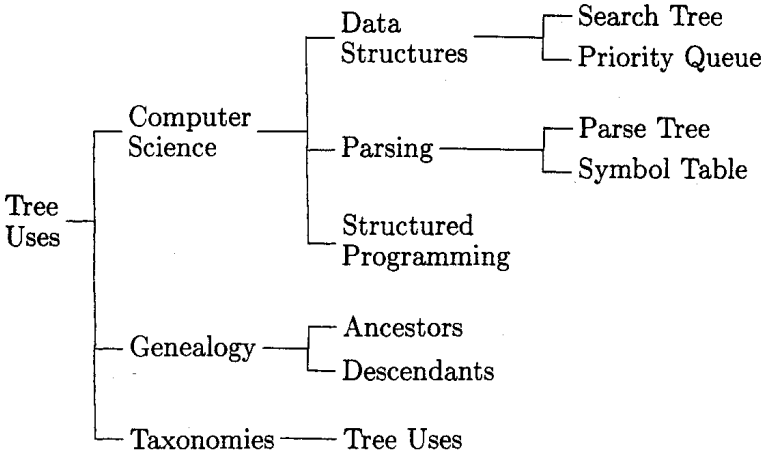
TREES IN TEX

David Eppstein

Introduction

There are many possible uses for trees in typeset text. The following taxonomy illustrates some of them.



Unfortunately TEX provides no easy way to typeset such trees. One possible method is given in exercise 22.14 of the TEXbook: using TEX's alignment primitives by hand. This method becomes very clumsy as the trees grow, however. A more general technique is to write a set of tree construction macros; that is the approach taken in this paper. The taxonomy above was typeset with the following input:

```
\tree
  Tree
  Uses
  \subtree
    Computer
    Science
    \subtree
      Data
      Structures
      \leaf{Search Tree}
      \leaf{Priority Queue}
    \endsubtree
    ...
  \endsubtree
  ...
\endtree
```

It turns out that TEX's alignment primitives are not very well suited to automatic generation of trees. The left edges of the trees at each level can easily be made to line up, but it is difficult to center lines of text for the root of a tree in vertical relation to its subtrees. Instead, the macros described here construct trees from boxes and glue, doing the alignment themselves. This is not quite as simple as it sounds—it would be incorrect to set subtrees independently of each other, because then the edges would not line up. For instance, in the taxonomy above, the text "Search Tree" should line up with "Parse Tree". A one-pass algorithm would set the former somewhat to the right of the latter.

To solve this problem, the macros described here set a tree using three passes. First, a data structure is built up from the tree definition. Second, that data structure is used to calculate the width of each level of the tree, so that the subtrees can be aligned with each other. Finally, the data structure and the calculated list of widths are used to set the system of boxes, glue, and rules that make up the tree.

Pass 1: Internal storage of the tree structure

There are several possible ways to store the structure defined by the tree macros. Since we want to remember already-set text (the words at the roots of each tree of subtree) we will use a nested structure of boxes. Each subtree is stored in an `\hbox`, so that pieces of it can be pulled off easily using `\lastbox` and `\unskip`. To distinguish it from another subtree, the text at the root of a subtree is stored in a `\vbox`. To make this clearer, let us return to our original taxonomy. We shall ignore for the moment the details inside the text `\vboxes`, and the glue between boxes. After the first pass, the tree as a whole would look like the following set of boxes:

```
\hbox{\vbox{Tree Uses}
  \hbox{\vbox{Computer Science}
    \hbox{\vbox{Data Structures}
      \hbox{\vbox{Search Tree}}
      \hbox{\vbox{Priority Queue}}}
    ...}
  ...}
```

Now we can begin defining the tree macros. We start defining a tree with the `\tree` macro; this merely sets up the assignment of the boxed tree structure into a box called `\treebox`. Starting a subtree is similar, but there is no assignment; also, if it is the first `\subtree` of its tree or subtree, we must stop making the `\vbox` containing the root text. A leaf is merely a subtree without any sub-subtrees.

```
\newbox\treebox
\def\tree{\global\setbox\treebox=\boxtree}
\def\subtree{\ettext \boxtree}
\def\leaf#1{\subtree#1\endsubtree}
```

