August 21, 2025

The Mindmap Module

Li Yanrui (lyr.m2@live.cn)

https://github.com/liyanrui/mindmap

Introduction

The mindmap is a ConTEXt module written in MetaPost for drawing mind maps. Unlike most mind-mappint software, the mindmap module places all information on paths, its nodes carry no content at all. In other words, the module sees no container-like nodes filled with text or images. A mind map is simply a set of connected paths, and the information appears as annotations along them.

The simplest ConTEXt source file for using the mindmap module is as follows.

```
\usemodule[mindmap]
\startMPpage

% some MetaPost code for drawing mind map.

\stopMPpage
```

Creat a souce file named foo.tex that its content is

```
\usemodule[mindmap]
\startMPpage
mind.enter("$\delta_{ij}$ is", 15);
  mind("1, if $i=j$.", 10);
  mind("0, otherwise.", -20);
mind.exit;
\stopMPpage
```

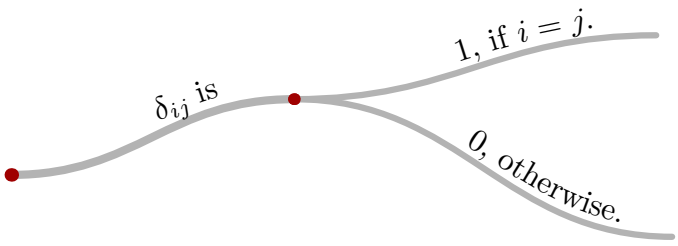
Use the `context` command to compile it into foo.pdf in the same directory.

```
$ context foo.tex
```

or

```
$ context foo
```

Then you can get the following result as shown in Example **1**.



Example 1 First mind map

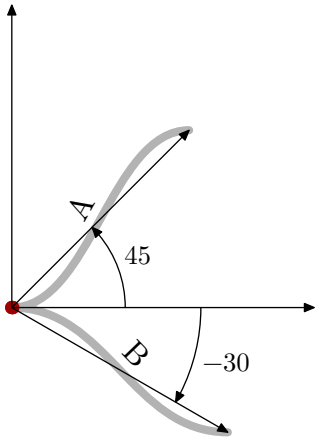
1 Branches

Every thought of yours can be expressed as a single branch in a mind map—just keep it as concise as possible, for instance:

```
mind("$\delta_{ij}$", 15);
```

In the code above, the `15` is the angle that indicates the direction of the branch in map. Every branch need an angle degree like this. The example below can help you understanding these.

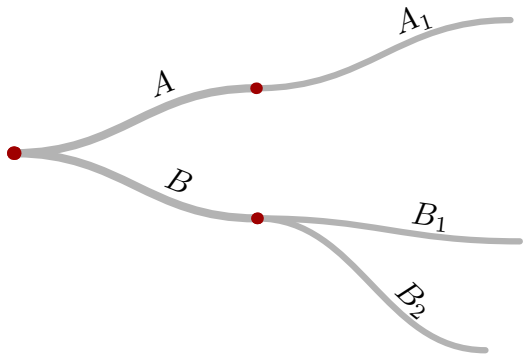
```
pair base, a, b;
base := MM.currentbase;
mind("A", 45); a := MM.currentend;
mind("B", -30); b := MM.currentend;

path pa, pb, ox, oy;
pa := base -- a; pb := base -- b;
ox := base -- (base + (4cm, 0));
oy := base -- (base + (0, 4cm));

path angle_a, angle_b;
anglelength := 1.5cm;
angle_a := anglebetween(ox, pa, "\tfx $45$");
anglelength := 2.5cm;
angle_b := anglebetween(ox, pb, "\tfx $-30$");

for it = pa, pb, ox, oy, angle_a, angle_b:
  drawarrow it;
endfor;
```



Example 2 Angles in mind map

If a branch has some deeper ones, you need to `enter` it and create child branches for it. When you want to go back to the parent branch and start new thought in the same level, you must `exit` from current child branches; see the following example.

```
mind.enter("$A$", 15);
  mind("$A_{1}$", 15);
mind.exit;
mind.enter("$B$", -15);
  mind("$B_{1}$", -5);
  mind("$B_{2}$", -30);
mind.exit;
```



