Tries

Construction, manipulation, and visualization of prefix trees

Anton Antonov
Mathematica for Prediction blog
Mathematica for Prediction project at GitHub
October 2013
November 2013
December 2013

Introduction

This document is a guide for using the functions for creation, manipulation, and visualization of prefix trees (tries) of the package TriesWithFrequencies.m provided by the project MathematicaForPrediction at GitHub; see [1].

The first section gives a brief discussion of what are prefix trees. See also the Wikipedia entry [2] for an alternative introduction and references. The second section provides basic examples of construction, search, and visualization with prefix trees. The third section gives a concrete example with a trie construction with the text of the U.S. constitution. The fourth section discusses data mining and machine learning applications. (To be written...) The last section provides details of the trie implementation in the package [1]. (To be written...)

Prefix trees are also known as “tries”. We are going to use “trie” and “prefix tree” as synonyms.

What are prefix trees (tries)?

In computer science a prefix tree (or a trie) is a data structure that allows quick retrieval of associations between a set of keys and a set of values. It is assumed that each key is a sequence of elements. (Most often the keys are considered to be strings i.e. sequences of characters.) The prefix tree nodes do not store the keys. Instead the position of a node in a prefix tree corresponds to a key.

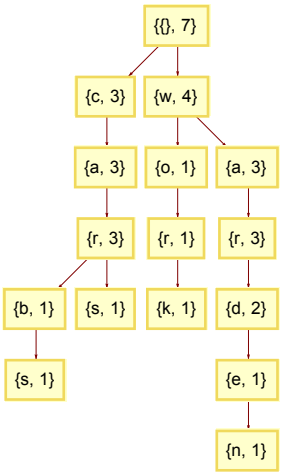
Consider the following list of words:

Out[207]//TableForm=

```
car
carbs
cars
war
ward
warden
work
```

Here is a prefix tree in which the keys are the list of words above and the values are the number of appearances of the corresponding prefix:

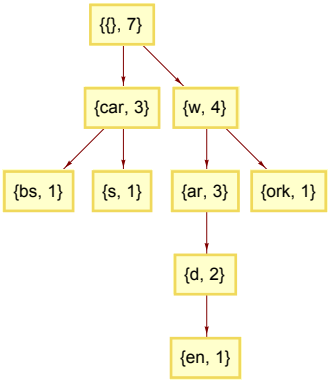
Out[210]=



For example, the prefix “car” appears three times in the list of words; that is why we have path {{}, 7} → {"c", 3} → {"a", 3} → {"r", 3}.

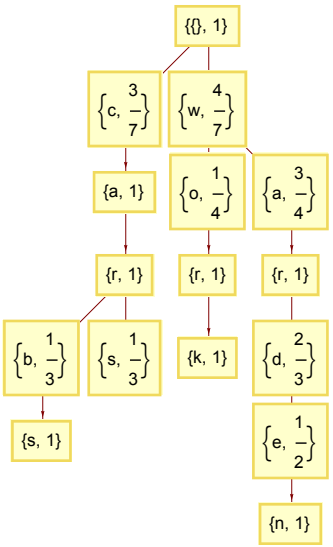
We can shrink the tree by merging the nodes that have only one child with their child. In this way we produce a tree in which each node would have an inseparable sub-sequences of elements. (In these examples these elements are characters.)

Out[212]=



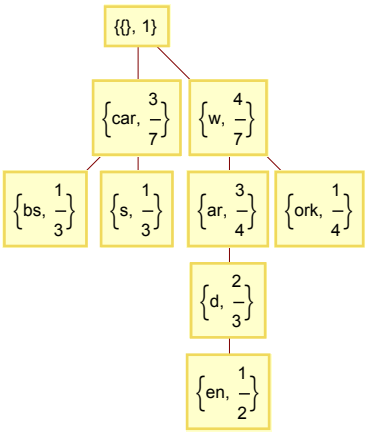
Another transformation of a trie is to replace the numbers of appearances with probabilities -- each value of a node is the probability to arrive to that node from the parent.

Out[215]=



Of course we can do both transformations.

Out[217]=



If we want to develop a trie package the most immediate trie operations to implement are: (i) trie creation from a list of words, (ii) finding the longest prefix of a word known to a trie, (iii) inserting a new word into a trie, (iv) finding the value corresponding to a key (i.e. word). The next section shows how to do with [1] these operations and the transformations described above. The package [1] implements trie operations and transformations for “words” that are strings or lists (of atom elements).

