How to TikZ?

An overview

Jan-Philipp Kappmeier

Technische Universität Berlin

13. 12. 2010

TikZ

TikZ ist *kein* Zeichenprogramm

- A language to create images within LaTex
- Two components:
  1. TikZ
     easy to use high-level commands
  2. PGF (Portable Graphics Format)
     low-level commands (at aleast: lower level)

Usage

First: include the package tikz. Then:

- directly code images. See details in this talk
- create images in Inkscape and export
- lots of other tools provide TikZ-Output

For starters

How to add a TikZ picture to your document?

- TikZ-code is written in an `tikzpicture` environment.
- Put into a picture environment to add caption, reference etc.
- Inline-TikZ: use the `\tikz` command to create inline graphics like this nice 5-wheel here.

Example

It is easy to draw a thistle .

```
\tikz{ \filldraw[color=Thistle] circle (0.5ex); }
```

Drawing on paths

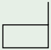
Generel principle

- Everything is drawn on a so-called path
- A sequence of coordinates and drawing commands
- General syntax:

  `\draw[options] (coordinate) command (coordinate) ...;`

  like moving a pencil to some place and start drawing something.
- Special commands like `\fill` also exists.

Example



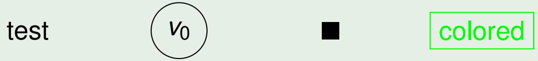
```
\draw (1,0) rectangle +(2,1) -- (3,2);
\draw (0,0) -- (0,2) -- (1,3.25){[rounded corners] -- (2,2)
  -- (2,0)} -- (0,2) -- (2,2) -- (0,0) -- (2,0);
```

Drawing Graphs

Nodes

- Nodes can be put on any path using the command `node`
- `\draw[parameter] node at coordinate {content};`
- Special draw command for nodes: `\node`
- by default, a node is just a position and no shape is drawn

Example




```
\node at (0,0) {test};
\node[draw,circle] at (2,0) {$v_0$};
\node[fill] at (4,0) {};
\node[draw,color=red] at (6,0) [green] {colored};
```

Drawing the 5 wheel

We are ready to pimp our slides with the COGA-5-Wheel!

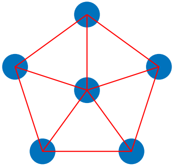
Drawing the 5 wheel

We are ready to pimp our slides with the COGA-5-Wheel!

```
\node[fill,circle,draw,RoyalBlue] at (0,1) {};
\node[fill,circle,draw,RoyalBlue] at (-0.9511,0.3091) {};
\node[fill,circle,draw,RoyalBlue] at (-0.5878,-0.8091) {};
\node[fill,circle,draw,RoyalBlue] at (0.5878,-0.8091) {};
\node[fill,circle,draw,RoyalBlue] at (0.9511,0.3091) {};
\node[fill,circle,draw,RoyalBlue] at (0,0) {};
\draw[red] (0,1) to (-0.9511,0.3091) to (-0.5878,-0.8091)
    to (0.5878,-0.8091) to (0.9511,0.3091) to (0,1);
\draw[red] (0,0) to (0,1) (0,0) to (-0.9511,0.3091) (0,0)
    to (-0.5878,-0.8091) (0,0) to (0.5878,-0.8091) (0,0)
    to (0.9511,0.3091);
```

Drawing the 5 wheel

We are ready to pimp our slides with the COGA-5-Wheel!



Unfortunately we have do to a lot of computation.
➡ Can be done by PGF!

Also, lines are drawn over the nodes.
➡ Solve this using named nodes.

Drawing the 5 wheel

Computations using PGF

- `\pgfmathsetmacro{\x}{computation}`
  Creates a variable `\x` with the result of the computation
- `\pgfmathparse{computation}`
  Stores the result in the variable `\pgfmathresult`

Drawing the 5 wheel

Computations using PGF

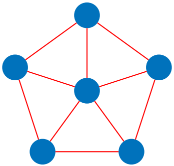
- `\pgfmathsetmacro{\x}{computation}`
  Creates a variable `\x` with the result of the computation
- `\pgfmathparse{computation}`
  Stores the result in the variable `\pgfmathresult`

```
\pgfmathsetmacro{\xa}{cos(90)}
\pgfmathsetmacro{\ya}{sin(90)}
\pgfmathsetmacro{\xb}{cos(90+72)}
\pgfmathsetmacro{\yb}{sin(90+72)}
...
\node[fill,circle,draw,RoyalBlue] (1) at (\xa,\ya) {};
\node[fill,circle,draw,RoyalBlue] (2) at (\xb,\yb) {};
...
\draw[red] (1) to (2) to (3) to (4) to (5) to (1);
\draw[red] (0) to (1) (0) to (2) (0) to (3) (0) to (4) (0) to
```

Drawing the 5 wheel

Computations using PGF

- `\pgfmathsetmacro{\x}{computation}`
  Creates a variable `\x` with the result of the computation
- `\pgfmathparse{computation}`
  Stores the result in the variable `\pgfmathresult`



Still we need to know a lot about how to compute coordinates on a circle

Polar coordinates

Polar coordinates

- All coordinates can be defined via polar coordinates
- Needed only an angle and the radius (distnace from the origin)
- expressed as `(angle:radius)`

Polar coordinates

Polar coordinates

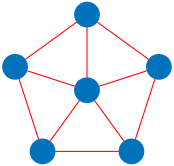
- All coordinates can be defined via polar coordinates
- Needed only an angle and the radius (distnace from the origin)
- expressed as `(angle:radius)`

```
\node[fill,circle,draw,RoyalBlue] (1) at (90+0*72:1) {};
\node[fill,circle,draw,RoyalBlue] (2) at (90+1*72:1) {};
\node[fill,circle,draw,RoyalBlue] (3) at (90+2*72:1) {};
\node[fill,circle,draw,RoyalBlue] (4) at (90+3*72:1) {};
\node[fill,circle,draw,RoyalBlue] (5) at (90+4*72:1) {};
\node[fill,circle,draw,RoyalBlue] (0) at (0,0) {};
\draw[red] (1) to (2) to (3) to (4) to (5) to (1);
\draw[red] (0) to (1) (0) to (2) (0) to (3) (0) to (4) (0) to
```

Polar coordinates

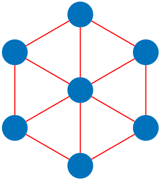
Polar coordinates

- All coordinates can be defined via polar coordinates
- Needed only an angle and the radius (distnace from the origin)
- expressed as `(angle:radius)`



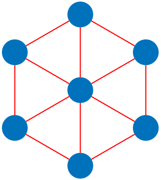
Loops

What happens when the logo surprisingly changes to a 6 wheel?

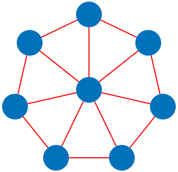


Loops

What happens when the logo surprisingly changes to a 6 wheel?

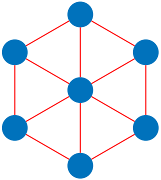


Or even to a 7 wheel?

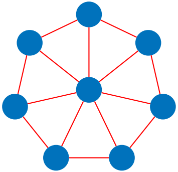


Loops

What happens when the logo surprisingly changes to a 6 wheel?



Or even to a 7 wheel?



The foreach command

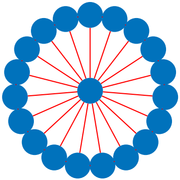
- executes the same commands for all items of a given set
- assigns the value to a variable
- `\foreach \var in \{ item1, item2, ..., itemN\} { }`

Putting things together

```
\pgfmathsetmacro{\n}{5}
\pgfmathtruncatemacro{\nodes}{\n-1}
\node[fill,circle,draw,RoyalBlue] (c) at (0,0) {};
\foreach \i in {0,...,\nodes}
  \node[fill,circle,draw,RoyalBlue] (\i) at (90+\i*360/\n:1)
\foreach \i in {0,...,\nodes} {
  \draw[red] (c) to (\i);
  \pgfmathtruncatemacro{\j}{mod(round(1+\i),\n)}
  \draw[red] (\i) -- (\j);
}
```

Putting things together

```
\pgfmathsetmacro{\n}{5}
\pgfmathtruncatemacro{\nodes}{\n-1}
\node[fill,circle,draw,RoyalBlue] (c) at (0,0) {};
\foreach \i in {0,...,\nodes}
  \node[fill,circle,draw,RoyalBlue] (\i) at (90+\i*360/\n:1)
\foreach \i in {0,...,\nodes} {
  \draw[red] (c) to (\i);
  \pgfmathtruncatemacro{\j}{mod(round(1+\i),\n)}
  \draw[red] (\i) -- (\j);
}
```



Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.
Give it a first try:

```
\draw[->] (0,0) to (8,0);
\foreach \x in {0, 1, 1.57, 3.14, 2.71} {
  \draw (\x,0.1) to (\x,-0.1);
  \node at (\x, -0.3) {\footnotesize{\x}};
```

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.
Give it a first try:

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.
Better:

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.

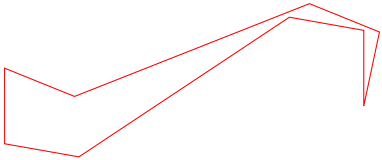
```
\draw[->] (0,0) to (4,0);
\foreach \x / \txt in
{0, 1, 1.57 / $\frac{\pi}{2}$, 3.14 / $\pi$, 2.71 / $e$} {
  \draw (\x,0.1) to (\x,-0.1);
  \node at (\x, -0.3) {\footnotesize{\txt}};
```

Calculate coordinates

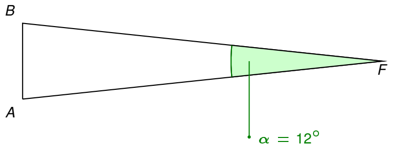
Coordinates

- Can be defined using `coordinate`
- Like nodes with empty text
- Coordinates can be computed (like vector math)
- need to add the package calc (`\includetikzpackage{calc}`)

An example using coordinate calculation - Background



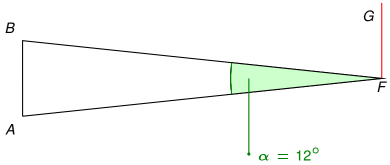
An example using coordinate calculation - Background



Construction

1. Start with an isosceles triangle with base of length a

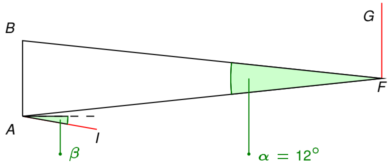
An example using coordinate calculation - Background



Construction

1. Start with an isosceles triangle with base of length *a*
2. Draw a copy of $\overline{AB}$ at *F*

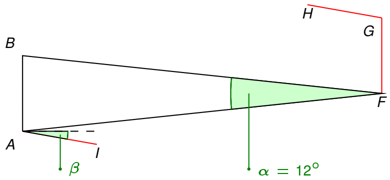
An example using coordinate calculation - Background



Construction

1. Start with an isosceles triangle with base of length a
2. Draw a copy of $\overline{AB}$ at F
3. Draw segment $\overline{AI}$ of length b with some angle β

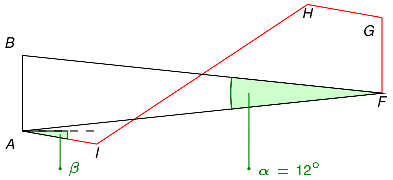
An example using coordinate calculation - Background



Construction

1. Start with an isosceles triangle with base of length a
2. Draw a copy of $\overline{AB}$ at F
3. Draw segment $\overline{AI}$ of length b with some angle β
4. and copy it to G

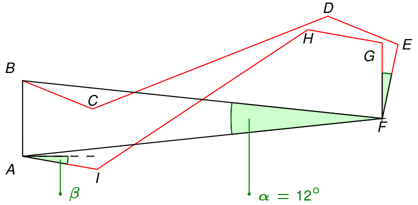
An example using coordinate calculation - Background



Construction

1. Start with an isosceles triangle with base of length a
2. Draw a copy of $\overline{AB}$ at F
3. Draw segment $\overline{AI}$ of length b with some angle β
4. and copy it to G
5. Connect H and I

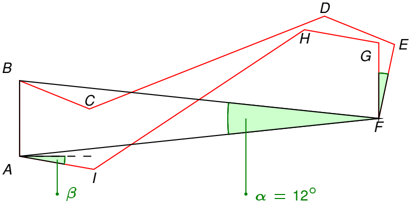
An example using coordinate calculation - Background