Example 3 Entering and exiting branch

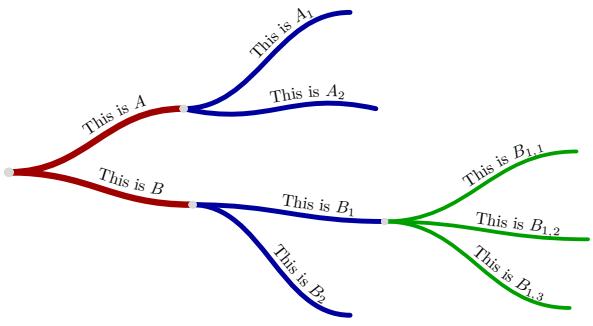
2 Style

The thickness of each branch decreases as the branch level increases. The top-level branch thickness defaults to `4pt`, but this can be changed with the `mind.thickness` macro. For the n-th level, the branch thickness equals the top-level thickness divided by 1.3^n.

By default, all branches are colored darkgray, but the macro `mind.colors` can be used to assign a specific color to each level's branches. The colors of branch knots can be controlled with the `mind.knotcolor` macro.

The example below sets the thickness of first level branch to `6pt`, and assigns colors to the branches and knots of levels 1 to 3.

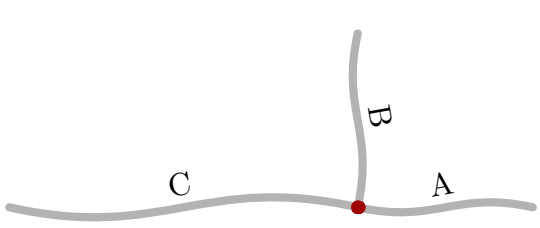
```
mind.thickness(6pt);
mind.colors(darkred,
            darkblue,
            darkgreen);
mind.knotcolor(lightgray);

mind.enter("This is $A$", 20);
  mind("This is $A_1$", 30);
  mind("This is $A_2$", 0);
mind.exit;

mind.enter("This is $B$", -10);
  mind.enter("This is $B_{1}$", -5);
    mind("This is $B_{1,1}$", 20);
    mind("This is $B_{1,2}$", -5);
    mind("This is $B_{1,3}$", -25);
  mind.exit;
  mind("This is $B_2$", -35);
mind.exit;
```



Example 4 Branch style setting

If you fell the lengths of branches too short, you can strech them by a given factor using the `mind.stretch` macro. The example below stretches the branches to twice their default length.

```
mind("A", 0);
mind("B", 90);

mind.stretch(2);
mind("C", -180);
```

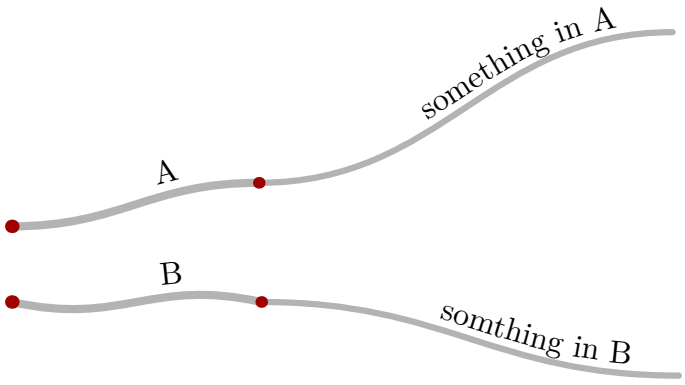


Example 5 Stretching branches

3 New Root

Mind maps drawn by the mindmap module are not strictly tree-structured. The default root is at `(0, 0)`, but you can use the `mind.newroot` macro to create the starting point or root of new mind map. For instance,

```
mind.enter("A", 10);
  mind("something in A", 20);
mind.exit;

mind.newroot(mind_b, (0, -1cm));
mind.enter("B", 0);
  mind("somthing in B", -10);
mind.exit;
```



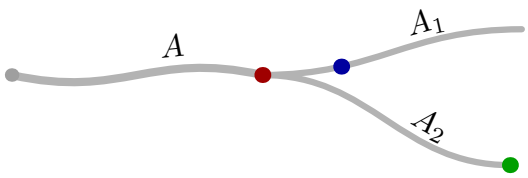
Example 6 New root

The `mind_b` in the code above is a variable of MetaPost's `pair` type, that stores the location of the new root.