Tries creation and functionality

Let us see how we can construct tries using [1].

In[219]:=
```
Get["~/MathFiles/MathematicaForPrediction/TriesWithFrequencies.m"]
```

The functions of the package work with both strings and lists as keys. We are going to demonstrate functionalities using mostly strings, but also examples with lists would be shown.

First we are going to construct a simple trie.

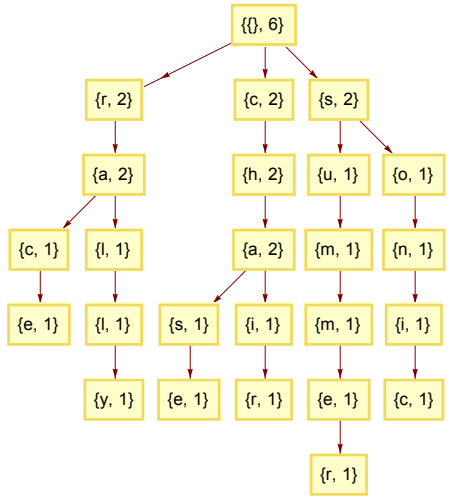
In[220]:=
```
mytrie = TrieCreate[{"race", "rally", "summer", "sonic", "chase", "chair"}]
```

Out[220]=
```
{{{}, 6}, {{r, 2}, {{a, 2}, {{c, 1}, {{e, 1}}}, {{l, 1}, {{l, 1}, {{y, 1}}}}}},
 {{c, 2}, {{h, 2}, {{a, 2}, {{s, 1}, {{e, 1}}}, {{i, 1}, {{r, 1}}}}}},
 {{s, 2}, {{u, 1}, {{m, 1}, {{m, 1}, {{e, 1}, {{r, 1}}}}}},
  {{o, 1}, {{n, 1}, {{i, 1}, {{c, 1}}}}}}}
```

Let us visualize the trie.

In[221]:=
```
TrieForm[mytrie]
```

Out[221]=



(Internally, the function `TrieForm` makes from its argument a list of rules that are suitable for visualization with `LayeredGraphPlot`.)

We can search for words in the trie with `TriePosition`.

In[222]:=
```
pos = TriePosition[mytrie, "sun"]
```

Out[222]=
```
{4, 2}
```

`TriePosition` returns the position of the subtree that has a root corresponding to the last character of the longest prefix of the word argument.

In[223]:=
```
mytrie[[Sequence @@ pos]]
```

Out[223]=
```
{{u, 1}, {{m, 1}, {{m, 1}, {{e, 1}, {{r, 1}}}}}}
```

As in illustration of the prefix tree structure manipulation let us retrieve the prefix from the prefix tree (instead of just using `StringTake["sun",Length[pos]]`).

In[224]:=
```
Map[mytrie[[Sequence @@ Join[#, {1, 1}]]] &,
 FoldList[Append, {First[#]}, Rest[#]] &@TriePosition[mytrie, "sun"]]
```

Out[224]=
```
{s, u}
```

Here is another example with a word known by the trie.

In[225]:=
```
pos = TriePosition[mytrie, "summer"]
```

Out[225]=
```
{4, 2, 2, 2, 2, 2}
```

Obviously, using the results `TriePosition` we can test which words are known by a trie. We can also use the function `TrieRetrieve` which returns the value corresponding to a key or an empty list if the key is not in the trie argument.

In[226]:=
```
TrieRetrieve[mytrie, "summer"]
```

Out[226]=
```
{r, 1}
```

In[227]:=
```
TrieRetrieve[mytrie, "serenity"]
```

Out[227]=
```
{}
```

We can insert a new word in the trie with the command `TrieInsert`:

