaking the graph produced by Ti*k*Z's graph drawing algorithms; see the PGF/TikZ Manual's Chapter 28: "Using Graph Drawing in TikZ" for details.)

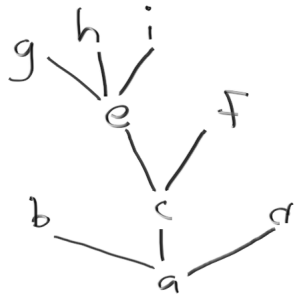
Exercises for Project 3

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:

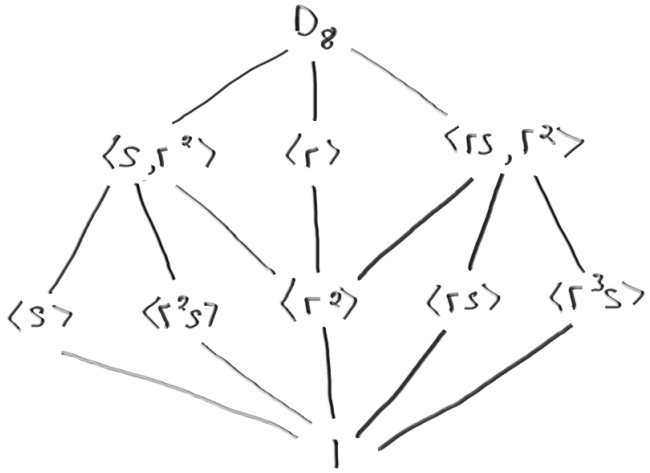
```
\begin{tikzpicture}
\graph [layered layout] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```



2. Create the following diagram in TikZ:

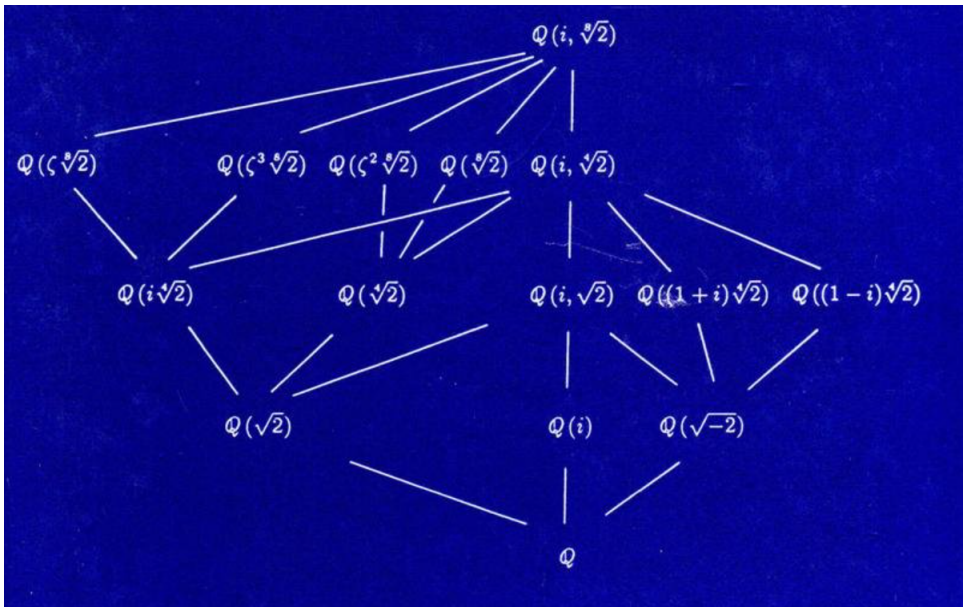


3. Create the following diagram in TikZ:



(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:

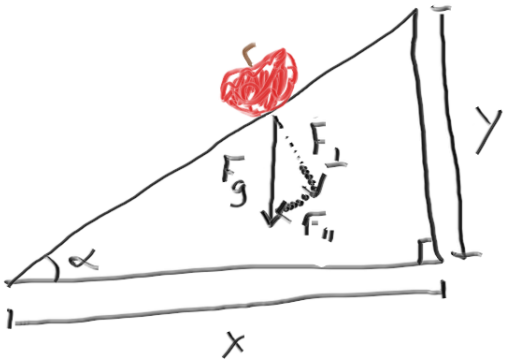


This is the diagram on the cover of the first edition of Dummit and Foote's *Abstract Algebra*. You may find it tricky to make Ti*k*Z reproduce the diagram exactly, but you should at least reproduce an isomorphic graph that preserves levels horizontally (though not necessarily the order of the nodes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

Project 4: inclined plane diagram with apple

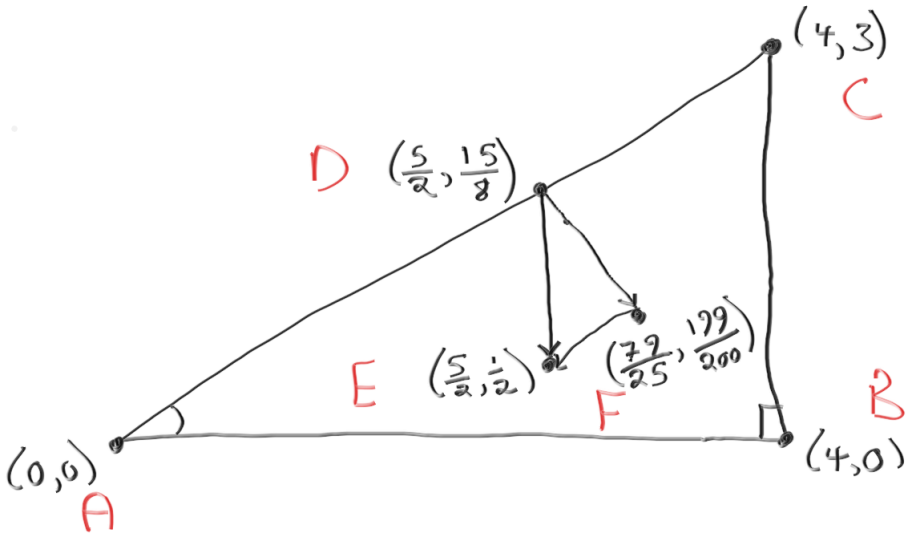
In this project, we are going to draw this free-body diagram of an apple sliding down an inclined plane:



… only we're going to include this photograph of an actual apple:

I found a photograph of an apple on an almost-white background and cropped it a bit so that it will appear to be resting on its base on the inclined plane later.

First, as we did in Project 2, we mark the (x, y) coordinates of the important pieces of the picture:



Some paper-and-pencil (and calculator) work was required to find these coordinates:

- I decided to use a 3-4-5 right triangle, oriented as in my hand-drawn picture and with vertex A at the origin. That gave me points B and C.
- Then I decided to put the apple so that its x-coordinate was 2.5, or $\frac{5}{2}$. Since the line joining points A and C is $y = \frac{3}{4}x$, this meant that the y-coordinate of point D is $\frac{3}{4} \cdot \frac{5}{2} = \frac{15}{8}$. **Note:** Ti*k*Z doesn't care whether you enter coordinates as decimals or fractions: putting the coordinates of point D as `(2.5, 1.875)` is fine, and so is `(5/2, 15/8)`.
- I decided that point E should be at height $\frac{1}{2}$, and it should be on the same vertical line as point D, so its coordinates are $(\frac{5}{2}, \frac{1}{2})$.
- To find the coordinates of point F, I first found the equation of the line through $D(\frac{5}{2}, \frac{15}{8})$ perpendicular to the line $y = \frac{3}{4}x$. This turns out to be $y = -\frac{4}{3}x + \frac{125}{24}$. Then I found the line through $E(\frac{5}{2}, \frac{1}{2})$ parallel to the line $y = \frac{3}{4}x$. This came out to be $y = \frac{3}{4}x - \frac{11}{8}$. Then I found the point of intersection of these two lines. This was $(\frac{79}{25}, \frac{199}{200})$, so that was where point F went.

First let's draw the big triangle, the horizontal and vertical measuring bars, and the two little angle markers.

We draw the big triangle with the `\draw` command, using the argument `thick` to make the sides of the triangle a bit thicker than the default thickness. Note that we can draw all three sides of the big triangle in a single `\draw` command:

```
\draw [thick] (0,0) -- (4,0) -- (4,3) -- (0,0);
```

accomplishes the same thing as

```
\draw [thick] (0,0) -- (4,0);
\draw [thick] (4,0) -- (4,3);
\draw [thick] (4,3) -- (0,0);
```

We do a similar thing for the little right triangle corner, only we don't make the lines thick:

```
\draw (3.8,0) -- (3.8, .2) -- (4,.2);
```

A few words about the horizontal and vertical measuring bars for x and y:

- We drew a long line segment for each of these measuring bars, rather than drawing two short segments for each. Later, we'll place nodes with x and y there, and give each of them a white background.

- Rather than draw the tiny endcaps for the x and y line segments "by hand", we used an arrow tip called `Bar` from the `arrows.meta` library, so we need to load this library first. (You could also include the command `\usetikzlibrary{arrows.meta}` in the preamble.) This library has **lots** of fancy, customizable arrow tips. See Section 16.5, Reference: Arrow Tips, from the Ti*k*Z Manual for details.

  Note the syntax for using a fancy arrow tip:

  `\draw [-{Bar}] (0,0) -- (1,0);` produces ⟝

  `\draw [{Bar}-] (0,0) -- (1,0);` produces ⟞

  `\draw [{Bar}-{Bar}] (0,0) -- (1,0);` produces ⟝⟞

- We drew the x and y measurement bars at a distance of 0.3 from the horizontal and vertical legs, respectively, of the big triangle, because … it looked like the right distance. You should adjust your measurement bars (and other details of the picture) to suit your taste.

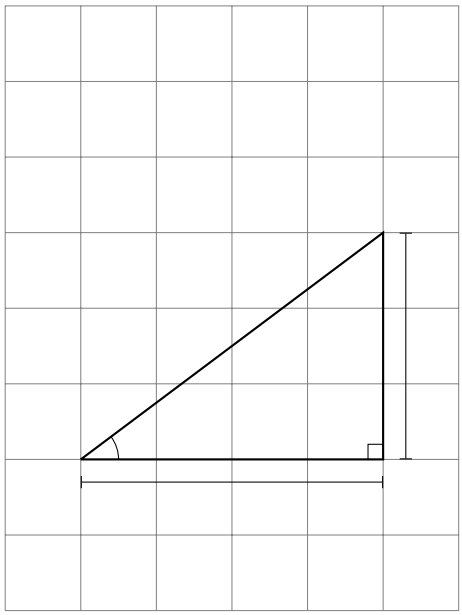
To draw the little arc for the acute angle α, we first had to do a bit of trig: $\tan(\alpha) = \frac{3}{4}$, and α is an acute angle, so $\alpha \approx 36.87°$. Once we know this, we use the `arc` path in the `draw` command. The following command tells Ti*k*Z to sweep out an arc of a circle with radius 0.5 centered at (0,0) from the starting angle of 0 to the ending angle of 36.87°:

```
\draw[radius=0.5 cm] (0.5,0) arc [start angle=0, end angle=36.87];
```

Notice that this command did not actually specify that the center of the circle from which the arc came is (0,0). This is implied: the point (0.5,0) is where the arc begins; the starting angle is 0; and the radius is 0.5 cm. Then to get from the starting point to the center of the circle, we'd go "backwards" a distance of 0.5 cm, putting us at (0,0).

As we did in Project 2, we'll have Ti*k*Z draw a grid to help us make sure we're placing the components of the picture where we intend to place them.

```
\begin{tikzpicture}
% load the arrows library
\usetikzlibrary{arrows.meta}

% help lines (comment out when done)
\draw [help lines] (-1,-2) grid (5,6);

% the triangle for the inclined plane
\draw [thick] (0,0) -- (4,0) -- (4,3) -- (0,0);

% the little right triangle corner
\draw (3.8,0) -- (3.8, .2) -- (4,.2);

% the x measuring bar
\draw [{Bar}-{Bar}] (0,-0.3) --  (4,-0.3);

% the y measuring bar
\draw [{Bar}-{Bar}] (4.3,0) --  (4.3,3);

% the little arc for the angle
\draw [radius = 0.5 cm] (0.5,0) arc
[start angle=0, end angle =  36.87];

\end{tikzpicture}
```

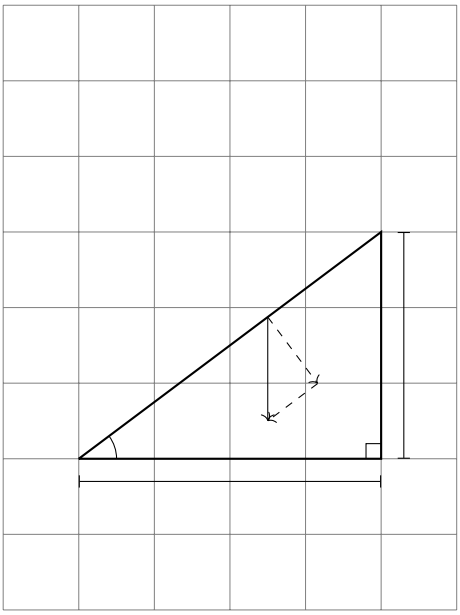


Next we'll draw the force arrows: the downwards arrow for the force of gravity on the apple, and the dotted lines for the components of the gravity force parallel and perpendicular to the inclined plane. (The apple itself we'll leave for last, and we can always re-size it.) I decided to have the downwards (gravity) force arrow go from the apple straight down to where the x-coordinate is 0.5.

To draw the dotted lines, we first needed to do a bit of math. Since the apple will be at $(\frac{5}{2}, \frac{15}{8})$, this vertical arrow will go from $(\frac{5}{2}, \frac{15}{8})$ to $(\frac{5}{2}, \frac{1}{2})$. The line parallel to the inclined plane is then going to have slope $\frac{3}{4}$ and contain the point $(\frac{5}{2}, \frac{1}{2})$, so its equation is $y = \frac{3}{4}x - \frac{11}{8}$. The line from the apple perpendicular to the inclined plane must have slope $-\frac{4}{3}$ and contain the point $(\frac{5}{2}, \frac{15}{8})$, so its equation is $y = -\frac{4}{3}x + \frac{125}{24}$. Finally, the point where these two lines intersect is $(\frac{79}{25}, \frac{199}{200})$, and this is found by setting $\frac{3}{4}x - \frac{11}{8} = -\frac{4}{3}x + \frac{125}{24}$. Notice that we can leave the coordinates of the endpoints of our lines as fractions if we like; TikZ will do the calculation when it draws the line.