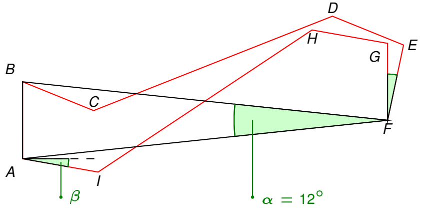
Construction

1. Start with an isosceles triangle with base of length a
2. Draw a copy of $\overline{AB}$ at F
3. Draw segment $\overline{AI}$ of length b with some angle β
4. and copy it to G
5. Connect H and I
6. Rotate the polygonal path $\overline{FGHIA}$ around A by $12°$

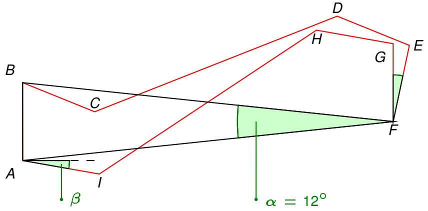
An example using coordinate calculation - Ti*k*Z realization



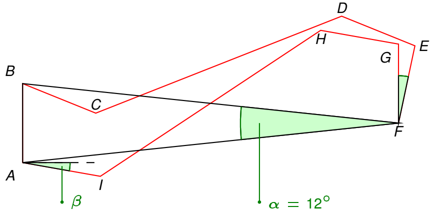
An example using coordinate calculation - Ti*k*Z realization



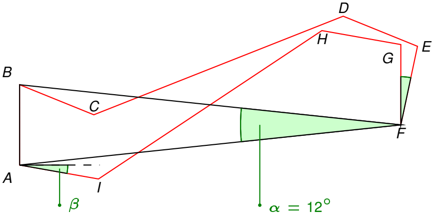
An example using coordinate calculation - Ti*k*Z realization



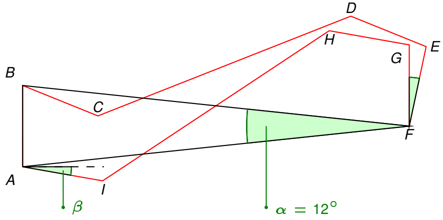
An example using coordinate calculation - TikZ realization



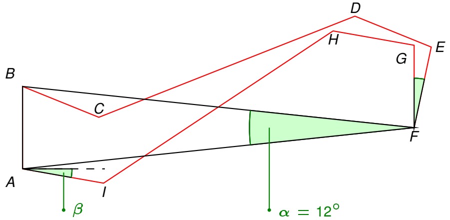
An example using coordinate calculation - Ti*k*Z realization



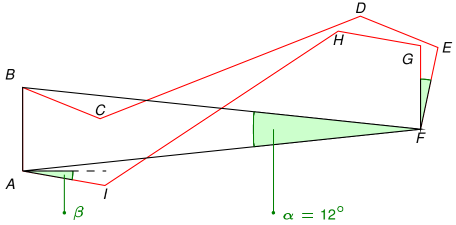
An example using coordinate calculation - Ti*k*Z realization



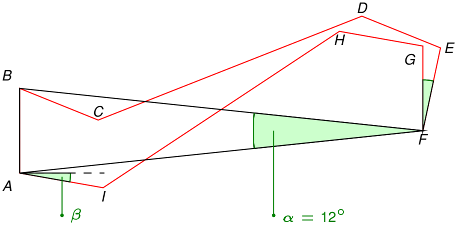
An example using coordinate calculation - Ti*k*Z realization



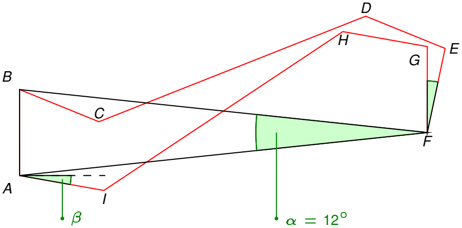
An example using coordinate calculation - Ti*k*Z realization



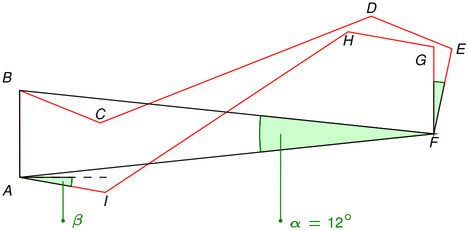
An example using coordinate calculation - Ti*k*Z realization



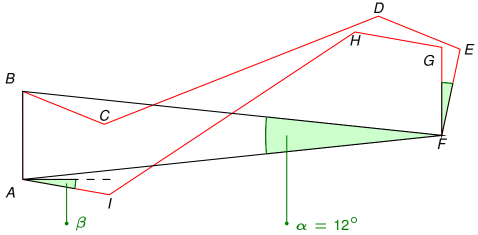
An example using coordinate calculation - Ti*k*Z realization



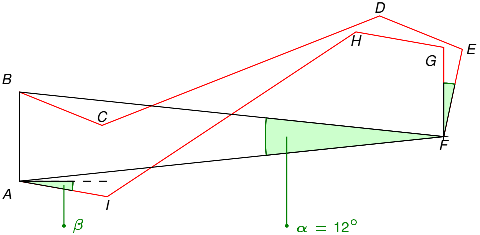
An example using coordinate calculation - TikZ realization



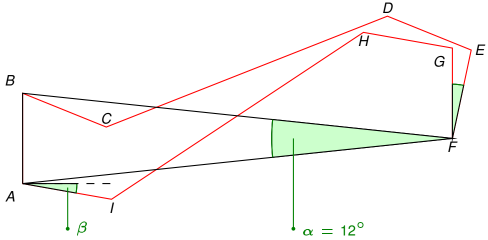
An example using coordinate calculation - Ti*k*Z realization



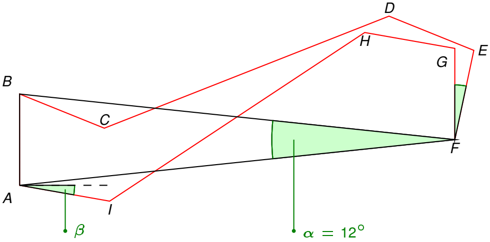
An example using coordinate calculation - Ti*k*Z realization



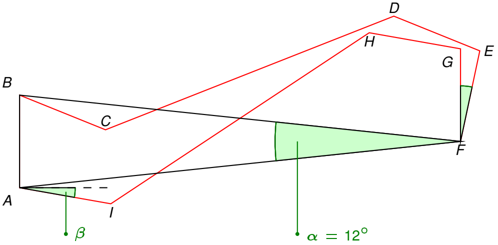
An example using coordinate calculation - Ti*k*Z realization



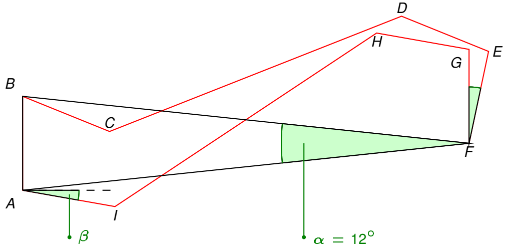
An example using coordinate calculation - Ti*k*Z realization



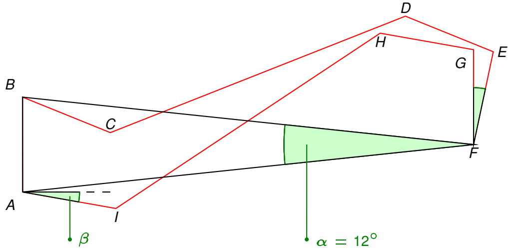
An example using coordinate calculation - Ti*k*Z realization



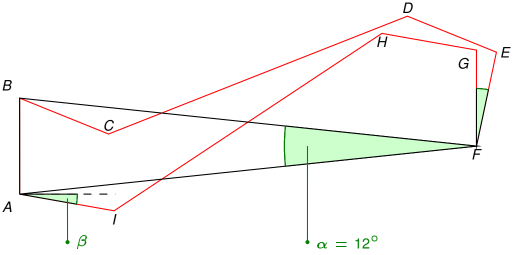
An example using coordinate calculation - TikZ realization



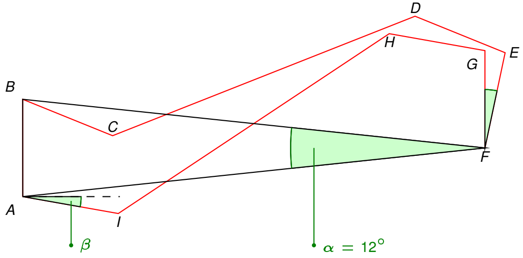
An example using coordinate calculation - Ti*k*Z realization



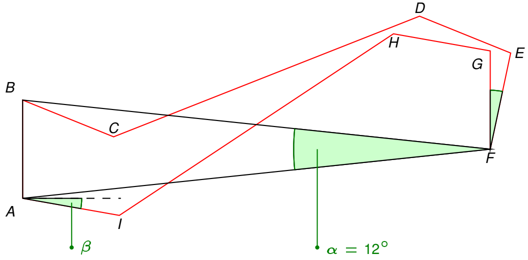
An example using coordinate calculation - Ti*k*Z realization



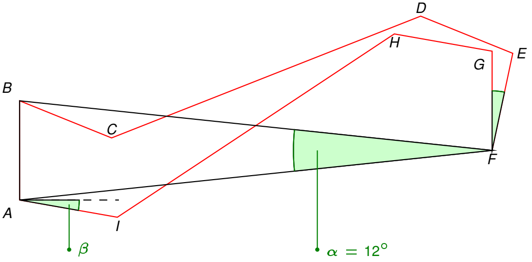
An example using coordinate calculation - Ti*k*Z realization



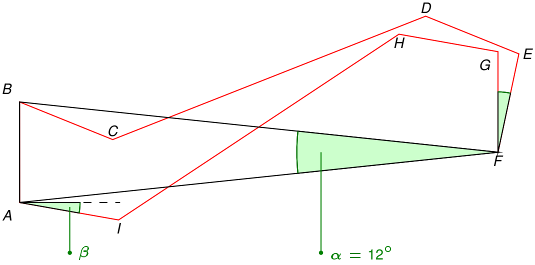
An example using coordinate calculation - Ti*k*Z realization



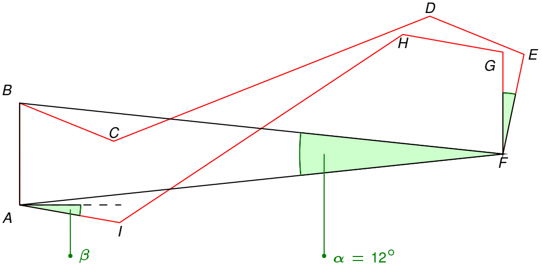
An example using coordinate calculation - Ti*k*Z realization



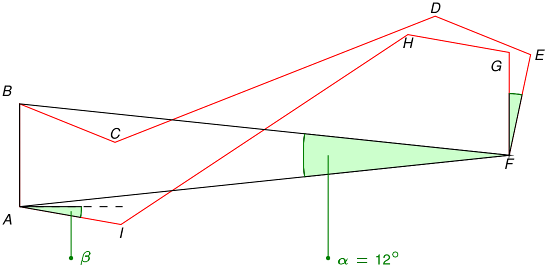
An example using coordinate calculation - Ti*k*Z realization



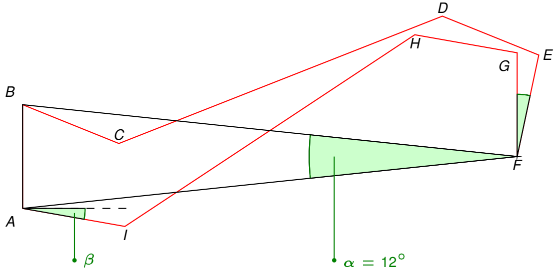
An example using coordinate calculation - Ti*k*Z realization



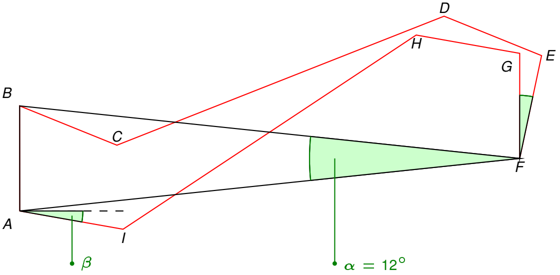
An example using coordinate calculation - Ti*k*Z realization



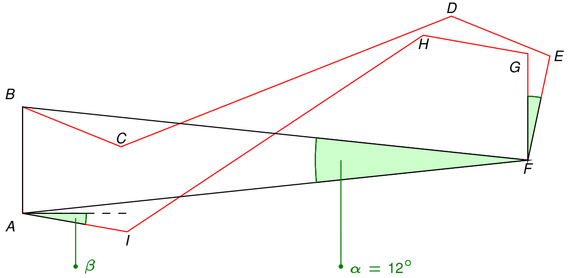
An example using coordinate calculation - Ti*k*Z realization