Finishing a subtree merely involves first making sure the root text is complete, and second completing the box that was started in the expansion of `\subtree`. Finishing a whole tree involves both of those steps, but then after the box is completed the remaining two passes must be run.

```
\def\endsubtree{\ettext \egroup}
\def\endtree{\endsubtree \settreesizes \typesettree}
```

Now all that remains to be defined of the first pass is the construction of the `\vbox` containing the root text. The difficulty here is convincing TEX to make the `\vbox` only as wide as the widest line of text, rather than the width of the entire page. One solution is to put the text in an `\halign`, with `\crcr` implicit at the end of each line. The `\iftreetext` test is used to tell whether we are still inside the `\halign` and `\vbox`, so that `\ettext` can tell whether it should do anything. It is globally false, but within the `\vbox` it gets set to true.

```
\newif\iftreetext\treetextfalse          % Whether still aligning text
\def\boxtree{\hbox\bgroup                % Start outer box of tree or subtree
  \baselineskip 2.5ex                    % Narrow line spacing slightly
  \tabskip 0pt                           % No spurious glue in alignment
  \vbox\bgroup                           % Start inner text \vbox
  \treetexttrue                          % Remember for \ettext
  \let\par\crcr \obeylines               % New line breaks without explicit \cr
  \halign\bgroup##\hfil\cr}              % Start alignment with simple template
\def\ettext{\iftreetext                  % Are we still in inner text \vbox?
  \crcr\egroup \egroup \fi}              % Yes, end alignment and box
```

Pass 2: Calculation of widths at each level

Here we calculate a list of the dimensions of each level of the tree; that is, the widths of the widest \vbox at each level. To do this, we need to be able to maintain lists of things. Since these are dimensions rather than boxes of text it will be most convenient to use macros like the ones given on page 378 of the TEXbook. However, it turns out that we need to set our lists both locally to some grouping and also globally. Therefore, we will use a stripped down version of those list macros that can handle the \global flag. To implement this feature, we have to lose some others; the chief losses are that the contents of the lists will be macro-expanded by various of the list manipulation macros, and that we can't use redefinitions of \\ to perform some operation on the whole list.

To initialize a control sequence to the empty list, we do \let\csname\nil. Then to add an element to the start of the list we do \cons{tokens}\csname, and to remove that element we do \cdr\csname. Both \cons and \cdr can be prefixed with \global. The first element on the list can be expanded into the token stream by doing \car\csname. There is no error checking, so giving \car or \cdr the empty list will cause mysterious errors later on. Because of the macro expansion performed by \cons and \cdr, the token used to separate list elements expands to itself, and unlike the TEXbook macros cannot be redefined to do anything useful.

```
\def\cons#1#2{\edef#2{\xmark #1#2}}      % Add something to start of list.
\def\car#1{\expandafter\docar#1\docar}  % Take first element of list
\def\docar\xmark#1\xmark#2\docar{#1}    % ..by ignoring rest in expansion.
\def\cdr#1{\expandafter\docdr#1\docdr#1}% Similarly, drop first element.
\def\docdr\xmark#1\xmark#2\docdr#3{\def#3{\xmark #2}}
\def\xmark{\noexpand\xmark}              % List separator expands to self.
\def\nil{\xmark}                         % Empty list is just a separator.
```

We calculate the level widths by calling \setsizes on the tree; it will in turn call itself recursively for each of its subtrees. The tree being sized will be in \box0, which is used as scratch in this macro, and the list of widths already found for this level and below will be in \treesizes (initially \nil). When the macro exits, \treesizes will be updated with the widths found in the various levels of the given tree or subtree. A new \dimen, \treewidth, is used within the macro to remember the previous maximum width at the level of the tree's root.

```
\def\settreesizes{\setbox0=\copy\treebox \global\let\treesizes\nil \setsizes}
\newdimen\treewidth                      % Width of this part of the tree.
\def\setsizes{\setbox0=\hbox\bgroup      % Get a horiz list as a workspace.
  \unhbox0\unskip                        % Take tree, unpack it into horiz list.
  \inittreewidth                         % Get old width at this level.
  \sizesubtrees                          % Recurse through all subtrees.
  \sizelevel                             % Now set width from remaining \vbox.
  \egroup}                               % All done, finish our \hbox.
```