```
\begin{tikzpicture}
% load the arrows library
\usetikzlibrary{arrows.meta}

% help lines (comment out when done)
\draw [help lines] (-1,-2) grid (5,6);

% the triangle for the inclined plane
\draw [thick] (0,0) -- (4,0) -- (4,3) -- (0,0);

% the little right triangle corner
\draw (3.8,0) -- (3.8, .2) -- (4,.2);

% the x measuring bar
\draw [{Bar}-{Bar}] (0,-0.3) --  (4,-0.3);

% the y measuring bar
\draw [{Bar}-{Bar}] (4.3,0) --  (4.3,3);

% the little arc for the angle
\draw [radius = 0.5 cm] (0.5,0) arc
[start angle=0, end angle =  36.87];

% the dashed line pointing about 5:00 from the apple
\draw [dashed, ->] (5/2,15/8) -- (79/25,199/200);
```

```
% the line pointing down from the apple
\draw [->] (2.5,15/8) -- (5/2,.5);

% the line parallel to the plane
\draw [dashed, ->] (79/25,199/200) -- (5/2,1/2);

\end{tikzpicture}
```



Before we go any further: do those arrows on the little triangle look a little hard to see to you? They do to me, so I'm going to make them look nicer. First of all, Ti*k*Z's default arrow tip looks a bit curvy and insubstantial for this picture. I want the arrow tips to be solid triangles. Looking through the section 16.5 Reference: Arrow Tips in the Ti*k*Z Manual, I find an arrow tip called `Triangle`. Here is what just the little force arrow triangle looks like with the `Triangle` arrow tip substituted for the default one:

```
\[
\begin{tikzpicture}
% the dashed line pointing about 5:00 from the apple
\draw [dashed, -{Triangle}] (5/2,15/8) -- (79/25,199/200);

% the line pointing down from the apple
\draw [-{Triangle}] (2.5,15/8) -- (5/2,.5);

% the line parallel to the plane
\draw [dashed, -{Triangle}] (79/25,199/200) -- (5/2,1/2);
\end{tikzpicture}
\]
```

Now one of the arrows looks great, but the other two are on top of each other, making it hard to distinguish them. One solution to this problem is to have one (or both) arrow stop just slightly short of their destination. Here we've made the downwards-pointing arrow stop a distance of 0.1 above its usual destination.

```
\[
\begin{tikzpicture}
% the dashed line pointing about 5:00 from the apple
```

```
\draw [dashed, -{Triangle}] (5/2,15/8) -- (79/25,199/200);

% the line pointing down from the apple
\draw [-{Triangle}] (2.5,15/8) -- (5/2,.5+.05);

% the line parallel to the plane
\draw [dashed, -{Triangle}] (79/25,199/200) -- (5/2,1/2);
\end{tikzpicture}
\]
```

Another possible solution is to leave the arrows in their original positions but use a narrower arrow tip, like `Stealth`:

```
\[
\begin{tikzpicture}
% the dashed line pointing about 5:00 from the apple
\draw [dashed, -{Stealth}] (5/2,15/8) -- (79/25,199/200);

% the line pointing down from the apple
\draw [-{Stealth}] (2.5,15/8) -- (5/2,.5);

% the line parallel to the plane
\draw [dashed, -{Stealth}] (79/25,199/200) -- (5/2,1/2);
\end{tikzpicture}
\]
```

I like this option – using the `Stealth` tip – better, so I've incorporated it into our big picture below.

I've also inserted force arrow labels near the midpoints of the lines using the `node` command, as we did in Project 1. I did some fine-tuning of the placement of the labels using the optional offset to the node placement arguments. For example, the command

```
\node [below right] at (283/100, 299/400) {$F_{||}$};
```

put the label $F_{||}$ a bit below and to the right of the point $(\frac{283}{100}, \frac{299}{400})$. In my opinion, this default `below right` placement put the label a little too far from its line, so I inserted a negative offset:

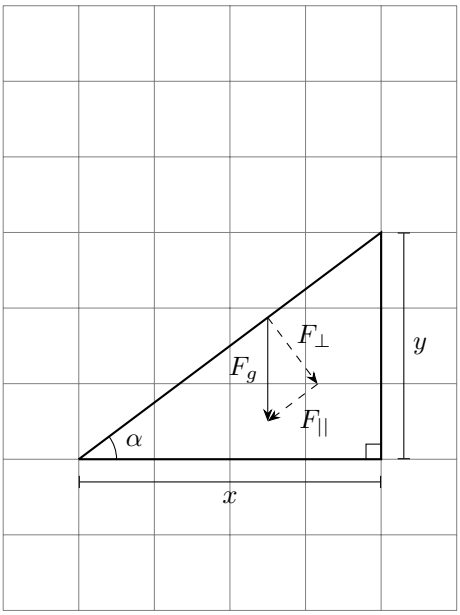
```
\node [below right=-1pt] at (283/100, 299/400) {$F_{||}$};
```

(See Section 17.5.2, Basic Placement Options, in the Nodes and Edges chapter of the Ti*k*Z Manual for more details.)

Here is what we have so far:

```
\begin{tikzpicture}
% load the arrows library
\usetikzlibrary{arrows.meta}

% help lines (comment out when done)
\draw [help lines] (-1,-2) grid (5,6);

% the triangle for the inclined plane
```

```
\draw [thick] (0,0) -- (4,0) -- (4,3) -- (0,0);

% the little right triangle corner
\draw (3.8,0) -- (3.8, .2) -- (4,.2);

% the x measuring bar
\draw [{Bar}-{Bar}] (0,-0.3) --  (4,-0.3);

% the y measuring bar
\draw [{Bar}-{Bar}] (4.3,0) --  (4.3,3);

% the little arc for the angle
\draw [radius = 0.5 cm] (0.5,0) arc
[start angle=0, end angle =  36.87];

% the dashed line pointing about 5:00 from the apple, and the label on it
\draw [dashed, -{Stealth}] (5/2,15/8) -- (79/25,199/200);
\node [above right=-2pt] at (283/100, 287/200) {$F_{\perp}$};

% the line pointing down from the apple, and the label on it
\draw [-{Stealth}] (2.5,15/8) -- (5/2,.5);
\node [left] at (5/2, 19/16) {$F_g$};

% the line parallel to the plane, and the label on it
\draw [dashed, -{Stealth}] (79/25,199/200) -- (5/2,1/2);
\node [below right=-1pt] at (283/100, 299/400) {$F_{||}$};

\end{tikzpicture}
```



Now we just need to place the apple!

Before we go any further, a **NOTE:** If all you want to do it to display a picture amongst your text, there's no need to use Ti*k*Z: LaTeX's `graphicx` package, which includes the `includegraphics` command, is perfectly fine. There are a couple of situations where you'd want to use Ti*k*Z rather than LaTeX's `graphicx` package:

- Use Ti*k*Z when you are including a graphic within a Ti*k*Z picture (so that the graphic has to be properly sized, spaced, layered, and rotated with respect to the rest of the picture), which is what we're doing here.

- Use Ti*k*Z when you wish to use masking (using an image to hide or reveal part of another image) – as far as I know, this isn't possible with just LaTeX, although it might be in the future.

Now let's talk about including images in Ti*k*Z pictures. My apple picture is called `apple.png`, and I've saved it in the same directory on Overleaf as this present document.

Before I can use the apple picture in my Ti*k*Z picture, I need to **declare the image** at the size it'll be used in my Ti*k*Z picture. This I've done by putting `\pgfdeclareimage[height=10mm]{the_apple}{apple}` in the preamble. (Note: you can also just include the image within the `tikzpicture` environment where you're using it. I've declared it in the preamble here because I'll be using it in several different Ti*k*Z pictures within the same document.)

- I've included the optional `height` argument in my declaration. By setting only the height and not the width (or vice versa), I preserve the aspect ratio of the original picture. I chose 10mm for the height after playing around with re-sizings of the apple compared to the inclined plane picture until I found the size that looked right.
- `the_apple` is the nickname I'm giving to my apple picture – this is how I'll refer to it within a Ti*k*Z picture.
- `apple` is the actual name of the file with my apple picture in it – *without* the file extension.

Having declared the apple picture, I can use it in the `tikzpicture` environment. Here it is, at the size I want it for the big picture, with no rotation:

```
\begin{tikzpicture}
\pgftext[at=\pgfpoint{0cm}{0cm}]{\pgfuseimage{the_apple}}
\end{tikzpicture}
```

Next I need to determine how much to rotate the apple. Some more paper-and-pencil work involving similar triangles told me that I needed to rotate the apple by $\tan^{-1}(\frac{3}{4}) \approx 36.87°$ anticlockwise from the vertical (that's the same angle measure as α), in order that it appear to be sliding on its base down the inclined plane. That's what the next command accomplishes … only what Ti*k*Z is *really* doing is a coordinate transformation. It's rotating the entire plane anticlockwise by 36.87°. (See Section 25.3: Coordinate Transformations in the Ti*k*Z Manual.)