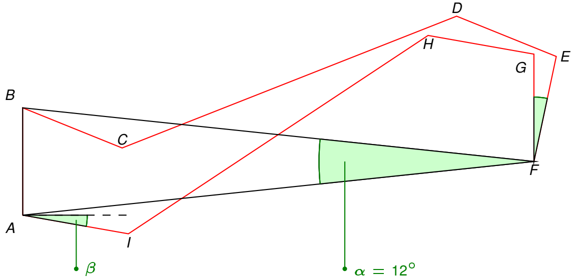
An example using coordinate calculation - Ti*k*Z realization



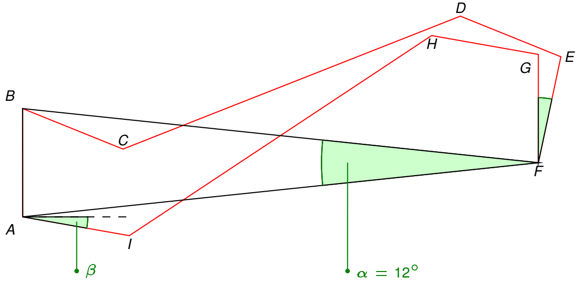
An example using coordinate calculation - Ti*k*Z realization



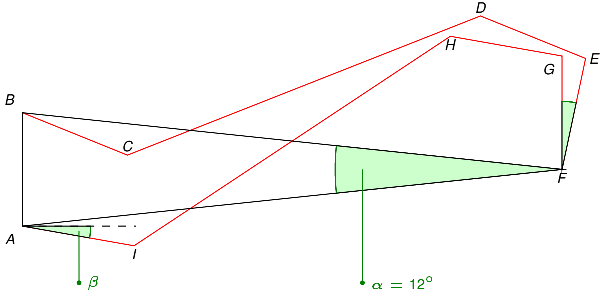
An example using coordinate calculation - Ti*k*Z realization



An example using coordinate calculation - TikZ realization

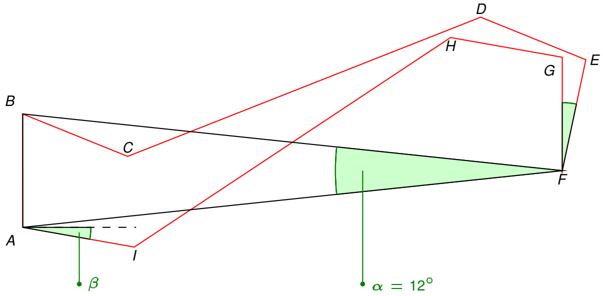


An example using coordinate calculation - Ti*k*Z realization



$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

An example using coordinate calculation - TikZ realization



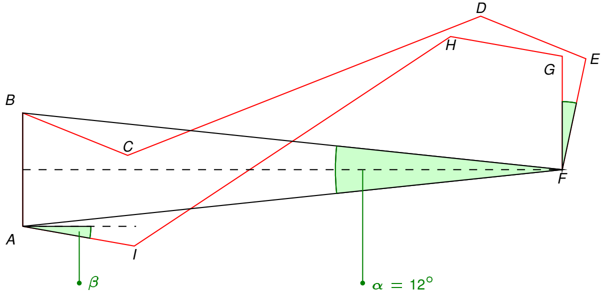
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Define Variables to use:

```
\def\a{0.5}
\def\b{0.5}
\def\bAngle{-10}
```

An example using coordinate calculation - Ti*k*Z realization



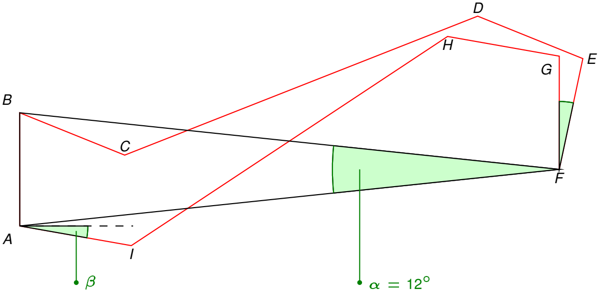
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute Distance of Point F from $\overline{AB}$:

```
\pgfmathsetmacro{\hyp}{\a*0.5 / cos(84)}
\pgfmathsetmacro{\len}{sqrt(\hyp*\hyp-0.25*\a*\a)}
```

An example using coordinate calculation - TikZ realization



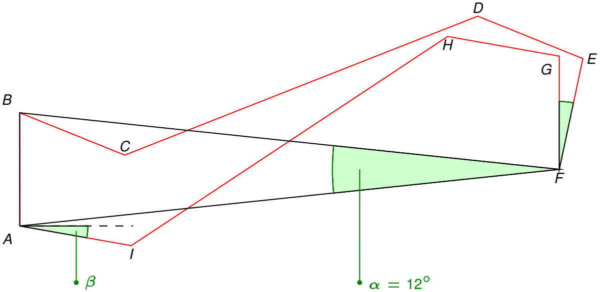
$$\boxed{\begin{aligned} a &= \overline{AB} = \overline{FG} \\ b &= \overline{AI} = \overline{HG} \end{aligned}}$$

Example

Compute the locations of the points:

```
\coordinate (A) at (0,0);        % start coordinate
\coordinate (B) at ($ (A) + (0,\a) $);
\coordinate (F) at ($ (B) + (\len,0.5*\a) $);
\coordinate (G) at ($ (F) + (0,\a) $);
```

An example using coordinate calculation - Ti*k*Z realization



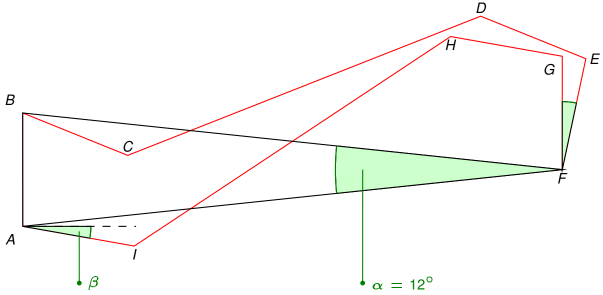
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute the locations of the points:

```
\coordinate (Htemp) at ($ (G) - (\b,0) $);
\coordinate (H) at ($ (G)!1!\bAngle:(Htemp) $);
\coordinate (Itemp) at ( $ (A) + (\b,0)$ );
\coordinate (I) at ($ (A)!1!\bAngle:(Itemp) $);
```

An example using coordinate calculation - Ti*k*Z realization



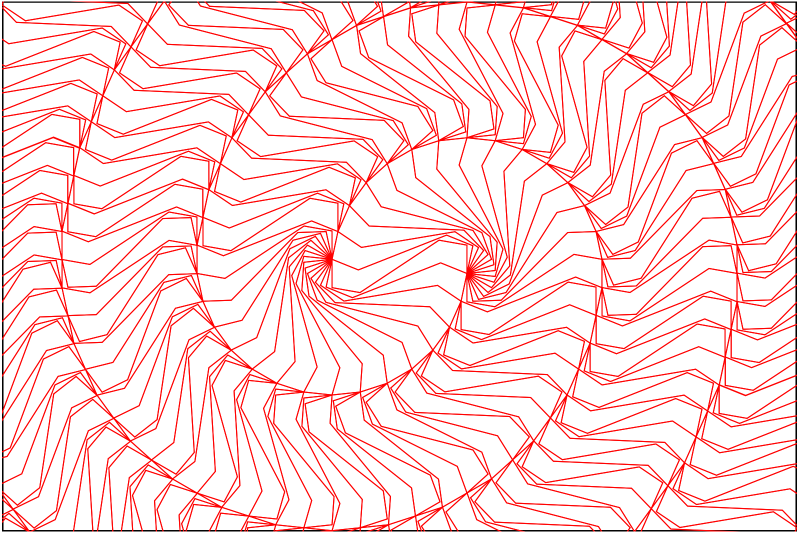
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute the locations of the points:

```
\coordinate (E) at ($ (F)!1!-12:(G) $);
\coordinate (D) at ($ (F)!1!-12:(H) $);
\coordinate (C) at ($ (F)!1!-12:(I) $);
```

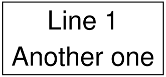
The complete example



Node features

- Multiline nodes: Allows to have several lines of text within one node

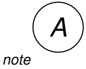
```
\node[align=center] {Line 1 \\ Another line};
```



Only works with TikZ 2.10

- Node labels: can add a label to all corners of the node

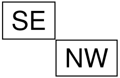
```
\node at (0,0) [label=below left:\tiny{$note$}] {A};
```



Node features (cont.)

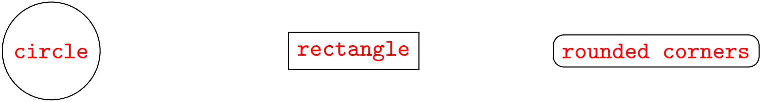
- Anchors: defines the corner of the node that lies at the specifies position. Default is `center`.

```
\node[draw,anchor=north west] at (0,0) {NW};
\node[draw,anchor=south east] at (0,0) {SE};
```



- Shapes: The outer appearance of a drawn node.

```
\node[circle] at (0,0) {ellipse};
```



Creating own shapes is possible, but not easy. Needs low level coding of so-called basic layer.

If you still miss a powerpoint feature

Do you really miss anything?

If you still miss a powerpoint feature

Do you really miss anything?

If you really, really need, just do, it's easy!

If you still miss a powerpoint feature

Do you really miss anything?

If you really, really need, just do, it's easy!

```
\node[starburst,fill=yellow,draw=red,line width=2pt] at
```

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`cloud callout` – the shape name

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`cloud puffs` – the number of puffs of the cloud

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`cloud puff arc` – the angle of two meeting puffs

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`callout pointer segments` – the number of round bubbles

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`callout relative pointer` – the angle and distance of the pointer

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`aspect` – ratio between width and height

The cloud code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds };
```

`current page.center` – *absolute* coordinate on the page

Scopes in Ti*k*Z

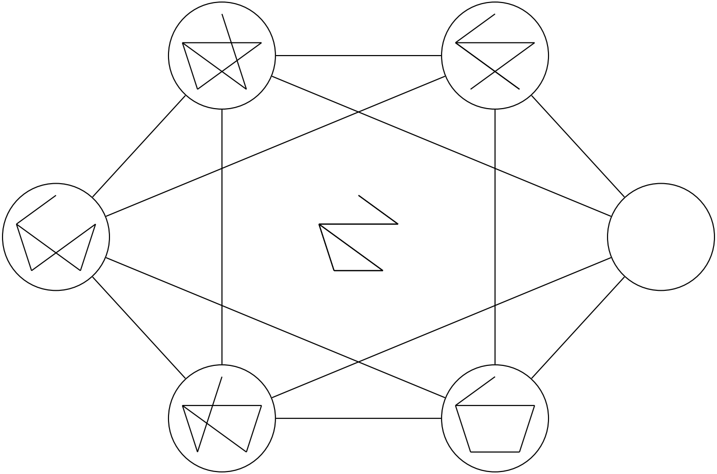
- Ti*k*Z allows scopes, just like e. g. JAVA
- Scopes can alter the drawing projection
- That means rotating, moving or scaling etc.

Possible commands are:

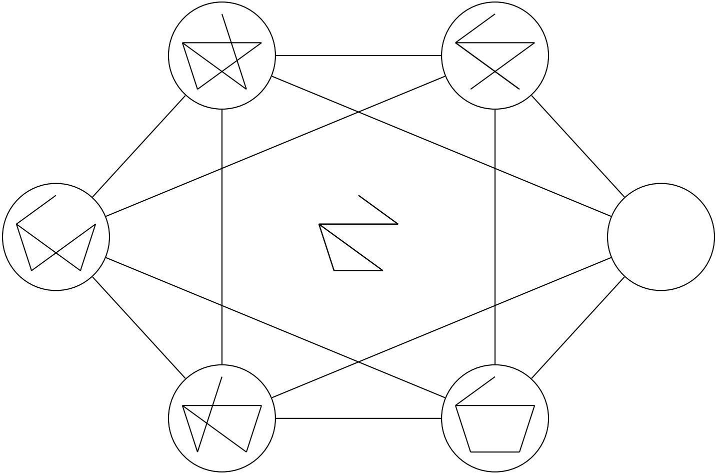
- `xscale`, `yscale` and `scale` for scaling
- `rotate` for rotation around an angle
- `xshift`, `yshift` and `shift` for movements of the origin