The first thing \setsizes does is to find out what the previous maximum at this level was, and store it in \treewidth. If \treesizes is \nil, we haven't seen anything this deep in the tree before, so the previous size is zero. Otherwise, it is \car\treesizes, and we also do \cdr\treesizes to prepare for later recursive calls to \setsizes.

```
\def\inittreewidth{\ifx\treesizes\nil   % If this is the first at this level
    \treewidth=0pt                     % ..then we have no previous max width.
  \else \treewidth=\car\treesizes      % Otherwise take old max level width
    \global\cdr\treesizes              % ..and advance level width storage
    \fi}                               % ..in preparation for next level.
```

At this point, we have a horizontal list (the \hbox in \setsizes) containing the \vbox for the text at the root of this subtree, followed by the \hboxes for all of its sub-subtrees. We loop pulling boxes from the end of the list with \lastbox until we find the text \vbox, calling \setsizes recursively for each \hbox we come across.

```
\def\sizesubtrees{\loop                 % For each box in horiz list (subtree)
  \setbox0=\lastbox \unskip             % ..pull it off list and flush glue.
  \ifhbox0 \setsizes                    % If hbox, it's a subtree - recurse
  \repeat}                              % ..and loop; end loop on tree text.
```

Now all that remains to do in this call to \setsizes is to update \treewidth if the text box, which can be found in \box0, is wider than the previous maximum. Then we add the (possibly updated) value of \treewidth as a text string back onto the head of \treesizes.

```
\def\sizelevel{\ifdim\treewidth<\wd0     % If greater than previous maximum
   \treewidth=\wd0 \fi                   % Then set max to new high
 \global\cons{\the\treewidth}\treesizes}% In either case, put back on list
```

Pass 3: Typesetting the tree

We are now ready to begin actual construction of the tree. This is done by calling \maketree, which like \setsizes calls itself recursively for all subtrees. It adds an \hbox containing the given subtree (which it finds in \treebox) to the current horizontal list; thus the outer call to \maketree sends the whole tree into TEX's output stream.

```
\newdimen\treeheight                    % Height of this part of the tree.
\newif\ifleaf                           % Tree has no subtrees (is a leaf).
\newif\ifbotsub                         % Bottom subtree of parent.
\newif\iftopsub                         % Top subtree of parent.
\def\typesettree{\medskip \maketree \medskip}   % Make whole tree with spacing.
\def\maketree{\hbox{\treewidth=\car\treesizes   % Get width at this level.
  \cdr\treesizes                        % Set up width list for recursion.
  \makesubtreebox\unskip                % Set \treebox to text, make subtrees.
  \ifleaf \makeleaf                     % No subtrees, add glue.
  \else \makeparent \fi}}               % Have subtrees, stick them at right.
```

After \maketree sets \treewidth from \treesizes, it calls \makesubtreebox. This opens up the horizontal list describing this level of the tree, and checks whether it has subtrees. If not, \ifleaf is set to true; otherwise it is set to false, and \box0 is set to contain a \vbox of them with their connecting rules, except for the horizontal rule leading from the tree text to the subtrees. In any case \treebox is set to the \vbox containing the tree text.

```
{\catcode`@=11                          % Be able to use \voidb@x.
\gdef\makesubtreebox{\unhbox\treebox    % Open up tree or subtree.
  \unskip\global\setbox\treebox\lastbox % Pick up very last box.
  \ifvbox\treebox                       % If we're already at the \vbox
    \global\leaftrue \let\next\relax    % ..then this is a leaf.
  \else \botsubtrue                     % Otherwise, we have subtrees.
    \setbox0\box\voidb@x                % Init stack of processed subs
    \botsubtrue \let\next\makesubtree   % ..and call \maketree on them.
  \fi \next}}                           % Finish up for whichever it was.
```