```
\begin{tikzpicture}
\pgftext[rotate=36.87, at=\pgfpoint{0cm}{0cm}]{\pgfuseimage{the_apple}}
\end{tikzpicture}
```

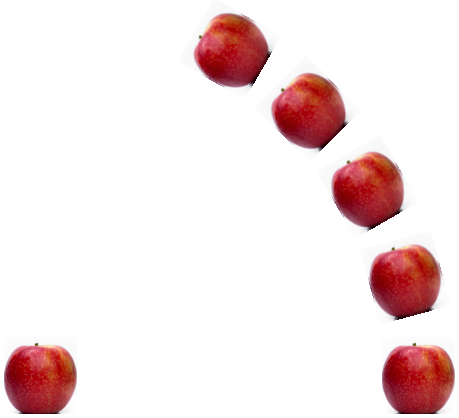
In the following command, the the `at=\pgfpoint{5cm}{0cm}` part doesn't actually move the resulting picture 5 units to the right, as you might expect; the result looks exactly like the previous picture. This is basically because there is nothing else for `pgf` to refer to when it places the image.

```
\begin{tikzpicture}
\pgftext[rotate=36.87, at=\pgfpoint{5cm}{0cm}]{\pgfuseimage{the_apple}}
\end{tikzpicture}
```

Now compare this with what happens when we place a sequence of apples at a distance of 5 cm from the origin, with rotations about the origin of 0°, 15°, 30°, 45°, and 60°.

```
\begin{tikzpicture}
% the base apple, sitting at (0,0) with no rotation
```

```
\pgftext[rotate=0, at=\pgfpoint{0cm}{0cm}]{\pgfuseimage{the_apple}}
% distance = 5cm, rotation = 0
\pgftext[rotate=0, at=\pgfpoint{5cm}{0cm}]{\pgfuseimage{the_apple}}
% distance = 5cm, rotation = 15
\pgftext[rotate=15, at=\pgfpoint{5cm}{0cm}]{\pgfuseimage{the_apple}}
% distance = 5cm, rotation = 30
\pgftext[rotate=30, at=\pgfpoint{5cm}{0cm}]{\pgfuseimage{the_apple}}
% distance = 5cm, rotation = 45
\pgftext[rotate=45, at=\pgfpoint{5cm}{0cm}]{\pgfuseimage{the_apple}}
% distance = 5cm, rotation = 60
\pgftext[rotate=60, at=\pgfpoint{5cm}{0cm}]{\pgfuseimage{the_apple}}
\end{tikzpicture}
```



This would imply that to get the apple into the right place in our inclined plane picture, we should give it a rotation of 36.87° and place it at a distance of $d((0,0),(2.5,\frac{15}{8}))$ from the origin, like this:

```
\pgftext[rotate=36.87, at=\pgfpoint{3.125cm}{0cm}]{\pgfuseimage{the_apple}}
```

The problem with this is that it puts the line of the inclined plane going right through the apple. (Try it in the finished picture below!)

What I actually did when I placed the apple was to eyeball it. The x-value 3.125 looked OK, but then I needed to move the apple up and to the left. Fortunately, this was easy to do, because when `pgf` does a rotation, it is really making a coordinate transformation. That is, having rotated 36.87°, we just need to move it up or down with respect to that coordinate transformation. For example, in the command

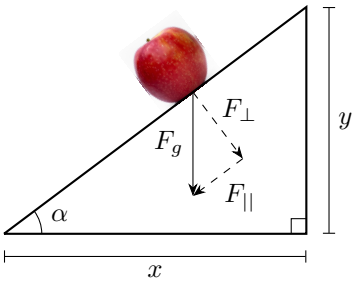
```
\pgftext[rotate=36.87, at=\pgfpoint{3.15cm}{.5cm}]{\pgfuseimage{the_apple}},
```

the .5cm moves the apple "up" in the rotated coordinate system a distance of .5cm. This corresponds to moving the apple in a direction perpendicular to the inclined plane by .5cm.

Finally, we placed the apple "first" in the picture so that it would be "under" the lines for the rest of the inclined plane picture. (This is so that the picture wouldn't cover up some of the width of the lines, and so that the apple would appear to be resting on the inclined plane.)

```
\begin{tikzpicture}
% load the arrows library
\usetikzlibrary{arrows.meta}

% the apple
% (place it first so it goes underneath!)
\pgftext[rotate=36.87, at=\pgfpoint{3.15cm}{.5cm}]{\pgfuseimage{the_apple}}

% help lines (comment out when done)
%\draw [help lines] (-1,-2) grid (5,6);

% the triangle for the inclined plane
```

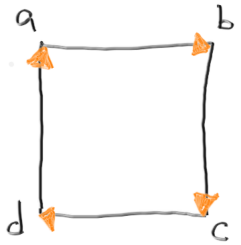
```
\draw [thick] (0,0) -- (4,0) -- (4,3) -- (0,0);

% the little right triangle corner
\draw (3.8,0) -- (3.8, .2) -- (4,.2);

% the x measuring bar, and the x label
\draw [{Bar}-{Bar}] (0,-0.3) --  (4,-0.3);
\node [below] at (2,-0.3) {$x$};

% the y measuring bar, and the y label
\draw [{Bar}-{Bar}] (4.3,0) --  (4.3,3);
\node [right] at (4.3,1.5) {$y$};

% the little arc for the angle,
% and the alpha label
\draw [radius = 0.5 cm] (0.5,0) arc
[start angle=0, end angle =  36.87];
\node [right] at (0.5,0.25) {$\alpha$};

% the dashed line pointing about 5:00 from the apple, and the label on it
\draw [dashed, -{Stealth}] (5/2,15/8) -- (79/25,199/200);
\node [above right=-2pt] at (283/100, 287/200) {$F_{\perp}$};

% the line pointing down from the apple, and the label on it
\draw [-{Stealth}] (2.5,15/8) -- (5/2,.5);
\node [left] at (5/2, 19/16) {$F_g$};

% the line parallel to the plane, and the label on it
\draw [dashed, -{Stealth}] (79/25,199/200) -- (5/2,1/2);
\node [below right=-1pt] at (283/100, 299/400) {$F_{||}$};

\end{tikzpicture}
```



I think this is done! As usual, fine-tune it if you wish, and then try these exercises.

Exercises for Project 4

1. Take a picture with your phone. Save it as a .jpg or.png file, upload it to Overleaf, and put it in the same folder as the one where you're doing these exercises. Declare it at a convenient size, and display it right-side up and then upside down.
2. Use Ti*k*Z to draw a square about 2 inches on a side. Put your picture from Problem 1 in the middle of it (you may have to change the size at which you declare it).
3. Draw a square with vertices at each corner, and label them a, b, c, and d. Put fancy arrow tips (from the `arrows.meta` library) on the lines so that they go clockwise around the square. Now you have a directed graph on 4 vertices.



If you want to have the arrows look as they do in the picture, with black lines and colored arrow tips, like this ⟶, the syntax for that is `\draw [-{Triangle[orange]}] (0,0) -- (1,0);` You can also adjust the size of the arrow tips with optional arguments for length and width: `\draw [-{Triangle[orange, length=3mm, width=3mm]}] (0,0) -- (1,0);` produces ⟶

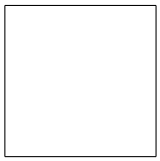
4. Take the completed inclined plane / apple picture from this project and substitute your picture for the apple. Rotate it so that it looks like it's sliding down the plane, just like we did for the apple.

Possible solutions to exercises

Project 1: parabola, possible exercise solutions

1. Make a square with its corners at $(-1, 1)$, $(1, 1)$, $(1, -1)$, and $(-1, -1)$.