```
\begin{scope}[rotate=30,xscale=0.5,shift={(0:\s)}]
...
\end{scope}
```

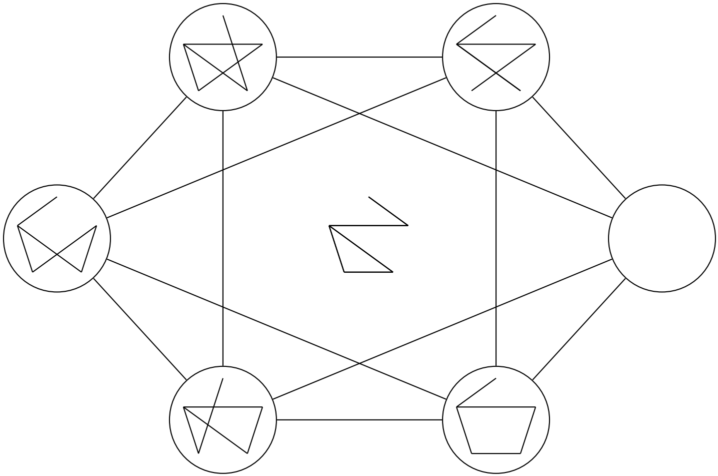
Example for scopes



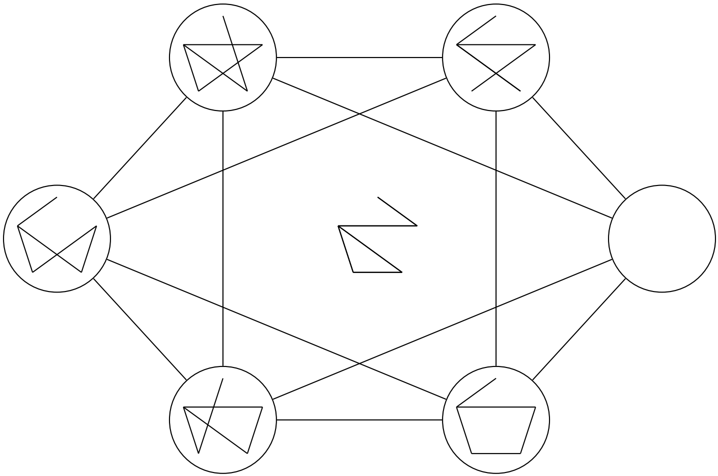
Example for scopes



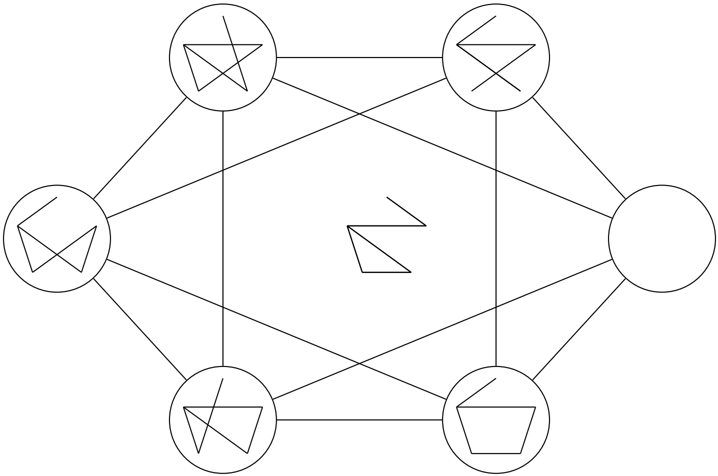
Example for scopes



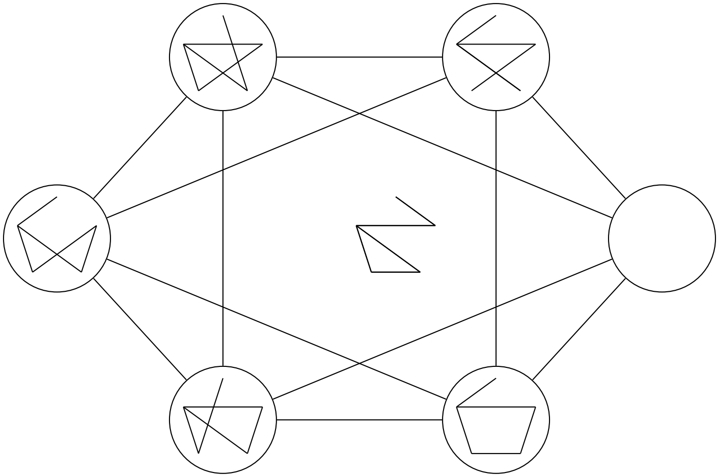
Example for scopes



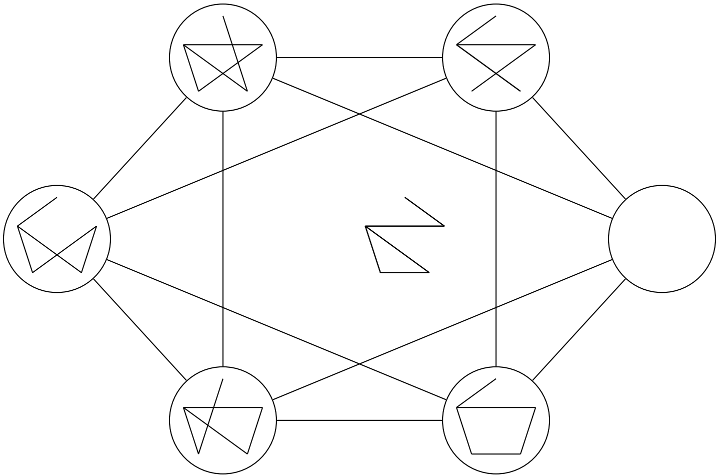
Example for scopes



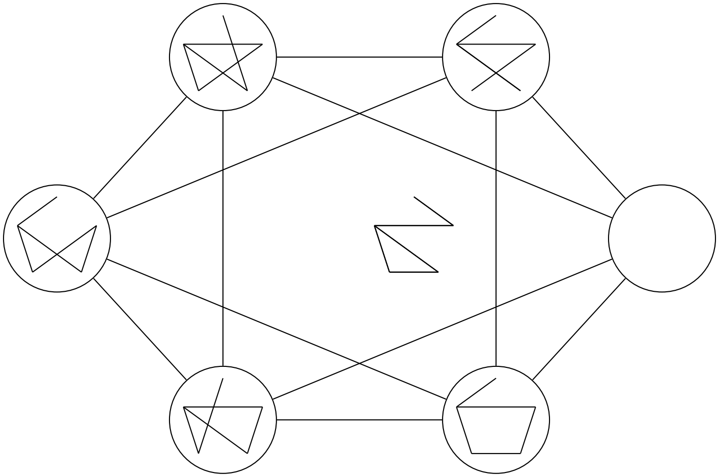
Example for scopes



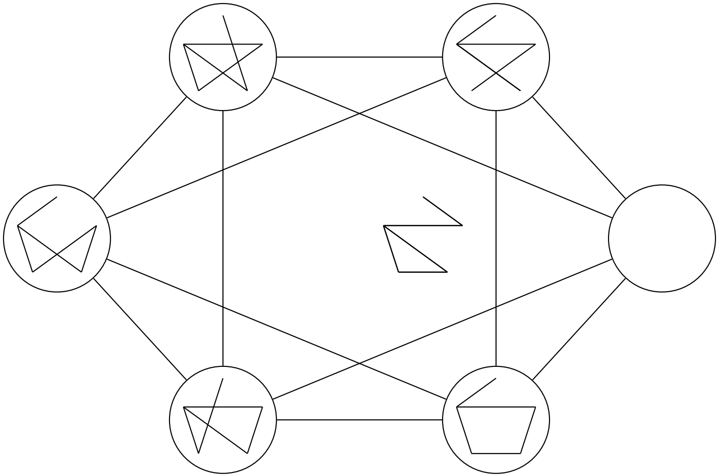
Example for scopes



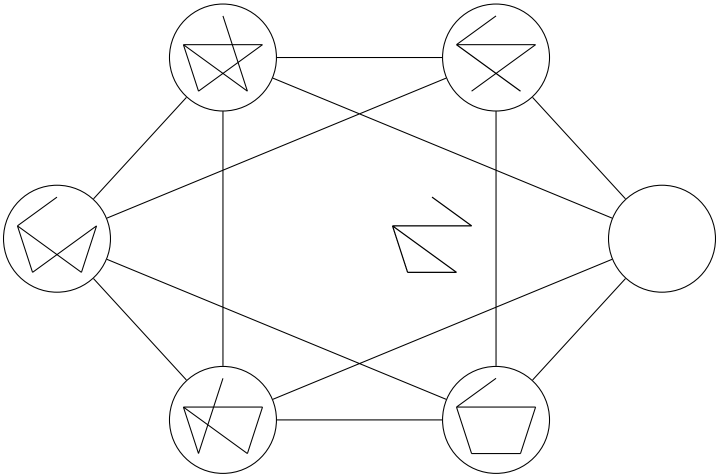
Example for scopes



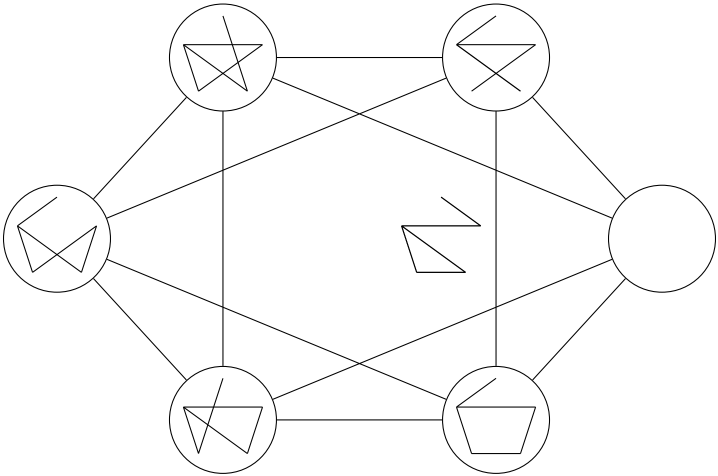
Example for scopes



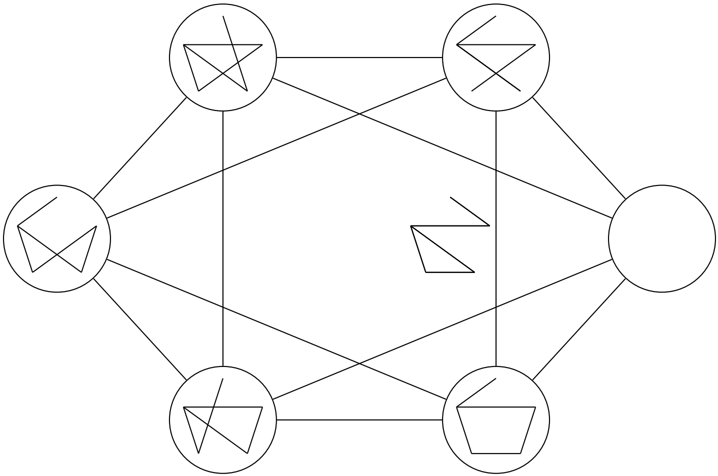
Example for scopes



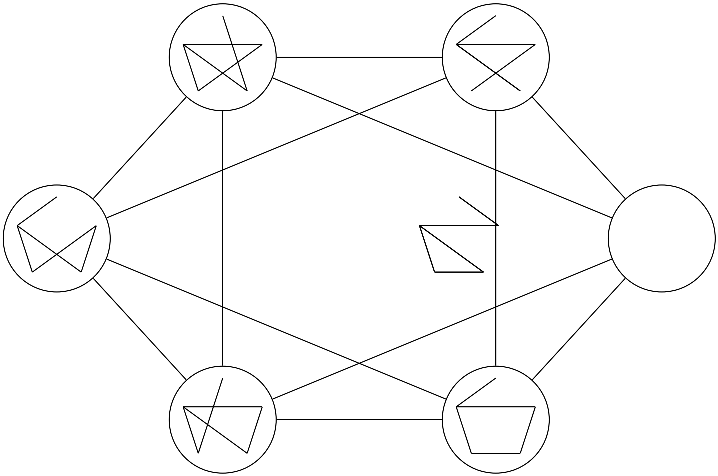
Example for scopes



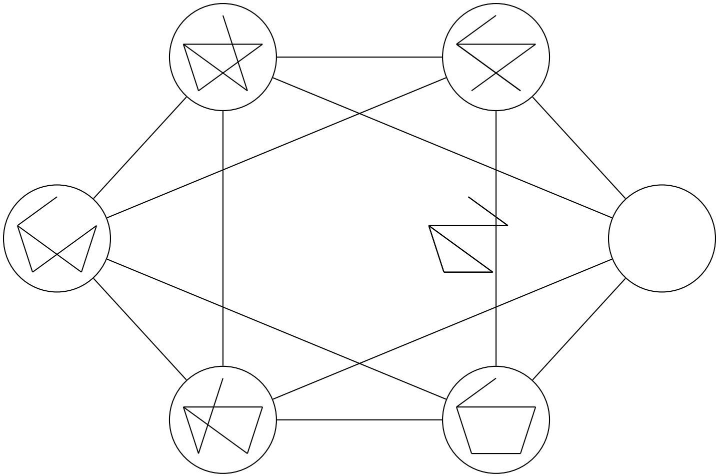
Example for scopes



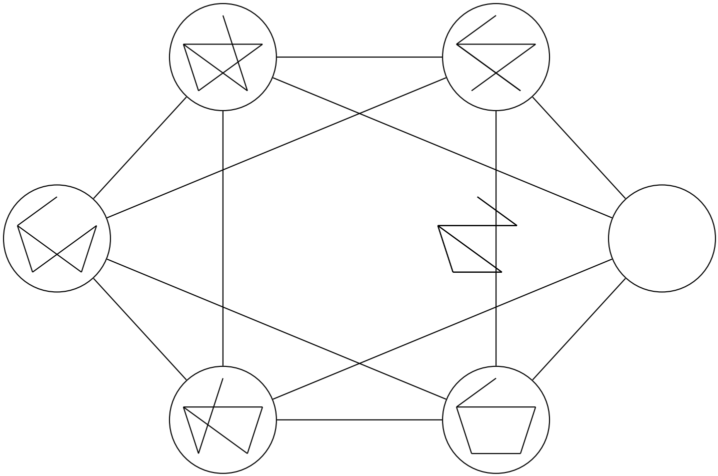
Example for scopes



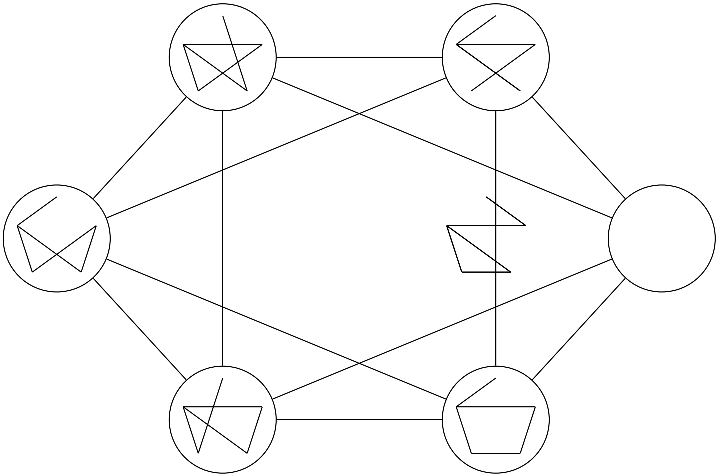
Example for scopes



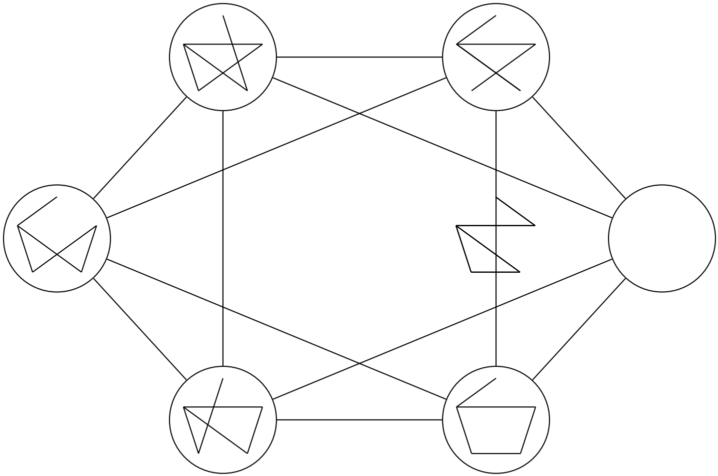
Example for scopes



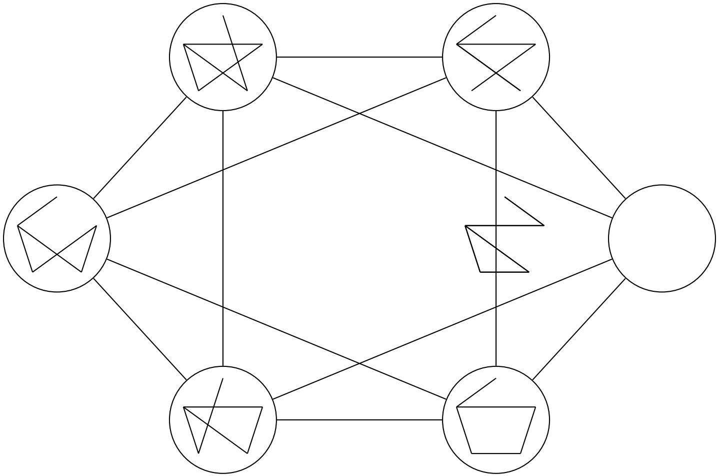
Example for scopes



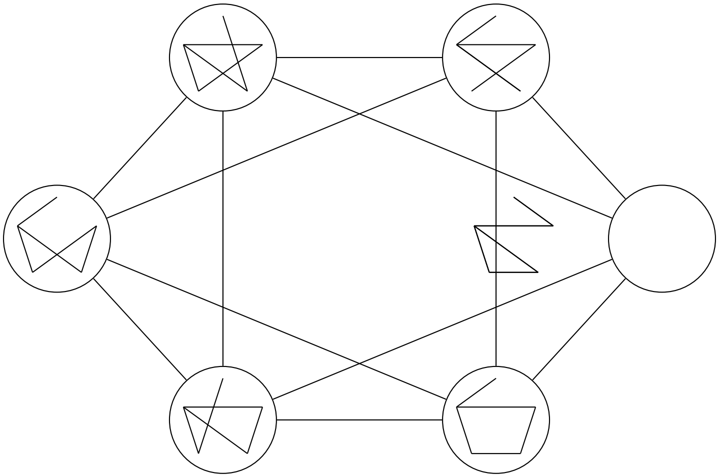
Example for scopes



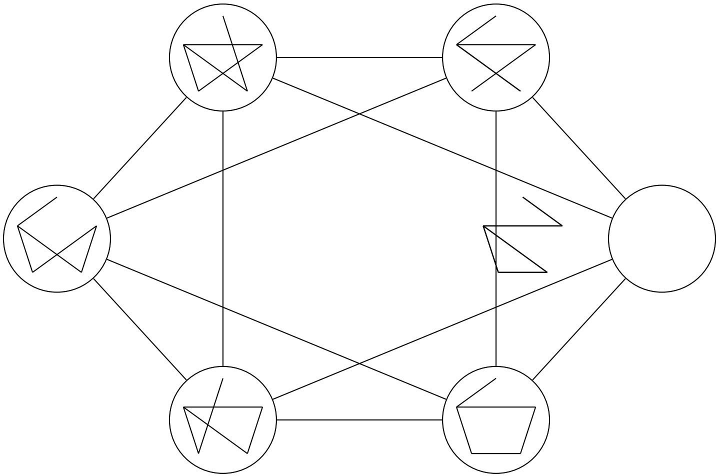
Example for scopes



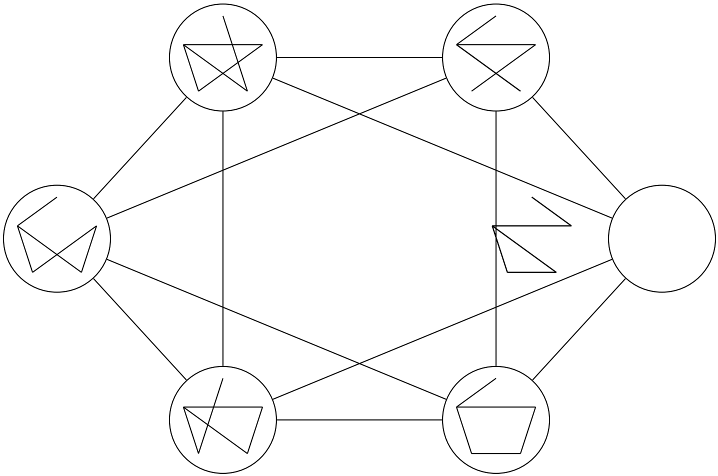
Example for scopes



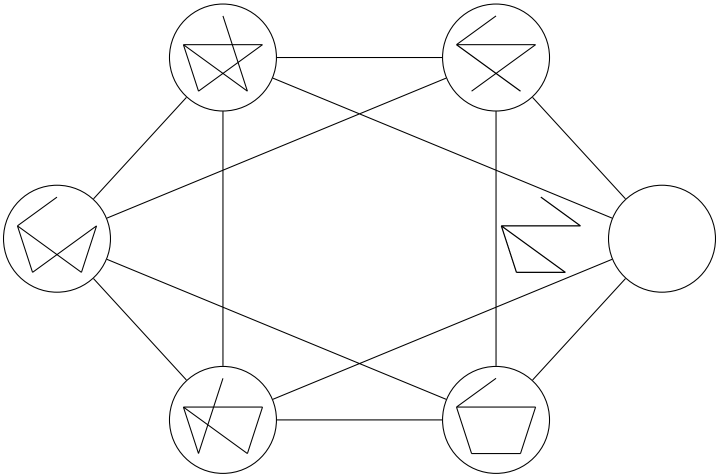
Example for scopes



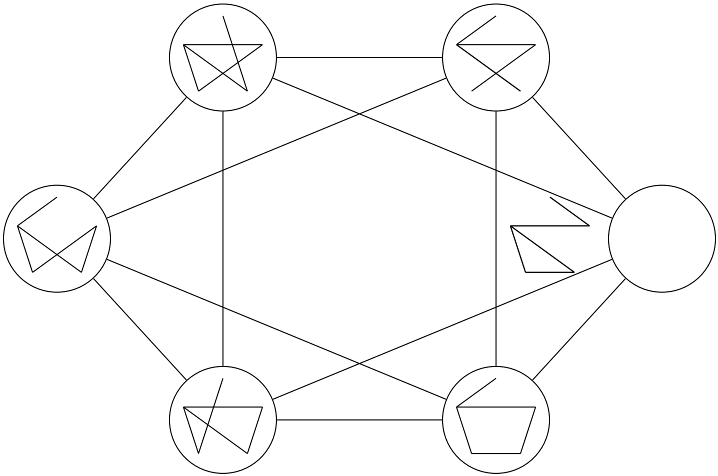
Example for scopes



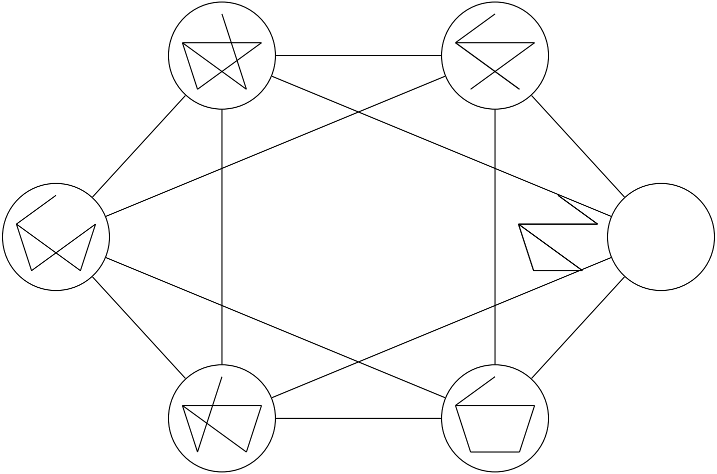
Example for scopes



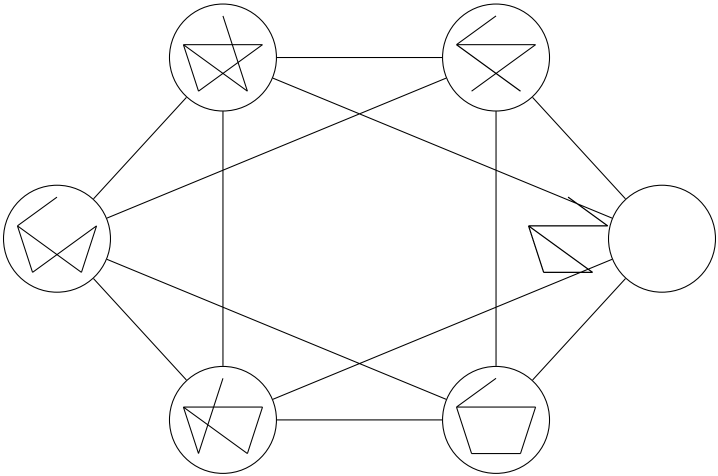
Example for scopes



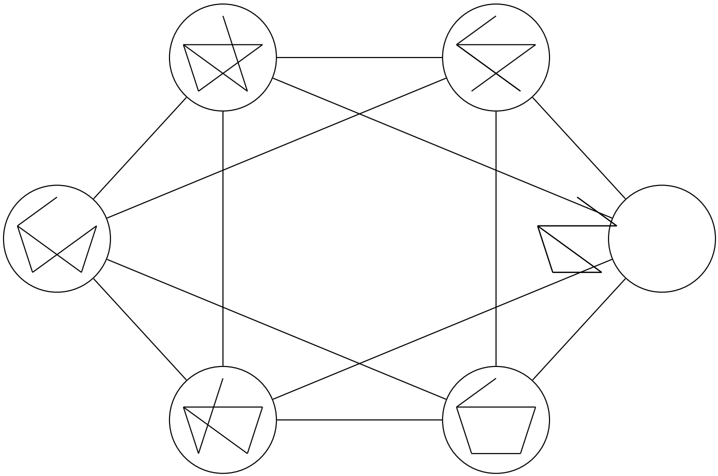
Example for scopes



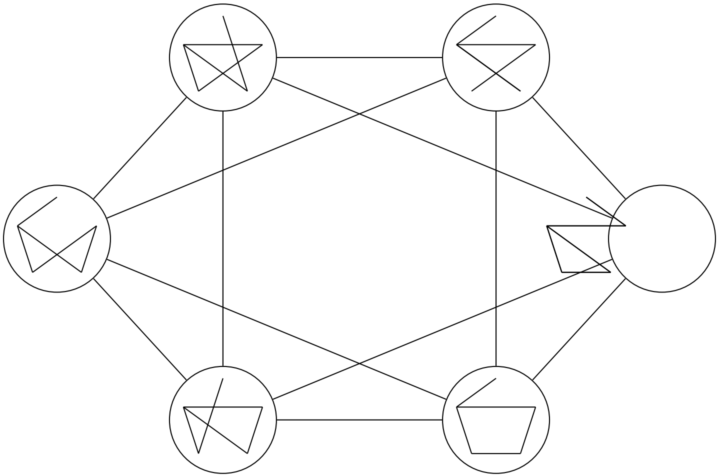
Example for scopes



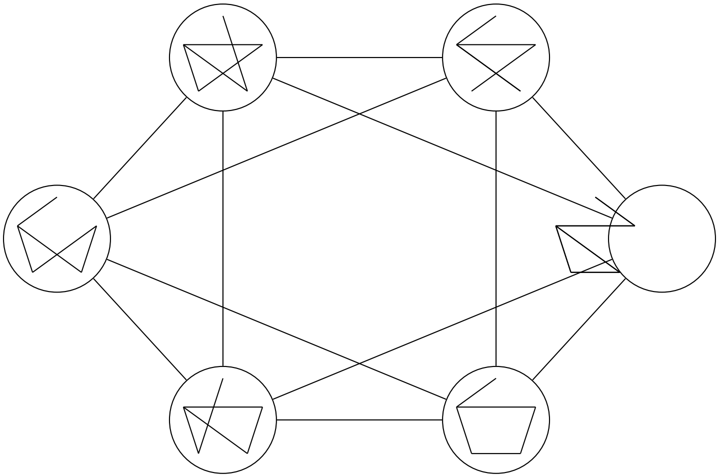
Example for scopes



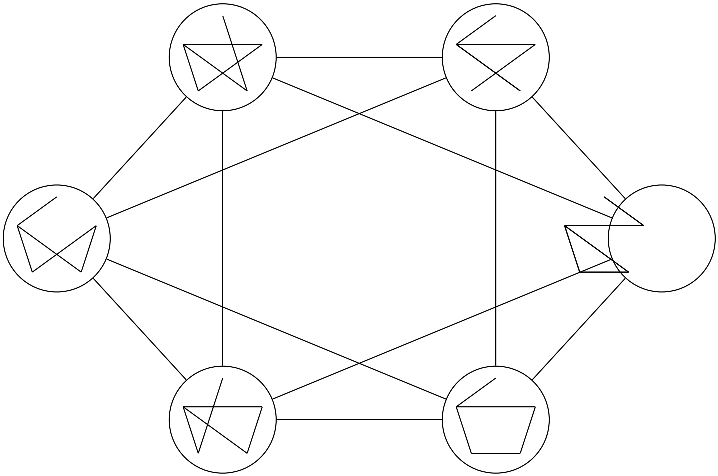
Example for scopes



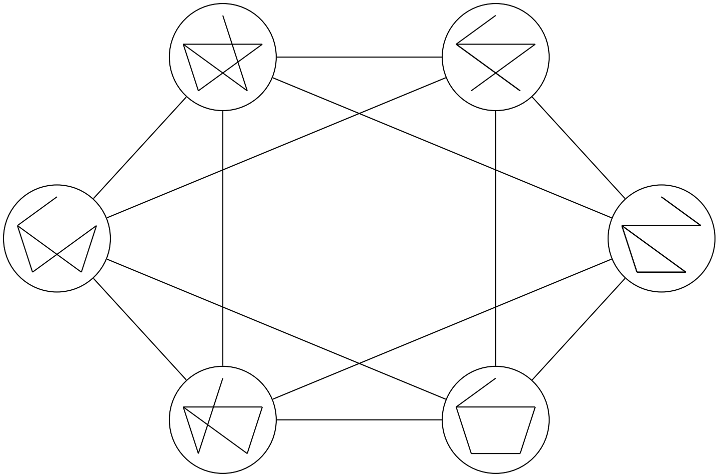
Example for scopes



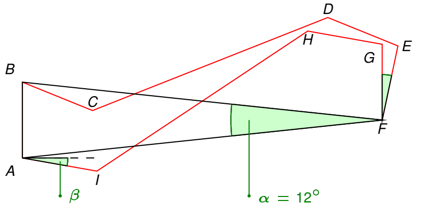
Example for scopes



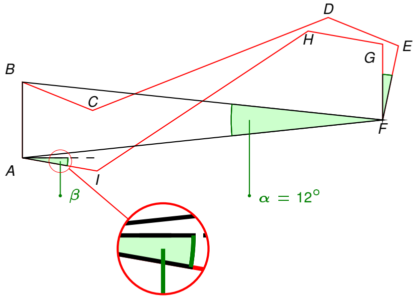
