Drawing Graphs with Ti*k*Z

This is a **LaTeX task**. You must submit both your `.tex` source file and a compiled `.pdf` file. Your response will be assessed on both content and presentation.

Last time, we looked at using Ti*k*Z to draw trees. This time, we'll look at graphs!

We'll start with an unweighted, undirected graph, then add directions and weights to turn it into a flow network. Finally, we'll add some styles to show the minimum cut.

▶ Steps for you to do are marked with a triangle.

Setting up

▶ To start, create a new *standalone* document, just like we did when making a tree. Import the Ti*k*Z package in a `.tex` file, and open a `tikzpicture` environment:

```
\documentclass{standalone}
\usepackage{tikz}
\begin{document}
    \begin{tikzpicture}

    \end{tikzpicture}
\end{document}
```

Placing Nodes

Just like last time, we'll start with a node. We've seen the syntax for this:

```
    \node[draw, fill=red!10] (a) at (0, 0) {Hello! $2 + 2 = 4$};
```

Recall that:

- `\node` is the command that tells Ti*k*Z to create a node,
- The square brackets `[]` contain a list of options, which can be keys or key-value pairs,
- `(a)` is the name of this node so we can refer to it later,
- `at (0, 0)` describes where to place this node on the x-y plane,
- Everything inside the curly brackets `{}` is the node label which gets displayed, and can include both text and maths,
- We end each Ti*k*Z command with a semicolon `;`.

This produces the following:

Hello! $2 + 2 = 4$

Remember that we can also apply a global style to every node, like this:

```
\begin{tikzpicture}[every node/.style = {...}]
```

One new thing here is how we've described the fill colour. The syntax `red!10` creates a colour that is 10% red and 90% white. We can also write something like `red!30!blue`, which is 30% red and 70% blue. RGB, CMYK and HTML/hex colours are also available, you can find out more here.

Since we're working with coordinates, it's helpful to draw a grid in the background for reference:

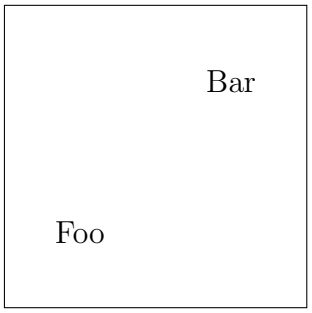
```
\draw[blue!10] (0, 0) grid (10, 6);
```

Of course, you can adjust the colour and intensity if you like.

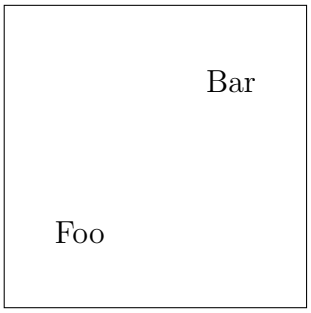
Ti*k*Z draws things in the order they appear in the source code, so we should put this right at the top of the `tikzpicture`. Otherwise, it will draw over the top of our graph.

Coordinates in Ti*k*Z use the normal mathematical definition (x increases from left to right, and y increases from bottom to top), unlike array coordinates where we think of $(0, 0)$ as being the top left. The coordinates are all *relative*, which means the actual numbers don't matter, only their relative positions. For example, we can create this same layout in two ways:

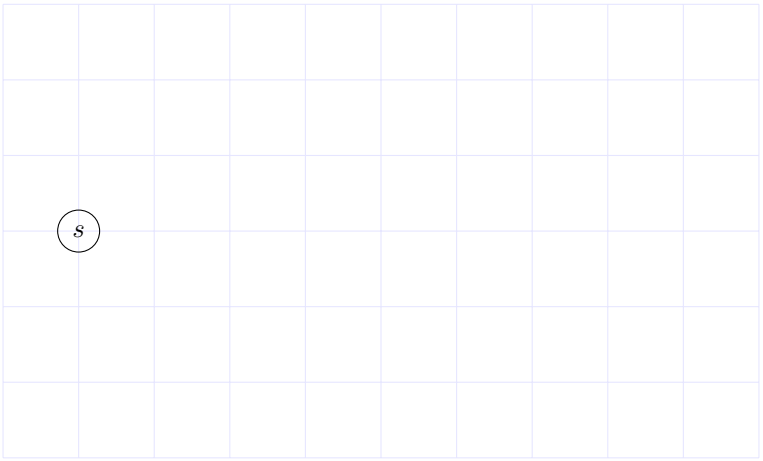
```
\draw (4, 4) rectangle (8, 8);
\node at (5, 5) {Foo};
\node at (7, 7) {Bar};
```