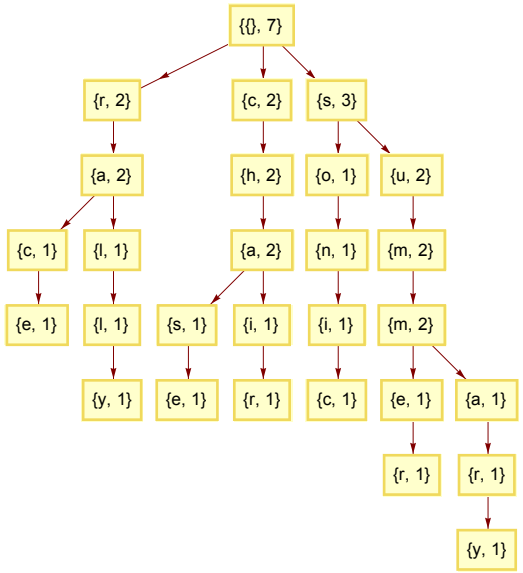
In[228]:=
```
mytrie = TrieInsert[mytrie, "summary"]
```

Out[228]=
```
{{{}, 7}, {{r, 2}, {{a, 2}, {{c, 1}, {{e, 1}}}, {{l, 1}, {{l, 1}, {{y, 1}}}}}},
 {{c, 2}, {{h, 2}, {{a, 2}, {{s, 1}, {{e, 1}}}, {{i, 1}, {{r, 1}}}}}},
 {{s, 3}, {{o, 1}, {{n, 1}, {{i, 1}, {{c, 1}}}}}, {{u, 2},
   {{m, 2}, {{m, 2}, {{e, 1}, {{r, 1}}}, {{a, 1}, {{r, 1}, {{y, 1}}}}}}}}}
```

Here is the graph form of the updated trie:

In[229]:=
```
TrieForm[mytrie]
```

Out[229]=



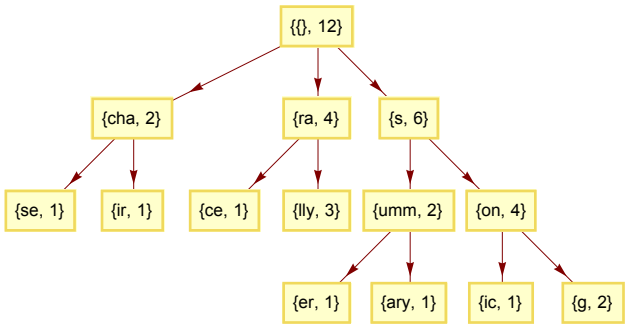
For the examples that follow let us insert several more words into the trie `mytrie`.

In[230]:=
```
mytrie = Fold[TrieInsert, mytrie, {"son", "song", "song", "rally", "rally"}];
```

Let us shrink the trie and visualize the result

In[231]:=
```
TrieForm[TrieShrink[mytrie]]
```

Out[231]=



The function `TrieShrink` groups internal nodes and leaves into “prefixes”.

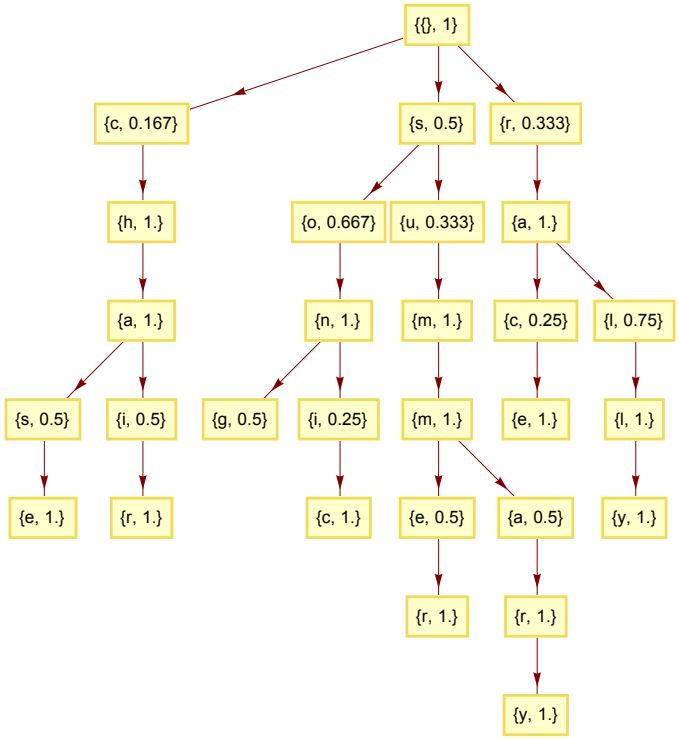
In order to replace the numbers of appearances with probabilities we use the function `TrieNodeProbabilities`.