```
\begin{tikzpicture}
\draw (-1,1) -- (1,1);
\draw (1,1) -- (1,-1);
\draw (1,-1) -- (-1,-1);
\draw (-1,-1) -- (-1,1);
\end{tikzpicture}
```

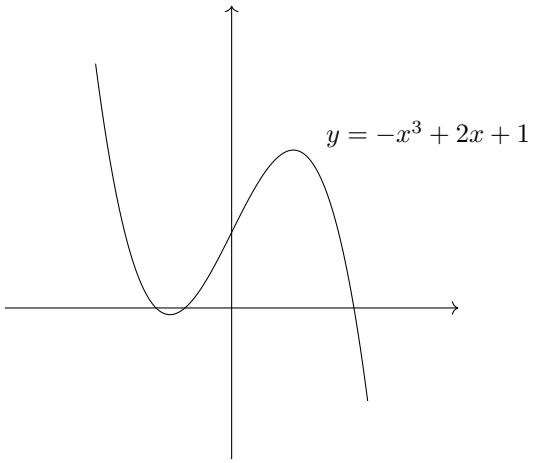


2. Make a yellow diamond with its corners at $(0, 1)$, $(1, 0)$, $(0, -1)$, and $(-1, -0)$. Center the picture on the page horizontally.

```
\[
\begin{tikzpicture}
\draw [yellow] (0,1) -- (1,0);
\draw [yellow] (1,0) -- (0,-1);
\draw [yellow] (0,-1) -- (-1,0);
\draw [yellow] (-1,0) -- (0,1);
\end{tikzpicture}
\]
```

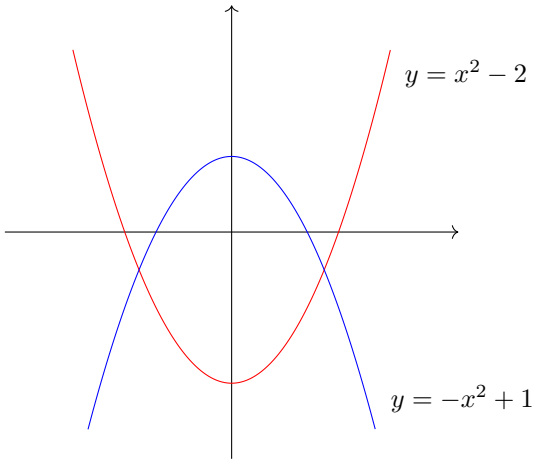

3. Make a picture similar to our final parabola picture, except change the function to $y = -x^3+2x+1$. Adjust the axes as needed. Include a label with the equation of the function.

```
\begin{tikzpicture}
\draw [->] (-3,0) -- (3,0);
\draw [->] (0,-2) -- (0,4);
\draw [smooth, samples=100, domain=-1.8:1.8] plot(\x, {-(\x)^3 + 2*(\x) +1});
\node at (2.6,2.3) {$y=-x^3 + 2x + 1$};
\end{tikzpicture}
```



4. Make a picture showing the graphs of the functions $y = x^2 - 2$ and $y = -x^2 + 1$ on the same set of axes. Color the $y = x^2 - 2$ graph red and the $y = -x^2 + 1$ graph blue. Include a label for each graph showing its equation.

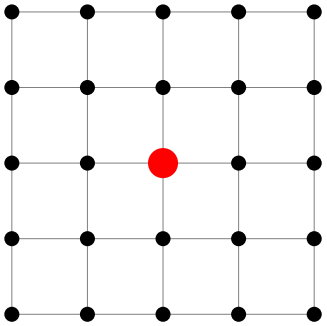
```
\begin{tikzpicture}
\draw [->] (-3,0) -- (3,0);
\draw [->] (0,-3) -- (0,3);
\draw [red, smooth, samples=100, domain=-2.1:2.1] plot(\x, {(\x)^2 -2});
\draw [blue, smooth, samples=100, domain=-1.9:1.9] plot(\x, {-(\x)^2 +1});
\node at (3.1,2.1) {$y=x^2-2$};
\node at (3.05,-2.2) {$y=-x^2+1$};
\end{tikzpicture}
```



Project 2: bipartite graph, possible exercise solutions

1. Make a grid of help lines with the lower left corner of the rectangular grid at $(-2, -2)$ and the upper right corner at $(2, 2)$. Using the `vertex` style developed in this project (modified if you like), put a dot at each non-origin intersection point of the help lines. Put a noticeably different sort of dot at the origin.

```
\begin{tikzpicture}

% vertex style for dots
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% grid lines
\draw[help lines] (-2,-2) grid (2,2);

% bottom row of dots (with y=-2)
\node [vertex] at (-2,-2) {};
\node [vertex] at (-1,-2) {};
\node [vertex] at (0,-2) {};
\node [vertex] at (1,-2) {};
\node [vertex] at (2,-2) {};

% next row up of dots (with y=-1)
\node [vertex] at (-2,-1) {};
\node [vertex] at (-1,-1) {};
\node [vertex] at (0,-1) {};
\node [vertex] at (1,-1) {};
\node [vertex] at (2,-1) {};

% middle row up of dots (with y=0)
\node [vertex] at (-2,0) {};
\node [vertex] at (-1,0) {};
% big red dot at (0,0)
\node [circle, red, fill, inner sep=4pt] at (0,0) {};
\node [vertex] at (1,0) {};
\node [vertex] at (2,0) {};

% next row up of dots (with y=1)
\node [vertex] at (-2,1) {};
\node [vertex] at (-1,1) {};
\node [vertex] at (0,1) {};
\node [vertex] at (1,1) {};
\node [vertex] at (2,1) {};

% top row up of dots (with y=2)
\node [vertex] at (-2,2) {};
\node [vertex] at (-1,2) {};
\node [vertex] at (0,2) {};
\node [vertex] at (1,2) {};
\node [vertex] at (2,2) {};

\end{tikzpicture}
```

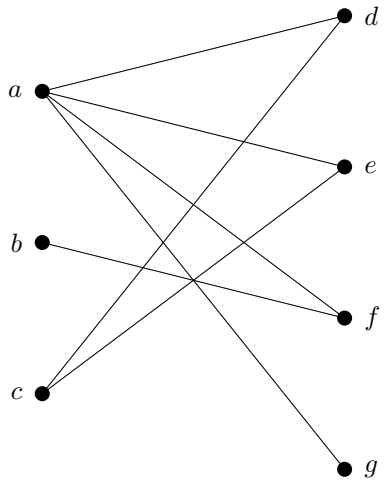


2. Modify the final graph picture in this project so that there is an additional edge from a to f, but

no edge from b to e.

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
% \draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);
\draw (0,2) -- (4,-1);

% edges from vertex b
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a
\node [left=4pt] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};
% label at vertex b
\node [left=4pt] at (0,0) {$b$};

% dot at vertex c
\node [vertex] at (0,-2) {};
% label at vertex a
\node [left=4pt] at (0,-2) {$c$};

% dot at vertex d
\node [vertex] at (4,3) {};
% label at vertex d
\node [right=4pt] at (4,3) {$d$};

% dot at vertex e
\node [vertex] at (4,1) {};
% label at vertex e
\node [right=4pt] at (4,1) {$e$};

% dot at vertex f
\node [vertex] at (4,-1) {};
% label at vertex f
\node [right=4pt] at (4,-1) {$f$};

% dot at vertex g
\node [vertex] at (4,-3) {};
% label at vertex g
\node [right=4pt] at (4,-3) {$g$};
```

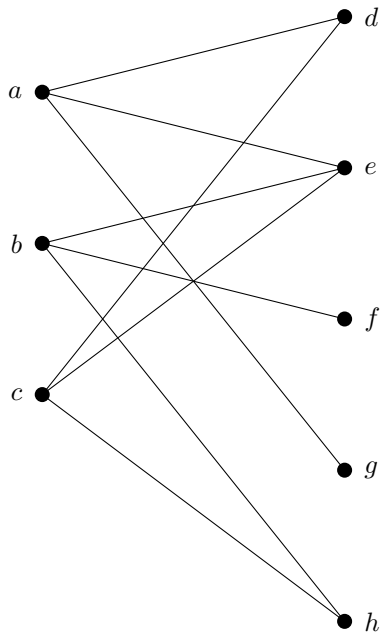
```
\end{tikzpicture}
```



3. Modify the final graph picture in this project by adding an additional vertex h at $(4, -5)$, and give it edges to c and b.