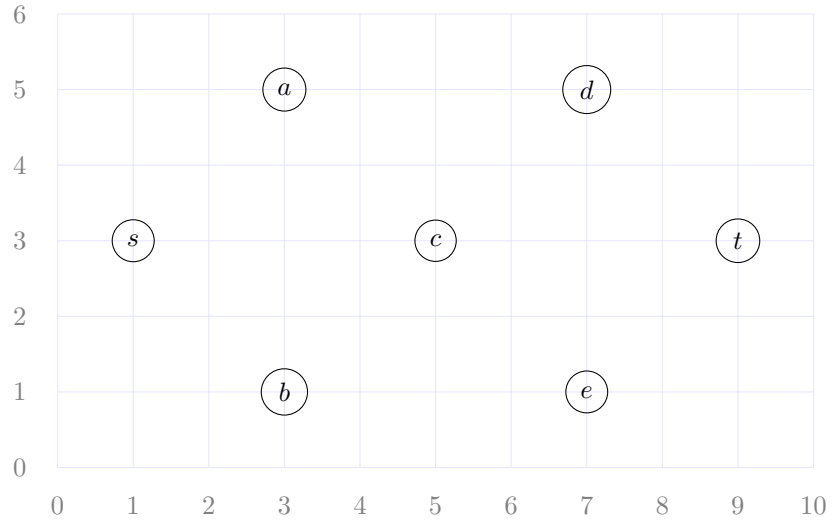

```
\draw (-1, -1) rectangle (3, 3);
\node at (0, 0) {Foo};
\node at (2, 2) {Bar};
```



▶ Draw a grid for reference, then add a node at $(1, 3)$ with the name `(s)` and the label s. This will be our source node. We want every node to be drawn as a circle - use a global node style to achieve this. You should end up with this:



▶ Then, add some more nodes to match the layout below (you don't need to create the axis numbers, they're just there to help):



You might notice that some nodes are larger than others - in particular, tall letters like b and d are a bit larger than smaller letters like s and c. To do this, we can add the `minimum size` option to our node style. I like to use `minimum size=2em`[1], but you can also use `pt` or `cm` if you like.

No minimum size Minimum size 2em

▶ Add a node style so that all of the nodes are the same size.

[1]The *em* is a unit equal to the current font size, and is used frequently in typography and web development.

Drawing Edges

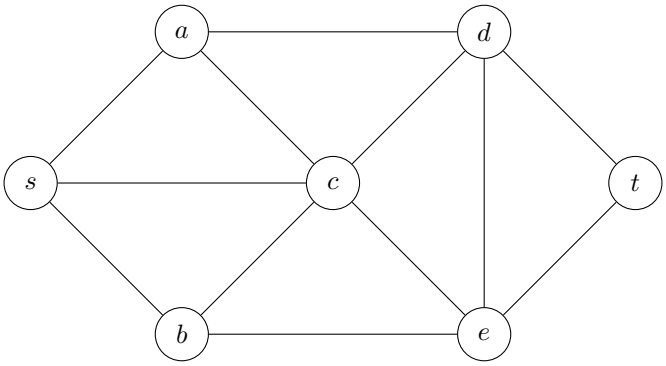
Now, let's join up our nodes using edges! To do this, we will use the `\draw` command. We've seen this before when we created our grid - the draw command has many uses. This is where we will use the node names:

```
\draw (s) edge (a);
```

We can chain these edge commands together to define edges with less writing, e.g.

```
\draw
    (s) edge (a)
        edge (b)
    (e) edge (t)
    ...
;
```

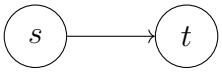
► Add edges to produce the graph shown below. We won't need the grid anymore, so remove that as well.



Adding Direction

We now want to turn this into a directed graph by adding arrows to our edges. All we have to do for this is add an arrow option to the `edge` command:

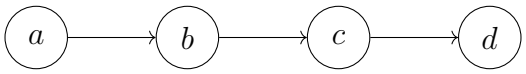
```
\draw (s) edge[->] (t);
```



If we reverse the `->` to `<-`, the arrow points the other way. We can use `<->` for a bidirectional edge.