Example for scopes



Intersections

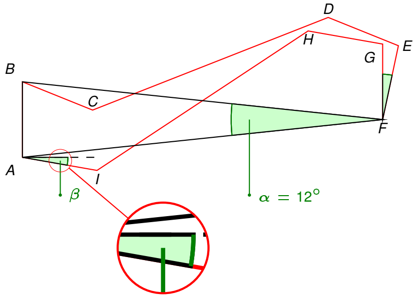


Intersections



Nice detail:
End point of helping line is in the middle of the angle.

Intersections



Nice detail:
End point of helping line is in the middle of the angle.

The good thing:
Ti*k*Z can automatically compute these intersection points.
Paths can be arbitrary, not only line segments!

Intersections (cont.)

How to compute intersections

1. Include the package `intersections`
2. Name paths using the option `[name path=pname]`
   Hint: invisible paths can be drawn using `\path`
3. Compute intersections as new path:
   `\path [name intersections={of=pname1 and pname2}];`

➡ new intersection points are now available at `(intersection-1)`, `(intersection-2)` etc.

Intersections (cont.)

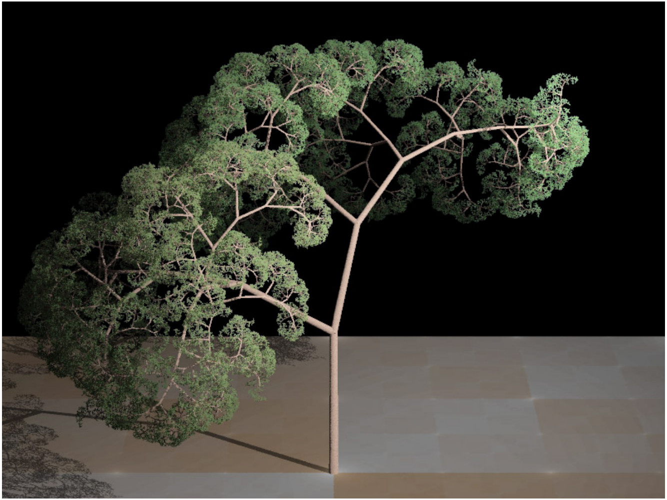
How to compute intersections

1. Include the package `intersections`
2. Name paths using the option `[name path=pname]`
   Hint: invisible paths can be drawn using `\path`
3. Compute intersections as new path:
   `\path [name intersections={of=pname1 and pname2}];`

➡ new intersection points are now available at `(intersection-1)`, `(intersection-2)` etc.

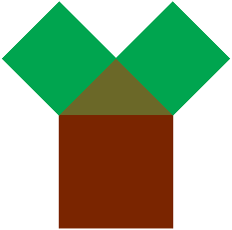
```
\path[name path=helpPath] (helpPoint) -- (labelPoint);
\path[name path=ai] (B) -- (F);
\path [name intersections={of=helpPath and ai}];
\coordinate (inters) at ($ (intersection-1)!0.5!(helpPoint) $
```

Trees

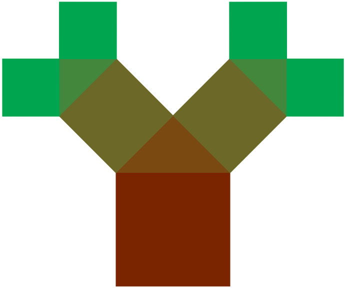


Trees

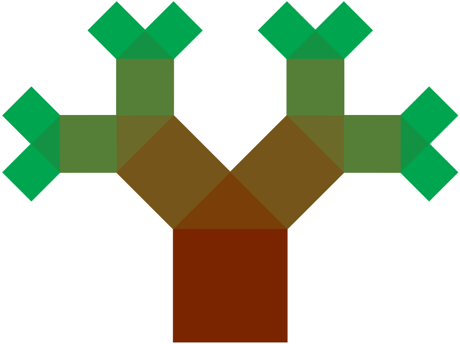
Trees



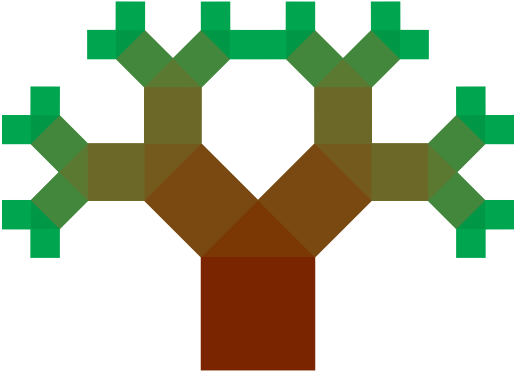
Trees



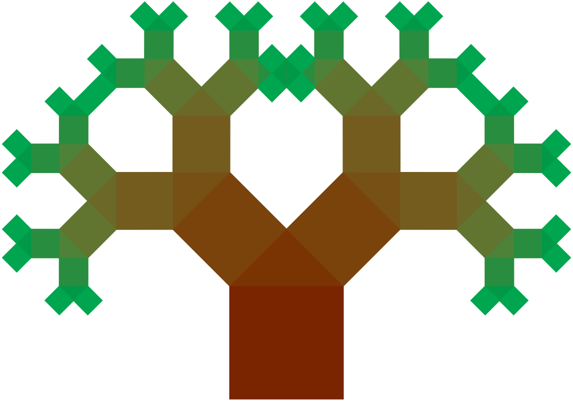
Trees



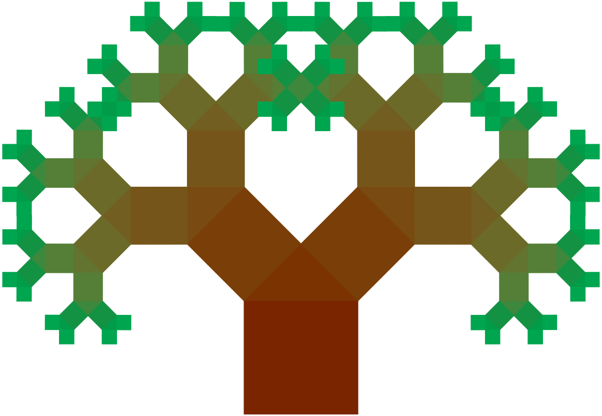
Trees



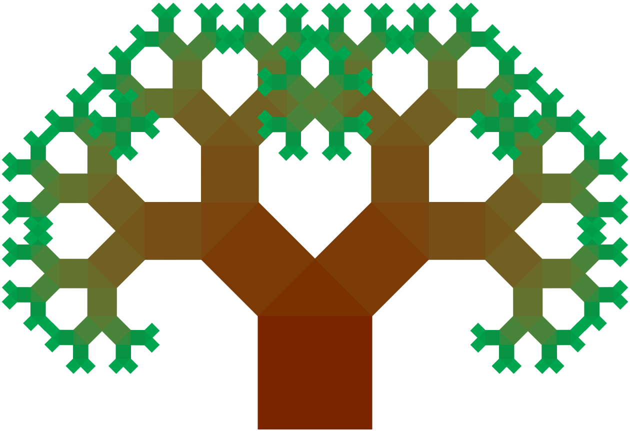
Trees



Trees



Trees

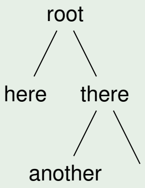


Trees – More like this

Defining Trees

- Use `child{}` in a node definition to create a child
- Use node and childs iterativeley to create a tree

Example