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
% \draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);
\draw (0,0) -- (4,-5);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);
\draw (0,-2) -- (4,-5);

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a
\node [left=4pt] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};
% label at vertex b
\node [left=4pt] at (0,0) {$b$};

% dot at vertex c
\node [vertex] at (0,-2) {};
% label at vertex a
\node [left=4pt] at (0,-2) {$c$};
```

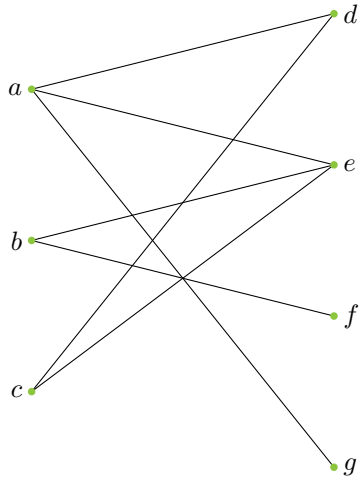
```
% dot at vertex d
\node [vertex] at (4,3) {};
% label at vertex d
\node [right=4pt] at (4,3) {$d$};

% dot at vertex e
\node [vertex] at (4,1) {};
% label at vertex e
\node [right=4pt] at (4,1) {$e$};

% dot at vertex f
\node [vertex] at (4,-1) {};
% label at vertex f
\node [right=4pt] at (4,-1) {$f$};

% dot at vertex g
\node [vertex] at (4,-3) {};
% label at vertex g
\node [right=4pt] at (4,-3) {$g$};

% dot at vertex h
\node [vertex] at (4,-5) {};
% label at vertex h
\node [right=4pt] at (4,-5) {$h$};

\end{tikzpicture}
```



4. Modify the final graph picture in this project by defining and using a new vertex style, called `vertex2`, that makes dots that are about half the size as the ones in the `vertex` style and are also green. (If you'd like to use something fancier than `green`, see the "Using colors in LaTeX" page in the Overleaf help documentation for a list of available `xcolor` colors. Be sure you've loaded the appropriate `xcolor` option before loading Ti*k*Z in your preamble.) You may want to adjust the spacing between the labels and the vertices too.

```
\begin{tikzpicture}

% \usepackage[dvipsnames]{xcolor} is needed in preamble
```

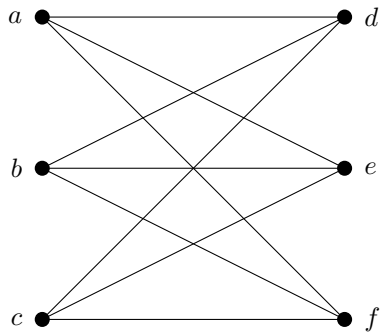
```
% for the LimeGreen color

% vertex style to be used at vertex nodes
\tikzstyle{vertex2}=[circle, fill=LimeGreen, inner sep=1pt]

% help lines (comment out when done)
% \draw [help lines] (-1,-4) grid (5,4);

% edges from vertex a
\draw (0,2) -- (4,3);
\draw (0,2) -- (4,1);
\draw (0,2) -- (4,-3);

% edges from vertex b
\draw (0,0) -- (4,1);
\draw (0,0) -- (4,-1);

% edges from vertex c
\draw (0,-2) -- (4,3);
\draw (0,-2) -- (4,1);

% dot at vertex a
\node [vertex2] at (0,2) {};
% label at vertex a
\node [left] at (0,2) {$a$};

% dot at vertex b
\node [vertex2] at (0,0) {};
% label at vertex b
\node [left] at (0,0) {$b$};

% dot at vertex c
\node [vertex2] at (0,-2) {};
% label at vertex a
\node [left] at (0,-2) {$c$};

% dot at vertex d
\node [vertex2] at (4,3) {};
% label at vertex d
\node [right] at (4,3) {$d$};

% dot at vertex e
\node [vertex2] at (4,1) {};
% label at vertex e
\node [right] at (4,1) {$e$};

% dot at vertex f
\node [vertex2] at (4,-1) {};
% label at vertex f
\node [right] at (4,-1) {$f$};

% dot at vertex g
\node [vertex2] at (4,-3) {};
% label at vertex g
\node [right] at (4,-3) {$g$};

\end{tikzpicture}
```



5. Draw a picture of $K_{3,3}$, the complete bipartite graph from 3 vertices to 3 vertices, using ingredients from the code in the block for our final graph picture in this project.

```
\begin{tikzpicture}

% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=2pt]

% help lines (comment out when done)
% \draw [help lines] (-1,-3) grid (5,3);

% edges from vertex a
\draw (0,2) -- (4,2);
\draw (0,2) -- (4,0);
\draw (0,2) -- (4,-2);

% edges from vertex b
\draw (0,0) -- (4,2);
\draw (0,0) -- (4,0);
\draw (0,0) -- (4,-2);

% edges from vertex c
\draw (0,-2) -- (4,2);
\draw (0,-2) -- (4,0);
\draw (0,-2) -- (4,-2);

% dot at vertex a
\node [vertex] at (0,2) {};
% label at vertex a
\node [left=4pt] at (0,2) {$a$};

% dot at vertex b
\node [vertex] at (0,0) {};
% label at vertex b
\node [left=4pt] at (0,0) {$b$};

% dot at vertex c
\node [vertex] at (0,-2) {};
% label at vertex a
\node [left=4pt] at (0,-2) {$c$};

% dot at vertex d
```

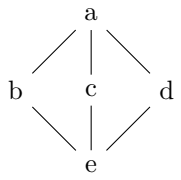
```
\node [vertex] at (4,2) {};
% label at vertex d
\node [right=4pt] at (4,2) {$d$};

% dot at vertex e
\node [vertex] at (4,0) {};
% label at vertex e
\node [right=4pt] at (4,0) {$e$};

% dot at vertex f
\node [vertex] at (4,-2) {};
% label at vertex f
\node [right=4pt] at (4,-2) {$f$};

\end{tikzpicture}
```



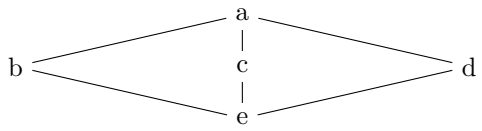
Project 3: a field extension diagram, possible exercise solutions

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:

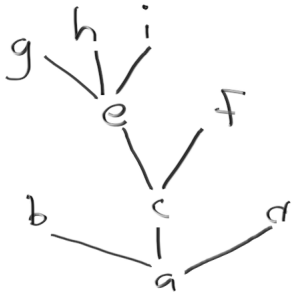
```
\begin{tikzpicture}
\graph [layered layout] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```




```
\begin{tikzpicture}
\graph [layered layout,  sibling distance=3cm,  level distance=0.5cm] {
a -- {b, c, d};
{b, c, d} -- e;
};
\end{tikzpicture}
```