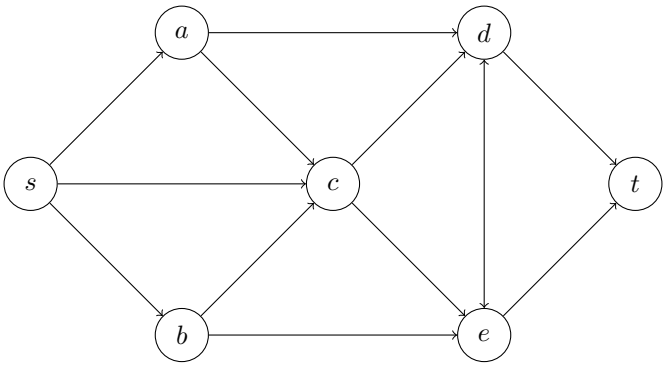
If we write all our edges in the same order (i.e. 'from' node on the left and 'to' node on the right), we can apply a single arrow option to the `\draw` command rather than to each edge individually:

```
\draw[->]
    (a) edge (b)
    (b) edge (c)
    (c) edge (d)
    (d) edge (a)
;
```



The `<` and `>` here actually refer to a specific arrow type. We can change them to create different arrow tips, for example `Square-Circle` or `Bar-Latex`. You can find a comprehensive list here.

▶ Add arrows to your graph to produce the following. Note the bidirectional edge $d \leftrightarrow e$; adding an arrow option to this individual edge will override the arrow in the `\draw` options.

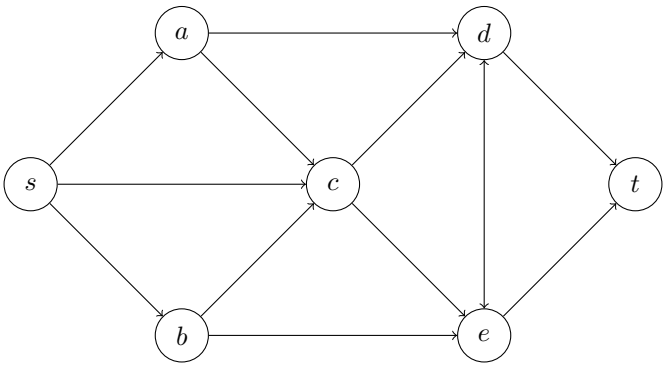


Bending Edges

Now, we'll start turning this into a flow network.

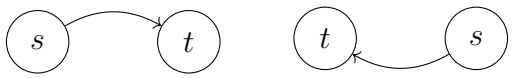
Recall that all edges in a flow network are directed, and that we represent a single bidirectional edge with two directed edges of the same capacity. Let's apply this to our edge $d \leftrightarrow e$:

```
...
(d) edge (e)
(e) edge (d)
...
```



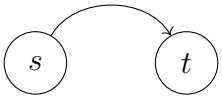
This did absolutely nothing! TikZ is indeed drawing two directed edges, but they overlap to appear as one edge. To fix this, we will bend the edges using the `bend left` and/or `bend right` options. Just like arrows, we specify these options on an edge. The direction of the bend is from the perspective of the 'from' node looking at the 'to' node. For example, these two edges have the same bend option (note that t and s are swapped on the right):

```
\draw (s) edge[->, bend left] (t);
```



We can also specify an angle, which is the angle at which the edge leaves the node:

```
\draw (s) edge[->, bend left=60] (t);
```

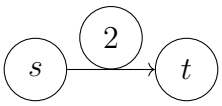


▶ Use the `bend` option to separate the two edges $d \to e$ and $e \to d$. To leave room for the capacities, use a bend angle of 10°.

Specifying Capacities

To add the edge capacities, we can add a node to each edge after the `edge` command, like this:

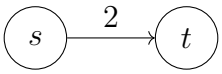
```
    \draw (s) edge[->] node[above] {2} (t);
```



We can of course add options to nodes on edges, and we'll usually add a position for the node relative to the edge (e.g. `above`, `left`, or `below right`). As you can see, the node style we defined earlier also got applied to the edge capacity node, which we don't want. Rather than removing the style and going back to manually adding options to each node, we will use *scopes* to restrict the style to a certain section of our code.