In[232]:=
```
mytriep =
 TrieNodeProbabilities[mytrie, "ProbabilityModifier" → (Round[#, 0.001] &)]
```

Out[232]=
```
{{{}, 1}, {{c, 0.167},
  {{h, 1.}, {{a, 1.}, {{s, 0.5}, {{e, 1.}}}, {{i, 0.5}, {{r, 1.}}}}}},
 {{s, 0.5}, {{u, 0.333}, {{m, 1.},
    {{m, 1.}, {{e, 0.5}, {{r, 1.}}}, {{a, 0.5}, {{r, 1.}, {{y, 1.}}}}}}}},
  {{o, 0.667}, {{n, 1.}, {{i, 0.25}, {{c, 1.}}}, {{g, 0.5}}}}}, {{r, 0.333},
  {{a, 1.}, {{c, 0.25}, {{e, 1.}}}, {{l, 0.75}, {{l, 1.}, {{y, 1.}}}}}}}
```

In[233]:=
```
TrieForm[mytriep]
```

Out[233]=



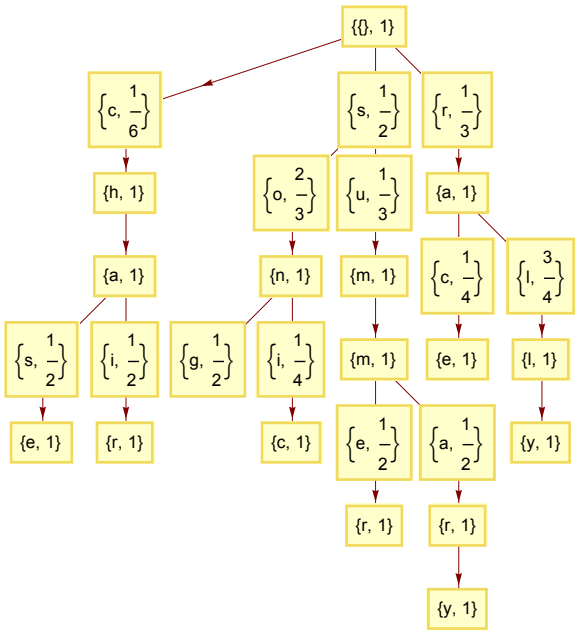
The function `TrieNodeProbabilities` takes the option “ProbabilityModifier” which can be used to change the appearance of the calculated probabilities. For example, we might prefer to see the probabilities as rational numbers

In[234]:=
```
TrieNodeProbabilities[mytrie, "ProbabilityModifier" → Rationalize]
```

Out[234]=
$$\left\{\{\{\}, 1\}, \left\{\left\{c, \frac{1}{6}\right\}, \left\{\{h, 1\}, \left\{\{a, 1\}, \left\{\left\{s, \frac{1}{2}\right\}, \{\{e, 1\}\}\right\}, \left\{\left\{i, \frac{1}{2}\right\}, \{\{r, 1\}\}\right\}\right\}\right\}\right\},\right.$$
$$\left\{\left\{s, \frac{1}{2}\right\}, \left\{\left\{u, \frac{1}{3}\right\},\right.\right.$$
$$\left\{\{m, 1\}, \left\{\{m, 1\}, \left\{\left\{e, \frac{1}{2}\right\}, \{\{r, 1\}\}\right\}, \left\{\left\{a, \frac{1}{2}\right\}, \{\{r, 1\}, \{\{y, 1\}\}\}\right\}\right\}\right\},$$
$$\left\{\left\{o, \frac{2}{3}\right\}, \left\{\{n, 1\}, \left\{\left\{i, \frac{1}{4}\right\}, \{\{c, 1\}\}\right\}, \left\{\left\{g, \frac{1}{2}\right\}\right\}\right\}\right\},$$
$$\left.\left.\left\{\left\{r, \frac{1}{3}\right\}, \left\{\{a, 1\}, \left\{\left\{c, \frac{1}{4}\right\}, \{\{e, 1\}\}\right\}, \left\{\left\{l, \frac{3}{4}\right\}, \{\{l, 1\}, \{\{y, 1\}\}\}\right\}\right\}\right\}\right\}$$

In[235]:=
```
TrieForm[%]
```

Out[235]=



For data mining purposes we want to find the probabilities to arrive at the leaves from the root of a trie. The function `TrieLeafProbabilities` does that.