```
\node {\footnotesize root}
  child {
    node {\footnotesize here}}
  child {node {\footnotesize there}
    child {
      node {\footnotesize another}}
    child {
      node {}
};
```

Split nodes

The Rectangle Split Shape

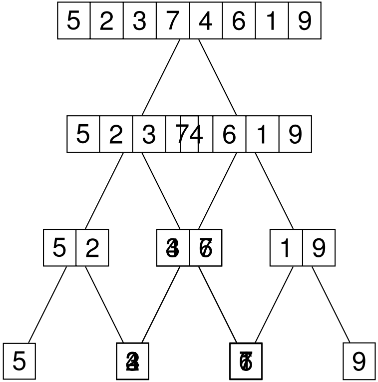
- Allows to put more than one text into nodes
- Need to include tikzlibrary `shapes.multipart`
- The style is `rectangle split`
- This gives a vertically split. For horizontally split add `rectangle split horizontal`

Example

```
\node[rectangle split,rectangle split parts=3,draw]
{1\nodepart{two}2\nodepart{three}3};
```

1
2
3

Recursive Sorting



Recursive Sorting

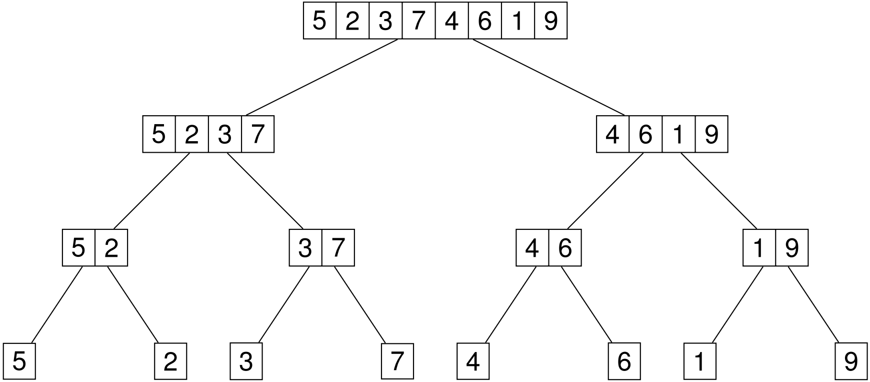
- Unfortunately children overlap
- Can be solved via `sibling distance`
- A style for each level of the tree

Example

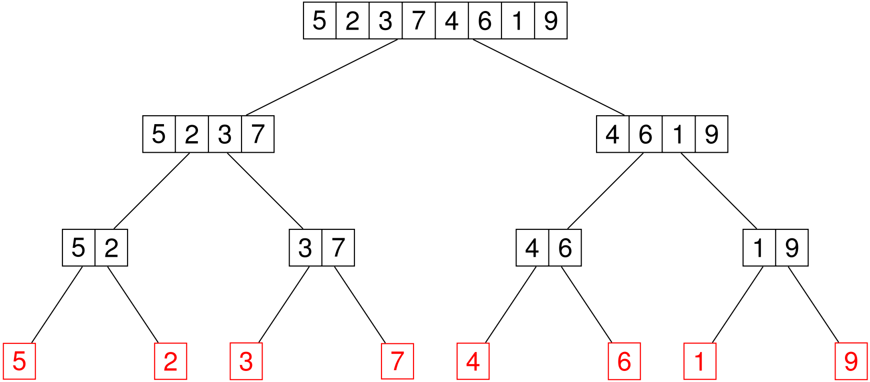
```
\begin{tikzpicture}[
level 1/.style={sibling distance=60mm},
level 2/.style={sibling distance=30mm},
level 3/.style={sibling distance=20mm}]
\end{tikzpicture}
```

➡ All siblings on level 1 will have a distance of $60mm$

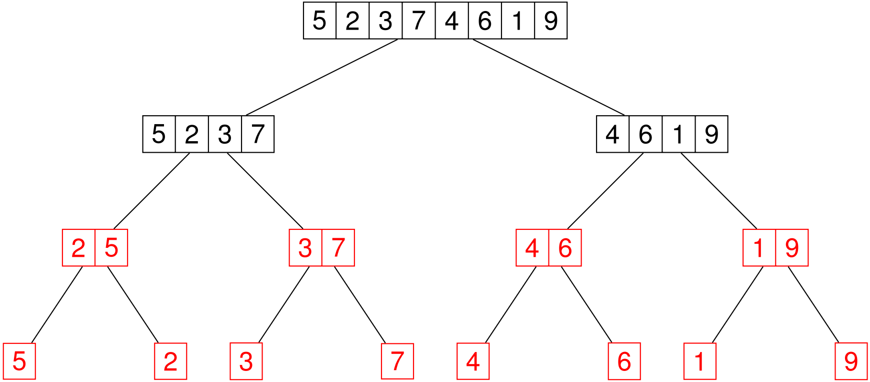
Recursive Sorting



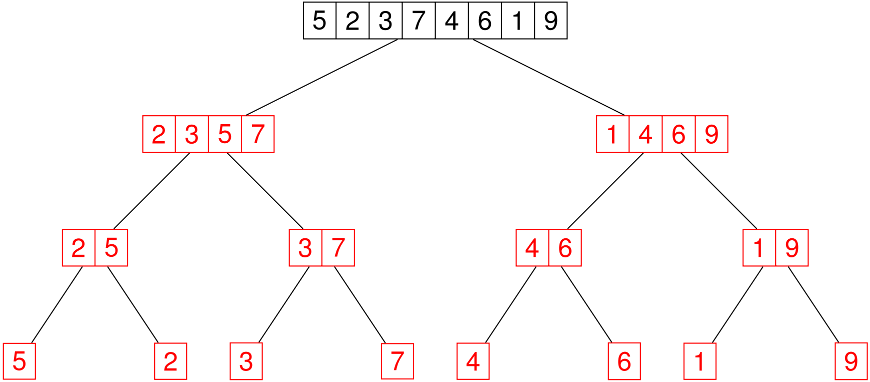
Recursive Sorting



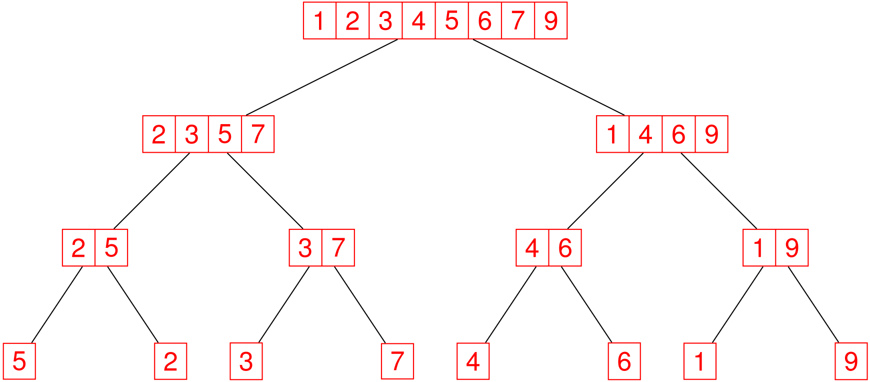
Recursive Sorting



Recursive Sorting



Recursive Sorting



Using styles to clean up code

Styles

- Using a lot of parameters creates ugly code
- User defined styles help keeping code clean
- Style needs to be changed once only

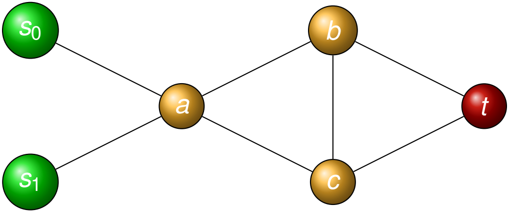
Usage:

```
\begin{tikzpicture}
[stylename/.style={some commands},
 otherstyle/.style={some commands}]
\end{tikzpicture}
```

Styles example

Example

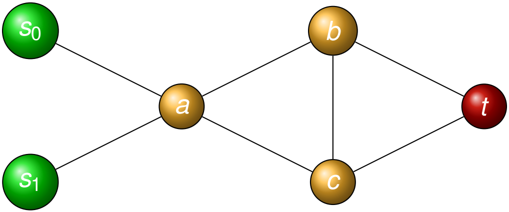
```
[default/.style={draw,fill,circle,shading=ball,
  ball color=Dandelion,text=white},
source/.style={draw,fill,circle,shading=ball,
  ball color=ForestGreen,text=white},
sink/.style={draw,fill,circle,shading=ball,
  ball color=BrickRed,text=white}]
```



Styles example

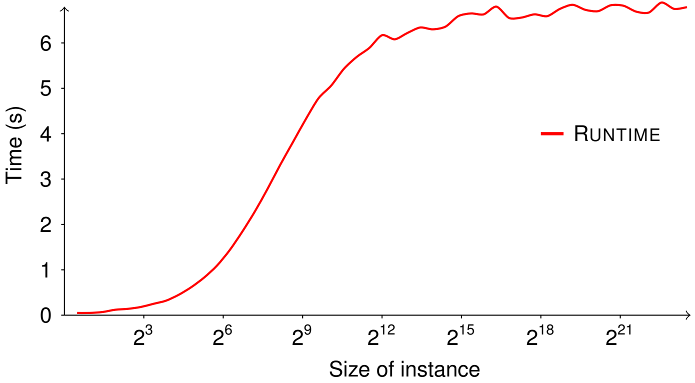
Example

```
\node[source] (1) at (0,1) {$s_0$};
\node[source] (2) at (0,-1) {$s_1$};
\node[default] (3) at (2,0) {$a$};
\node[default] (4) at (4,1) {$b$};
\node[default] (5) at (4,-1) {$c$};
\node[sink] (6) at (6,0) {$t$};
```



Plotting in Ti*k*Z

Something like this is possible in Ti*k*Z:



But: quite lengthy code, as axes and legend have to be drawn manually

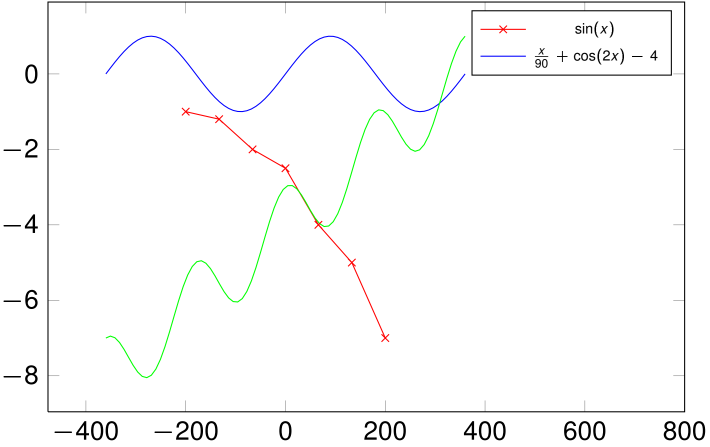
Plotting using PGFPLOTS

The PGFPLOTS package

- Package specialized for drawing plots
- Based upon PGF/Ti*k*Z
- Available at http://sourceforge.net/projects/pgfplots
- The manual is as good as the one of Ti*k*Z

On the following slides, there will be just three examples. For more, have a look in the manual.

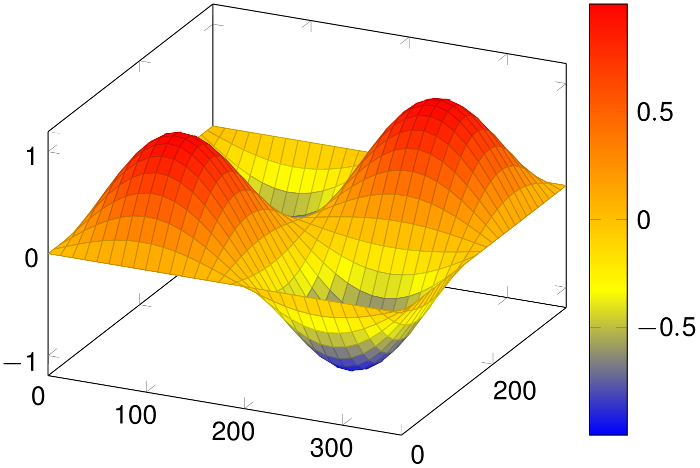
A starting example



A starting example