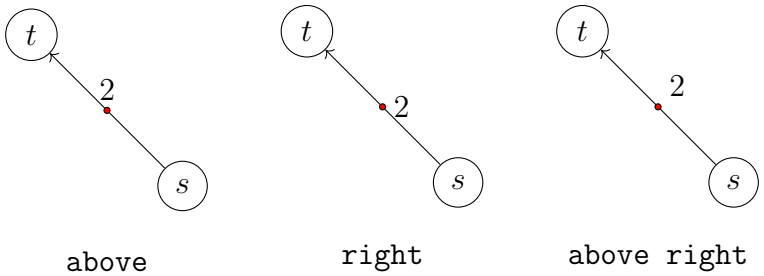
```
\begin{tikzpicture}
    % Scope for vertices
    \begin{scope}[every node/.style = {draw, circle, minimum size=2em}]
        \node (s) at (1, 1) {$s$};
        \node (t) at (3, 1) {$t$};
    \end{scope}

    % Scope for edges
    \begin{scope}
        \draw (s) edge[->] node[above] {2} (t);
    \end{scope}
\end{tikzpicture}
```

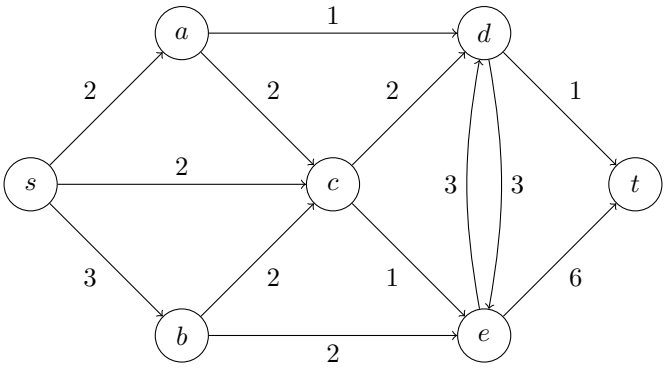


We can specify the `auto` option for our `scope` or `tikzpicture`, which will automatically place nodes on the left of edges, which can sometimes be helpful (but we don't want this here). What happens if we don't give a placement option?

We need to take particular care with diagonal edges - if we just specify `above` or `right` for the node placement, the result won't be quite right. To place the node near the middle of the edge, we need `above right`. Of course, we can also use `above left`, `below left` or `below right`. The node is placed relative to the midpoint of the line - in these examples, the midpoint is shown with a red dot.



- Add these edge capacities to your flow network, using the same node placements as below:

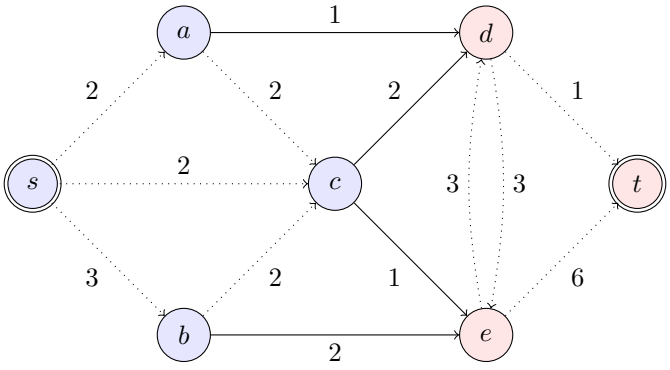


Showing the Min Cut

Our flow network is complete!

The network has a maximum flow of 6, with the minimum cut $S = \{s, a, b, c\}$ and $T = \{t, d, e\}$.

- Let's show this cut in the diagram:
  - Fill the nodes on the source side of the minimum cut with light blue,
  - Fill the nodes on the sink side of the minimum cut with light red,
  - Draw the edges across the cut with solid lines, and the others with dotted lines.
  - The source and sink are special, so we'll give them a double border.



We've seen how to do the first two. The last two are left for you to figure out!

Follow the steps given to produce a flow network using Ti*k*Z. Submit both the compiled PDF and your `.tex` code.

Produce a flow network in LaTeX using Ti*k*Z similar to the one shown. In particular, a good response has the following features:

- nodes are correctly placed and are the same size, with correct styles
- correct edges drawn with arrows, bends and dotted lines
- edge capacities are placed correctly
- global or scoped styles rather than individual options used where appropriate.

Your LaTeX/Ti*k*Z code should be well-formatted and readable, with good use of indentation and spacing.

Reminder. There is no `solution` environment in this template. This is intentional; you should *not* append the desired graph to this template file.

Instead, your PDF submission should be a `standalone` document, containing only the diagram.