In[236]:=
```
TrieLeafProbabilities[mytriep]
```

Out[236]=
```
{{e, 0.0835}, {r, 0.0835}, {r, 0.08325}, {y, 0.08325},
 {c, 0.083375}, {g, 0.16675}, {n, 0.083375}, {e, 0.08325}, {y, 0.24975}}
```

Let us verify that the sum of the probabilities is 1

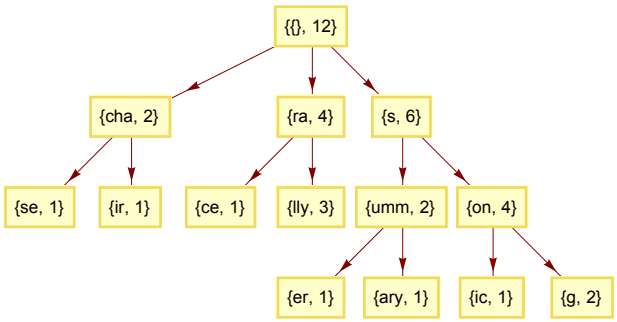
In[237]:=
```
Total[%[[All, 2]]]
```

Out[237]=
```
1.
```

Note that the internal node {“n”, _} is an “internal leaf”: the sum of the probabilities of its children is smaller than 1. This can be seen in also in the shrunk frequencies trie -- we have the word “son” inserted in the trie.

In[238]:=
```
TrieForm[TrieShrink[mytrie]]
```

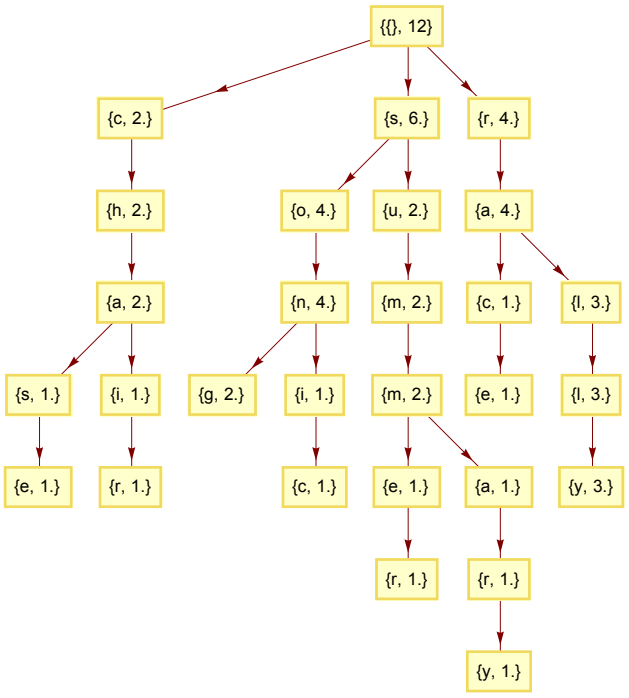
Out[238]=



The function `TrieNodeFrequencies` can be used to reverse the effect of `TrieNodeProbabilities`. Its first argument is a trie, the second one is a scaling parameter (which is 1 by default). Here is an example:

In[239]:=
```
TrieForm[TrieNodeFrequencies[TrieNodeProbabilities[mytrie], 12]]
```

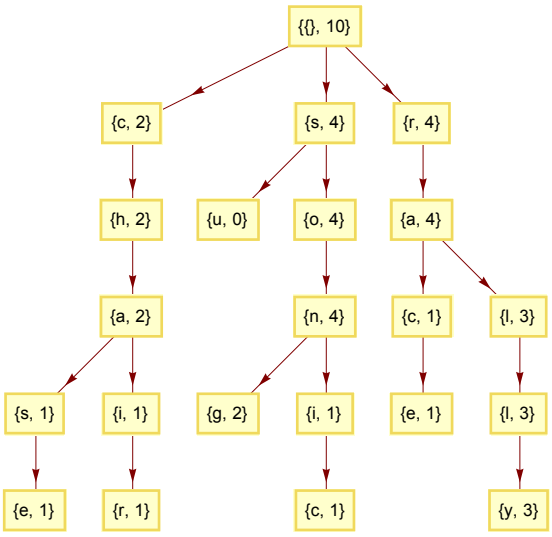
Out[239]=



The function `TrieRemove` can be used to remove sub-parts of a trie that correspond to a specified “word”. Here is an example using a trie with frequencies:

In[240]:=
```
TrieForm[TrieRemove[mytrie, "su"]]
```

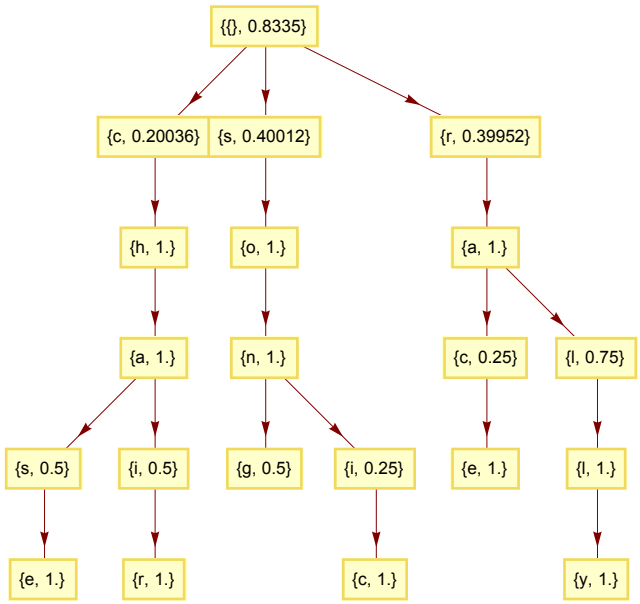
Out[240]=



Here is an example using a trie with probabilities:

In[242]:=
```
TrieForm[TrieRemove[mytriep, "su"]]
```

Out[242]=



Using lists

All the operations on tries for collections of strings also work for lists. This is demonstrated in this subsection with commands that mirror the ones used for tries of strings.

First let us convert a list of words into a list of integer lists.

In[243]:=
```
lwords = Flatten /@ Map[ToCharacterCode,
   Characters /@ {"race", "rally", "summer", "sonic", "chase", "chair"}, {2}]
```

Out[243]=
```
{{114, 97, 99, 101}, {114, 97, 108, 108, 121}, {115, 117, 109, 109, 101, 114},
 {115, 111, 110, 105, 99}, {99, 104, 97, 115, 101}, {99, 104, 97, 105, 114}}
```

Next is trie creation:

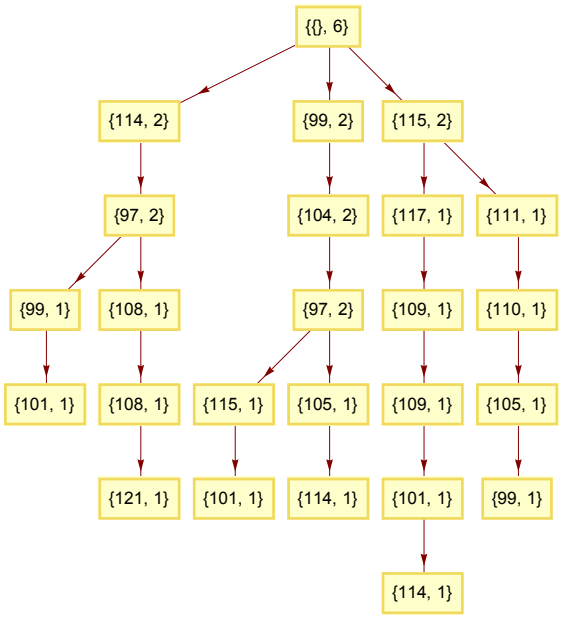
In[244]:=
```
ltrie = TrieCreate[lwords]
```

Out[244]=
```
{{{}, 6}, {{114, 2}, {{97, 2}, {{99, 1}, {{101, 1}}},
    {{108, 1}, {{108, 1}, {{121, 1}}}}}}, {{99, 2},
  {{104, 2}, {{97, 2}, {{115, 1}, {{101, 1}}}, {{105, 1}, {{114, 1}}}}}},
 {{115, 2}, {{117, 1}, {{109, 1}, {{109, 1}, {{101, 1}, {{114, 1}}}}}},
  {{111, 1}, {{110, 1}, {{105, 1}, {{99, 1}}}}}}}
```