Once you create a new branch, you can catch its base and handle point with the marcros `mind.base`, `mind.handle` and `mind.tailend`.

```
pair p[];
mind.knotcolor(darkgray);

mind.enter("$A$", 0);
  mind("$A_1$", 10); mind.handle(p1);
  mind("$A_2$", -20); mind.tailend(p2);
  mind.base(p3);
mind.exit;
```

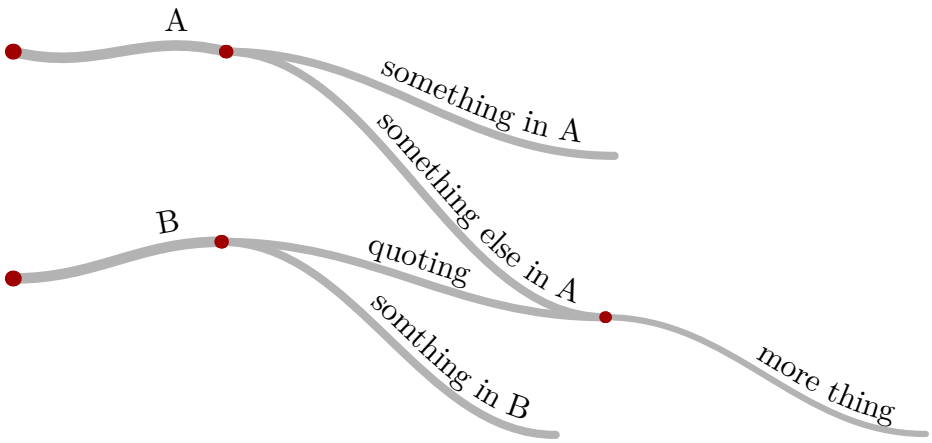
```
pickup pencircle scaled 6pt;
draw p1 withcolor darkblue;
draw p2 withcolor darkgreen;
draw p3 withcolor darkred;
```



Example 7 Anchors

Based on these anchor points, we can quote a branch in other tree with a new root. The following example shows a scenario where two trees share a branch.

```
pair demo;
mind.enter("A", 0);
  mind("something in A", -15);
  mind.enter("something else in A", -35);
    mind("more thing", -20); mind.base(demo);
  mind.exit;
mind.exit;
mind.newroot(B, (0, -3cm));
mind.enter("B", 10);
  mind.quote("quoting", demo);
  mind("somthing in B", -30);
mind.exit
```

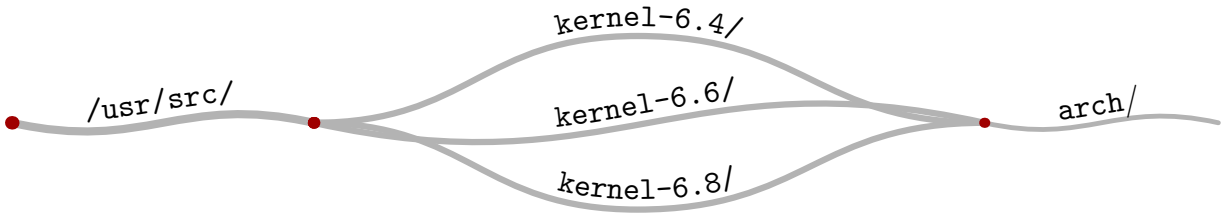


Example 8 Quoting branch

4 Convergence

Most mind-mapping software assumes that thinking is always divergent. The mindmap module disagrees, so it offers a convergent mind feature. You can converge a set branches with the macros `mind.converge` and `mind.converge.enter`.

```
pair p;
MM.BEGIN := 0; MM.END := 1;
mind.enter("\type{/usr/src/}", 0);
  mind.stretch(2);
  mind("\type{kernel-6.6/}", 0);
  mind.tailend(p);
  mind.converge("\type{kernel-6.4/}", 25, p);
  mind.converge.enter("\type{kernel-6.8/}", -25, p);
    mind.stretch(1);
    mind("\type{arch/}", 0);
  mind.ext;
mind.exit;
```



Example 9 Branch convergence

In the example above, MM.BEGIN and MM.END both are the module parameters. They delimit the segment of a branch where annotation text may appear; the text is restricted to the interval [MM.BEGIN, MM.END] along that branch. The segment [0, 1] represents the entire branch.

Afterwords

The mindmap module is a practice in learning the MetaPost language. Its inspiration and foundation come from the macro lmt_followtext, implemented by Hans Hagen in LuaMetaFun—the next-generation MetaPost still under development—which places text along an arbitrary curved path; see Chapter 5 of the LuaMetaFun manual. Within the ConTEXt LMTX environment, the manual can be founded with the following command:

```
$ mtxrun --search luametafun.pdf
```