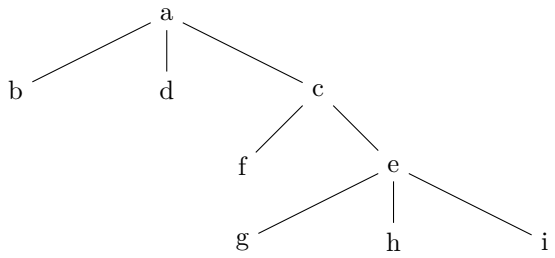


2. Create the following diagram in Ti*k*Z:



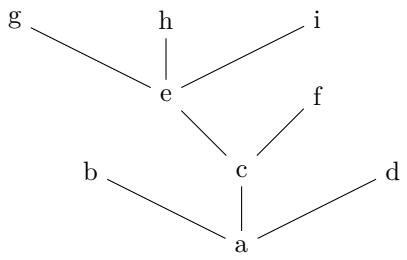
If you list the edges going from bottom to top, you will probably find that Ti*k*Z draws a graph that is isomorphic to the hand-drawn graph above, but with the order of the levels reversed – that is, the tree produced grows downward (like a root system instead of a tree). This orientation is baked into the algorithm that the Layered Layout uses; see `https://tikz.dev/gd-layered` for details. You may also find that the ordering of nodes within a level is different from the hand-drawn picture – see comment after the exercise 4. solution.

```
\begin{tikzpicture}
\graph [layered layout, sibling distance = 2cm] {
a -- {b, c, d};
c -- {e, f};
e -- {g, h, i};
};
\end{tikzpicture}
```

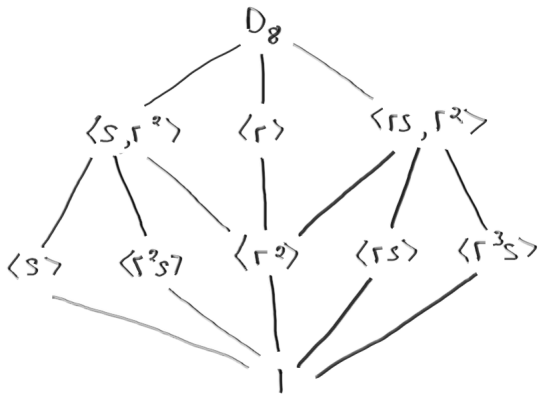


If you really do want this tree to grow from top to bottom just as in the hand-drawn picture, you can list the edges from top to bottom, like this:

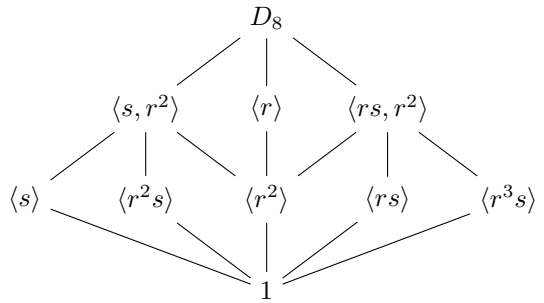
```
\begin{tikzpicture}
\graph [layered layout, sibling distance = 2cm] {
g -- e;
h -- e;
i -- e;
e -- c;
f -- c;
b -- a;
c -- a;
d -- a;
};
\end{tikzpicture}
```



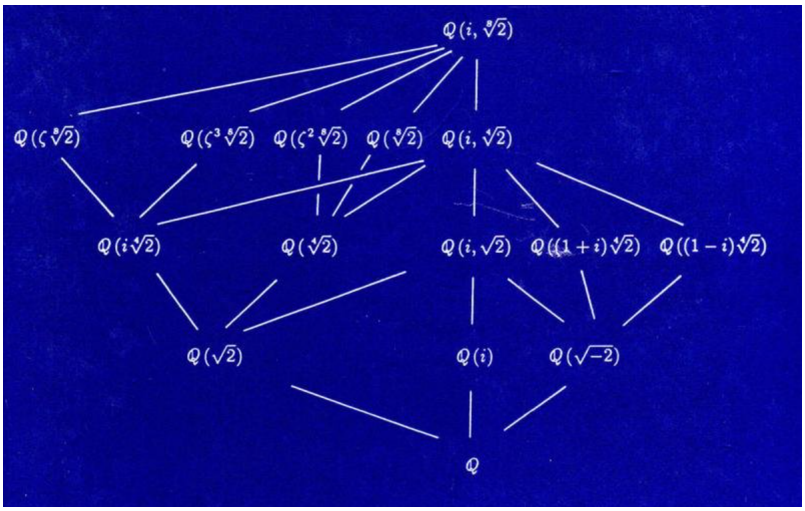
3. Create the following diagram in TikZ:



```
\begin{tikzpicture}
\graph [layered layout, sibling distance = 1.5cm,
level distance = 1.5cm] {"$S_3$" -- {"$\langle (1,2) \rangle$"
[nudge=(left:14mm)], "$\langle (1,3) \rangle$"[nudge=(left:13mm)],
"$\langle (2,3) \rangle$"[nudge=(left:13mm)],
"$\langle (1,2,3) \rangle$"[nudge=(up:7mm)]};
{"$\langle (1,2) \rangle$", "$\langle (1,3)
\rangle$", "$\langle (2,3) \rangle$", "$\langle (1,2,3) \rangle$"}
-- "$1$";
};
\end{tikzpicture}
```



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

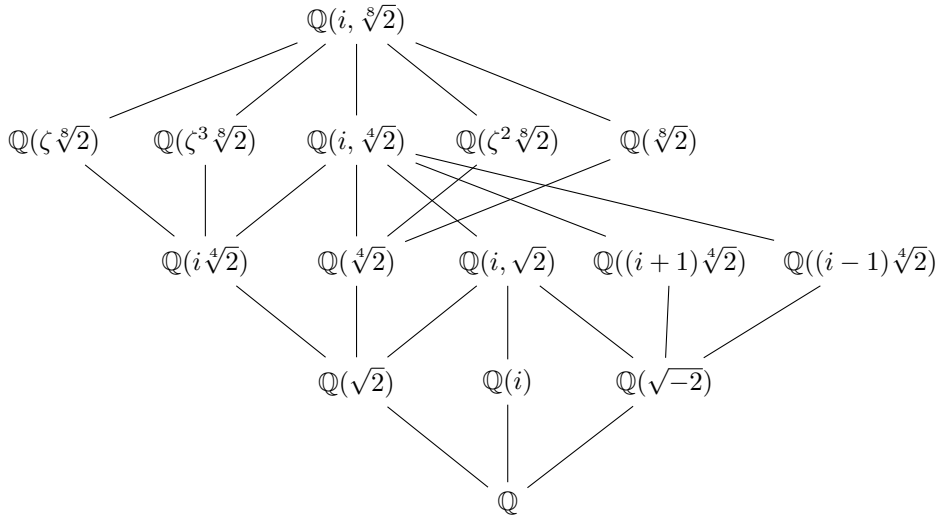


This is the diagram on the cover of the first edition of Dummit and Foote's *Abstract Algebra*. You may find it tricky to make Ti*k*Z reproduce the diagram exactly, but you should at least reproduce an isomorphic graph that preserves levels horizontally (though not necessarily the order of the nodes within levels) and has no more than a few edge-crossings.