If this tree or subtree itself has subtrees, we need to put them and their connections in \box0 for \makesubtreebox. We come here with the bottom subtree in \treebox, the remaining list of subtrees in the current horizontal list, and the already processed subtrees stacked in \box0. The \ifbotsub test will be true for the first call, that is, the bottom subtree. Here we process the subtree in \treebox. If this is the top subtree, we return; otherwise we tail recurse to process the remaining subtrees. We use \box1 as another scratch variable; this is safe because the \hbox in \maketree puts us inside a group, and also because we are not changing the output list.

```
\def\makesubtree{\setbox1\maketree      % Call \maketree on this subtree.
  \unskip\global\setbox\treebox\lastbox % Pick up box before it.
  \treeheight=\ht1                      % Get height of subtree we made
  \advance\treeheight 2ex               % Add some room around the edges
  \ifhbox\treebox \topsubfalse          % If picked up box is a \vbox,
    \else \topsubtrue \fi               % ..this is the top, otherwise not.
  \addsubtreebox                        % Stack subtree with the rest.
  \iftopsub \global\leaffalse           % If top, remember not a leaf
    \let\next\relax \else               % ..(after recursion), set return.
    \botsubfalse \let\next\makesubtree  % Otherwise, we have more subtrees.
  \fi \next}                            % Do tail recursion or return.
```

Each subtree in the list is processed and stacked in \box0; this is done by \addsubtreebox, which calls \subtreebox to add connecting rules to the subtree in \box1, and appends to that the old contents of \box0. The vertical connecting rules in the tree are made with tall narrow \hrules rather than a more simple calls to \vrule, because they are made inside a \vbox.

```
\def\addsubtreebox{\setbox0=\vbox{\subtreebox\unvbox0}}
\def\subtreebox{\hbox\bgroup            % Start \hbox of tree and lines
  \vbox to \treeheight\bgroup           % Start \vbox for vertical rules.
    \ifbotsub \iftopsub \vfil           % If both bottom and top subtree
        \hrule width 0.4pt              % ..vertical rule is just a dot.
      \else \treehalfrule \fi \vfil     % Bottom gets half-height rule.
    \else \iftopsub \vfil \treehalfrule % Top gets half-height the other way.
      \else \hrule width 0.4pt height \treeheight \fi\fi % Middle, full height.
    \egroup                             % Finish vertical rule \vbox.
  \treectrbox{\hrule width 1em}\hskip 0.2em\treectrbox{\box1}\egroup}
```

The last line of the definition of \subtreebox calls \treectrbox twice: once for the horizontal connecting rule, and once for the subtree box itself. This macro centers its argument in a \vbox the height of this subtree and surrounding space. We also define here \treehalfrule, the macro called to make an \hrule half the height of the subtree (with half the height of the horizontal connection added to make the corners come out square).

```
\def\treectrbox#1{\vbox to \treeheight{\vfil #1\vfil}}
\def\treehalfrule{\dimen0=\treeheight   % Get total height.
  \divide\dimen0 2\advance\dimen0 0.2pt % Divide by two, add half horiz height.
  \hrule width 0.4pt height \dimen0}    % Make a vertical rule that high.
```

That completes \makesubtree. If this subtree has no sub-subtrees under it, \maketree will now run \makeleaf; this merely adds the tree text to the \hbox opened in \maketree. Otherwise we call \makeparent to attach the sub-subtrees and connecting rules to the text at the root of the subtree.

```
\def\makeleaf{\box\treebox}             % Add leaf box to horiz list.
\def\makeparent{\ifdim\ht\treebox>\ht0  % If text is higher than subtrees
    \treeheight=\ht\treebox             % ..use that height.
  \else \treeheight=\ht0 \fi            % Otherwise use height of subtrees.
  \advance\treewidth-\wd\treebox        % Take remainder of level width
  \advance\treewidth 1em                % ..after accounting for text and glue.
  \treectrbox{\box\treebox}\hskip 0.2em % Add text, space before connection.
  \treectrbox{\hrule width \treewidth}\treectrbox{\box0}} % Add \hrule, subs.
```