```
\begin{tikzpicture}
  \begin{axis}[domain=-360:360,samples=80,
                 width=10cm,height=7cm,xmax=800]
    \addplot[color=red,mark=x] coordinates {
      (-200,-1)
      (-133,-1.2)
      (-66,-2)
      (0,-2.5)
      (66,-4)
      (133,-5)
      (200,-7)
    };
    \addplot[color=blue] {sin(x)};
    \addplot[color=green] {-4+x/90+cos(x*2)};
  \end{axis}
\end{tikzpicture}
```

Plots in 3D

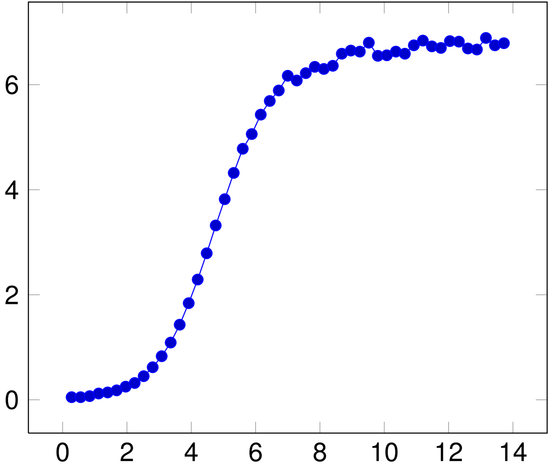


Plots in 3D

```
\begin{tikzpicture}
  \begin{axis}
    \addplot3[surf,domain=0:360,samples=50]
      {sin(x)*sin(y)};
  \end{axis}
\end{tikzpicture}
```

Take care, high sample values are not possible due to memory limitations!

Plotting from files



Plotting from files

```
\begin{tikzpicture}
  \begin{axis}
    \addplot file {charts/data.table};
  \end{axis}
\end{tikzpicture}
```

File features

- Reads gnuplot style files with datapoints specified as `x y i` with x and y beeing floating point values
- Also specific rows of a table can be read
- For details, see the manual

Some styles to define a graph and algorithm visualization

Requirements

- Need styles for nodes and edges
- Styles should change with the algorithm state

➡ Good idea to nest styles!

Some styles to define a graph and algorithm visualization

Requirements

- Need styles for nodes and edges
- Styles should change with the algorithm state

➡ Good idea to nest styles!

```
\tikzstyle{vertex}=[draw,circle,fill=Gray,minimum size=20pt]
\tikzstyle{selected vertex} = [vertex, fill=Maroon]
\tikzstyle{edge} = [draw,thick,-]
\tikzstyle{weight} = [font=\small]
\tikzstyle{selected edge} = [draw,line width=5pt,-,Green]
\tikzstyle{ignored edge} = [draw,line width=5pt,-,Salmon]
```

➡ Style „selected vertex“ based on „vertex“, but changes the fill color

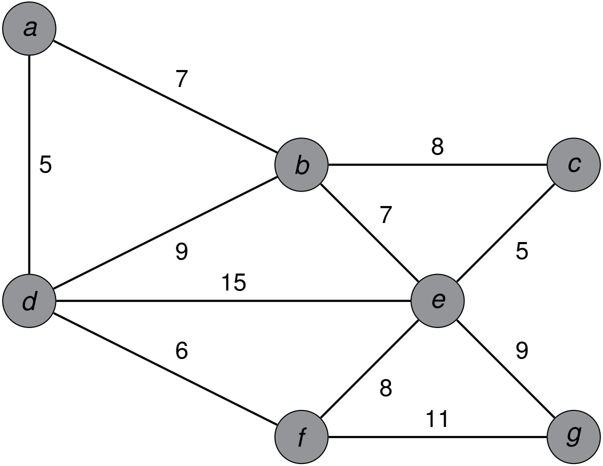
Defining a graph in four lines...

Using all our so far gained knowledge, we can say:

```
\foreach \pos/\name in {{(0,2)/a}, {(2,1)/b}, {(4,1)/c},
          {(0,0)/d}, {(3,0)/e}, {(2,-1)/f}, {(4,-1)/g}}
  \node[vertex] (\name) at \pos {$\name$};
\foreach \source/\dest /\weight in {b/a/7,c/b/8,d/a/5,d/b/9,
                e/b/7,e/c/5,e/d/15,f/d/6,f/e/8,g/e/9,g/f/11}
  \path[edge] (\source) -- node[weight] {$\weight$} (\dest);
```

Using: styles, node names and foreach with tuples

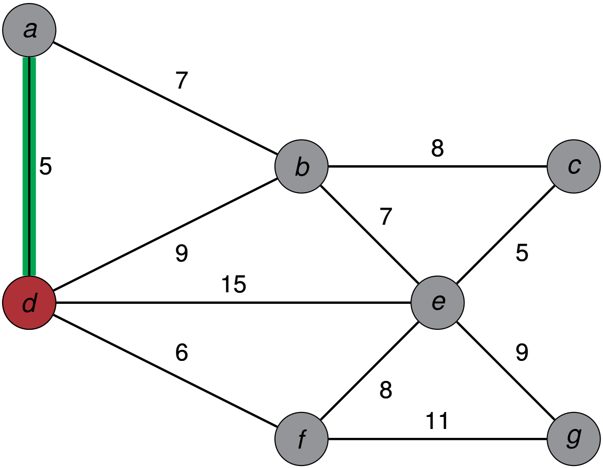
Defining a graph in four lines...



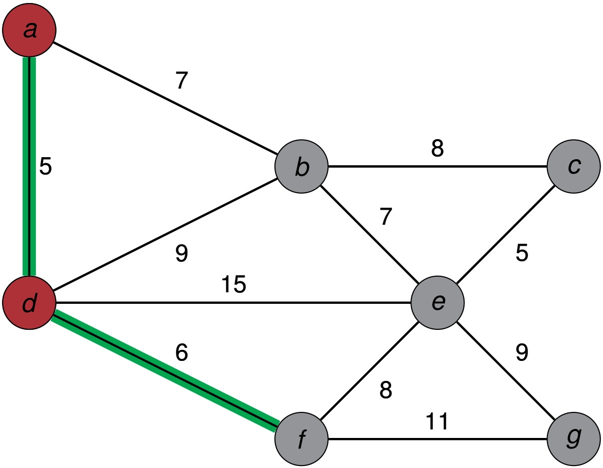
... and an algorithm in another six

```
\foreach \vertex / \slide in {d/1,a/2,f/3,b/4,e/5,c/6,g/7}
  \path<\slide-> node[selected vertex] at (\vertex) {$\vertex
\foreach \source / \dest in {d/a,d/f,a/b,b/e,e/c,e/g}
  \path<+->[selected edge] (\source) -- (\dest);
\foreach \source / \dest / \slide in
        {d/b/4,d/e/5,e/f/5,b/c/6,f/g/7}
  \path<\slide->[ignored edge] (\source) -- (\dest);
```

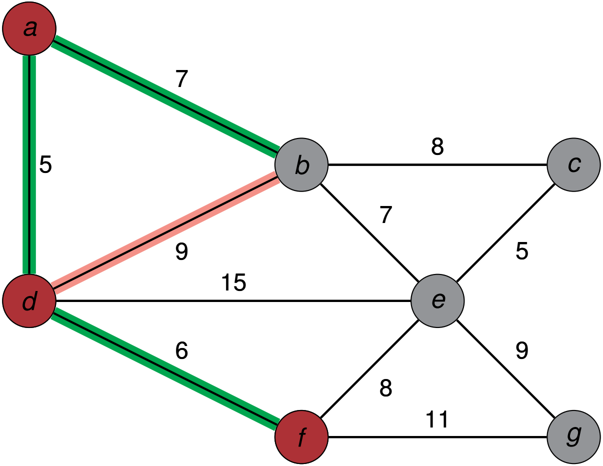
... and an algorithm in another six



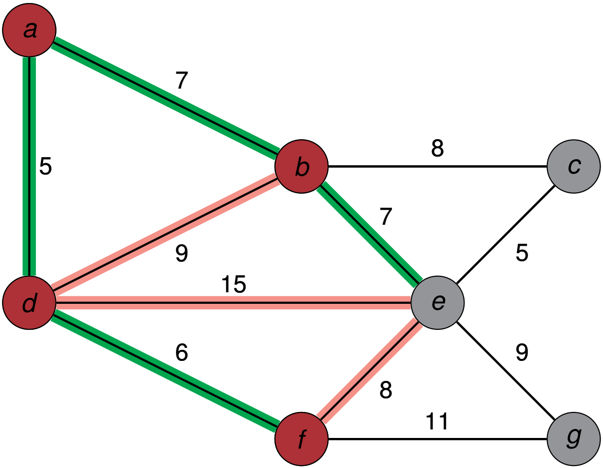
... and an algorithm in another six



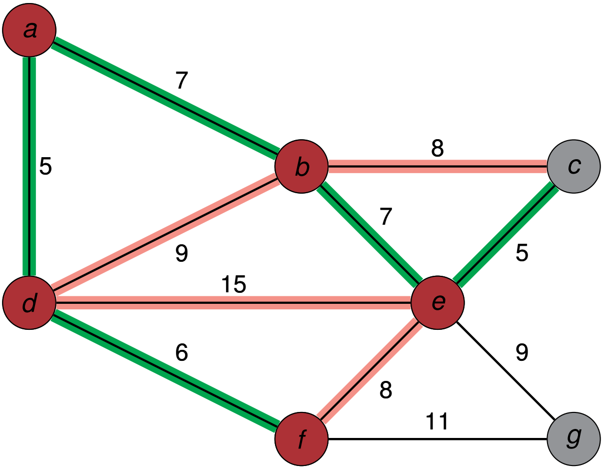
... and an algorithm in another six



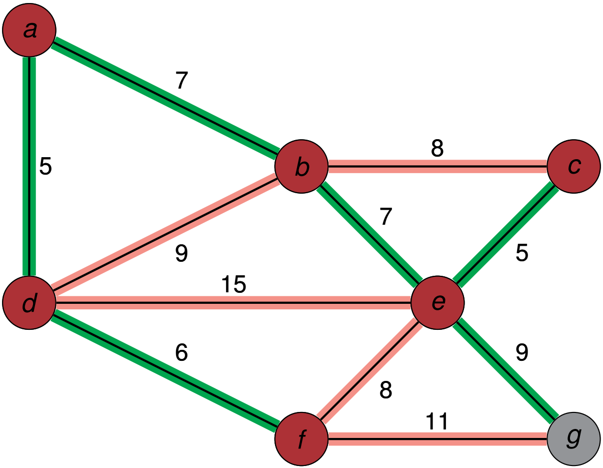
... and an algorithm in another six



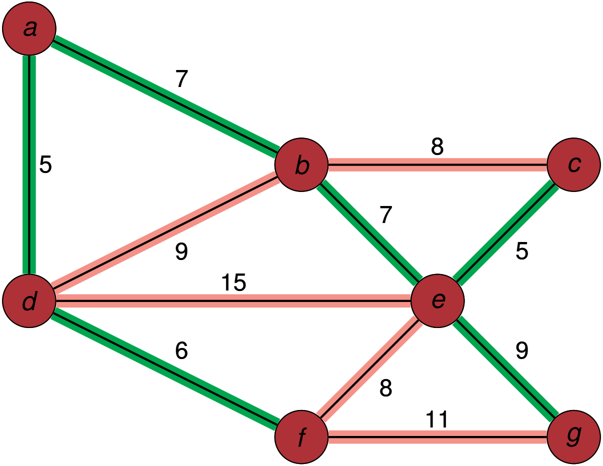
... and an algorithm in another six



... and an algorithm in another six

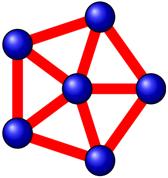


... and an algorithm in another six



Outlook

- Ti*k*Z really can do a lot of stuff
- Much more can be done using some of the many packages
- For example object oriented programming
- Worth reading a bit in the manual



Thank you!