Note: Ti*k*Z's Layered Layout algorithm uses a heuristic to try to 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.

```
\begin{tikzpicture}
\graph [layered layout, level distance = 1.6cm, sibling distance = 2cm] {
"$\mathbb{Q}(i, \sqrt[8]{2})$" --
{"$\mathbb{Q}(\zeta \sqrt[8]{2})$",
"$\mathbb{Q}(\zeta^3 \sqrt[8]{2})$",
"$\mathbb{Q}(\zeta^2 \sqrt[8]{2})$",
"$\mathbb{Q}(\sqrt[8]{2})$",
"$\mathbb{Q}(i, \sqrt[4]{2})$"};
"$\mathbb{Q}(\zeta \sqrt[8]{2})$" --
"$\mathbb{Q}(i \sqrt[4]{2})$";
"$\mathbb{Q}(\zeta^3 \sqrt[8]{2})$" --
"$\mathbb{Q}(i \sqrt[4]{2})$";
"$\mathbb{Q}(\zeta^2 \sqrt[8]{2})$" --
"$\mathbb{Q}(\sqrt[4]{2})$";
"$\mathbb{Q}(\sqrt[8]{2})$" --
"$\mathbb{Q}(\sqrt[4]{2})$";
"$\mathbb{Q}(i, \sqrt[4]{2})$" --
{"$\mathbb{Q}(i \sqrt[4]{2})$",
"$\mathbb{Q}(\sqrt[4]{2})$",
"$\mathbb{Q}(i, \sqrt{2})$",
"$\mathbb{Q}((i+1) \sqrt[4]{2})$",
"$\mathbb{Q}((i-1) \sqrt[4]{2})$"};
"$\mathbb{Q}(i \sqrt[4]{2})$" --
"$\mathbb{Q}(\sqrt{2})$";
"$\mathbb{Q}(\sqrt[4]{2})$" --
"$\mathbb{Q}(\sqrt{2})$";
"$\mathbb{Q}(i, \sqrt{2})$" --
{"$\mathbb{Q}(\sqrt{2})$",
"$\mathbb{Q}(i)$", "$\mathbb{Q}(\sqrt{-2})$"};
"$\mathbb{Q}((i+1) \sqrt[4]{2})$" --
"$\mathbb{Q}(\sqrt{-2})$";
"$\mathbb{Q}((i-1) \sqrt[4]{2})$" --
```

```
"$\mathbb{Q}(\sqrt{-2})$";
"$\mathbb{Q}(\sqrt{2})$" -- "$\mathbb{Q}$";
"$\mathbb{Q}(i)$" -- "$\mathbb{Q}$";
"$\mathbb{Q}(\sqrt{-2})$" --
"$\mathbb{Q}$";
};
\end{tikzpicture}
```



Project 4: inclined plane, possible exercise solutions

1. Take a picture with your phone. Save it as a .jpg or.png file, upload it to Overleaf, and put it in the same folder as the one where you're doing these exercises. Declare it at a convenient size, and display it right-side up and then – to the right of the original – upside down.

   Note for this one that because we want the second cat picture to end up (1) upside-down and (2) to the right of the original picture, we need to initially place the picture to the *left* of the original picture, and then rotate it *clockwise*.

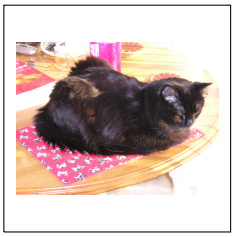
   ```
   \begin{tikzpicture}
   % declare the cat picture:
   \pgfdeclareimage[height=10mm]{cat_pic}{Sophie}
   % right-side-up cat:
   \pgftext[at=\pgfpoint{0cm}{0cm}]{\pgfuseimage{cat_pic}}
   % upside-down cat
   \pgftext[rotate=-180, at=\pgfpoint{-5cm}{0cm}]{\pgfuseimage{cat_pic}}
   \end{tikzpicture}
   ```

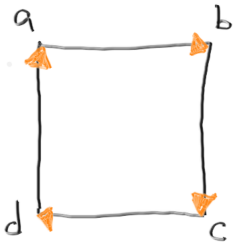
2. Use Ti*k*Z to draw a square about 2 inches on a side. Put your picture from Problem 1 in the middle of it (you may have to change the size at which you declare it).

   ```
   \begin{tikzpicture}
   % declaring the cat at a bit bigger scale
   \pgfdeclareimage[height=20mm]{cat_pic}{Sophie}
   % the square
   \draw (0,0) -- (3,0) -- (3,3) -- (0,3) -- (0,0);
   % the cat
   \pgftext[at=\pgfpoint{1.5cm}{1.5cm}]{\pgfuseimage{cat_pic}}
   ```

```
\end{tikzpicture}
```

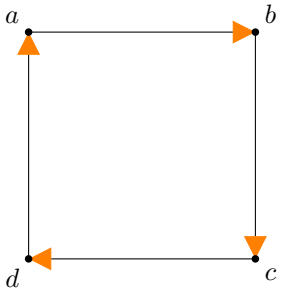


3. Draw a square with vertices at each corner, and label them a, b, c, and d. Put fancy arrow tips (from the `arrows.meta` library) on the lines so that they go clockwise around the square. Now you have a directed graph on 4 vertices.



If you want to have the arrows look as they do in the picture, with black lines and colored arrow tips, like this ⟶, the syntax for that is `\draw [-{Triangle[orange]}] (0,0) -- (1,0);` You can also adjust the size of the arrow tips with optional arguments for length and width: `\draw [-{Triangle[orange, length=3mm, width=3mm]}] (0,0) -- (1,0);` produces ⟶

```
\begin{tikzpicture}
% vertex style to be used at vertex nodes
\tikzstyle{vertex}=[circle, fill, inner sep=1pt]

% the square, with arrows
\draw  [-{Triangle[orange, length=3mm, width=3mm]}] (0,0) -- (0,3);
\draw  [-{Triangle[orange, length=3mm, width=3mm]}] (0,3) -- (3,3);
\draw  [-{Triangle[orange, length=3mm, width=3mm]}] (3,3) -- (3,0);
\draw  [-{Triangle[orange, length=3mm, width=3mm]}] (3,0) -- (0,0);

% vertices at corners
\node [vertex] at (0,0) {};
\node [vertex] at (3,0) {};
\node [vertex] at (0,3) {};
\node [vertex] at (3,3) {};

% vertex labels
\node [above left] at (0,3) {$a$};
\node [above right] at (3,3) {$b$};
\node [below right] at (3,0) {$c$};
\node [below left] at (0,0) {$a$};
\end{tikzpicture}
```



4. Take the completed inclined plane / apple picture from this project and substitute your picture for the apple. Rotate it so that it looks like it's sliding down the plane, just like we did for the apple.

```
\begin{tikzpicture}
% load the arrows library
\usetikzlibrary{arrows.meta}

% the cat
\pgfdeclareimage[height=15mm]{cat_pic}{Sophie}
% (place it first so it goes underneath!)
\pgftext[rotate=36.87, at=\pgfpoint{3.1cm}{.75cm}]{\pgfuseimage{cat_pic}}

% help lines (comment out when done)
%\draw [help lines] (-1,-2) grid (5,6);

% the triangle for the inclined plane
\draw [thick] (0,0) -- (4,0) -- (4,3) -- (0,0);

% the little right triangle corner
\draw (3.8,0) -- (3.8, .2) -- (4,.2);

% the x measuring bar, and the x label
\draw [{Bar}-{Bar}] (0,-0.3) --  (4,-0.3);
\node [below] at (2,-0.3) {$x$};

% the y measuring bar, and the y label
\draw [{Bar}-{Bar}] (4.3,0) --  (4.3,3);
\node [right] at (4.3,1.5) {$y$};

% the little arc for the angle,
% and the alpha label
\draw [radius = 0.5 cm] (0.5,0) arc
[start angle=0, end angle =  36.87];
\node [right] at (0.5,0.25) {$\alpha$};

% the dashed line pointing about 5:00 from the apple, and the label on it
\draw [dashed, -{Stealth}] (5/2,15/8) -- (79/25,199/200);
\node [above right=-2pt] at (283/100, 287/200) {$F_{\perp}$};

% the line pointing down from the apple, and the label on it
\draw [-{Stealth}] (2.5,15/8) -- (5/2,.5);
\node [left] at (5/2, 19/16) {$F_g$};

% the line parallel to the plane, and the label on it
\draw [dashed, -{Stealth}] (79/25,199/200) -- (5/2,1/2);
\node [below right=-1pt] at (283/100, 299/400) {$F_{||}$};

\end{tikzpicture}
```