Visualize the created trie:

In[245]:=
```
TrieForm[ltrie]
```

Out[245]=



For a list of integers find the position of the longest prefix known by the trie:

In[246]:=
```
TriePosition[ltrie, {114, 97, 200}]
```

Out[246]=
```
{2, 2}
```

Shrink the trie:

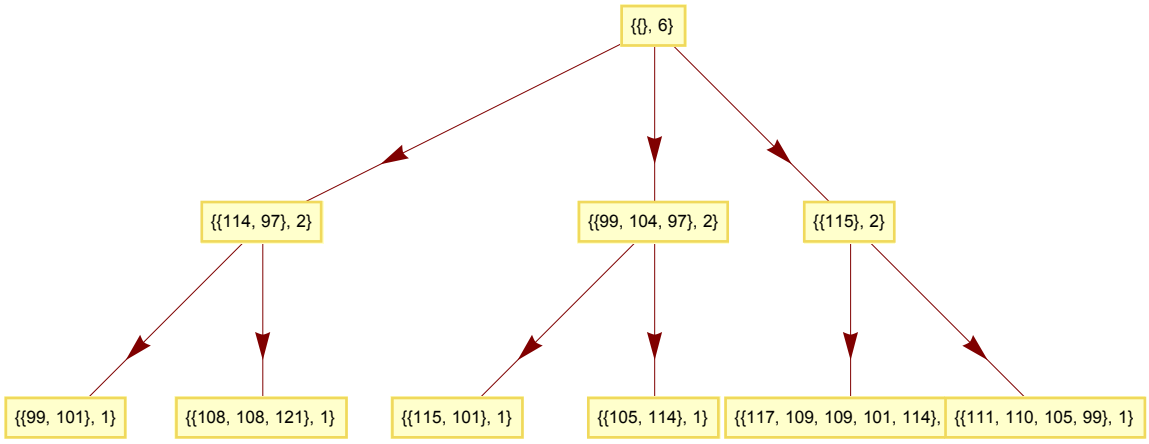
In[247]:=
```
sltrie = TrieShrink[ltrie]
```

Out[247]=
```
{{{}, 6}, {{{114, 97}, 2}, {{{99, 101}, 1}}, {{{108, 108, 121}, 1}}},
 {{{99, 104, 97}, 2}, {{{115, 101}, 1}}, {{{105, 114}, 1}}},
 {{{115}, 2}, {{{117, 109, 109, 101, 114}, 1}}, {{{111, 110, 105, 99}, 1}}}}
```

Visualize the shrunk trie:

In[248]:=
```
TrieForm[sltrie]
```

Out[248]=



Find the probabilities to reach the leaves:

In[249]:=
```
TrieLeafProbabilities[TrieNodeProbabilities[sltrie]] // Grid
```

Out[249]=

{99, 101}	0.166667
{108, 108, 121}	0.166667
{115, 101}	0.166667
{105, 114}	0.166667
{117, 109, 109, 101, 114}	0.166667
{111, 110, 105, 99}	0.166667

Using tries as associative arrays

We can create tries that can be used as associative arrays (dictionaries). The function `TrieInsert` has also the signature `TrieInsert[trie_,key_,value_]`.

Let us make a trie in which the values corresponding to the inserted words are their reversals.

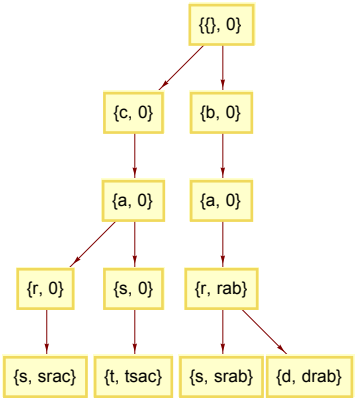
In[250]:=
```
aaTrie = Fold[TrieInsert[#1, #2, StringReverse[#2]] &,
  TrieCreate[{}], {"cars", "cast", "bar", "bars", "bard"}]
```

Out[250]=
```
{{{}, 0}, {{c, 0}, {{a, 0}, {{r, 0}, {{s, srac}}}, {{s, 0}, {{t, tsac}}}}}},
 {{b, 0}, {{a, 0}, {{r, rab}, {{s, srab}}, {{d, drab}}}}}}
```

Let us visualize the trie -- note that the prefixes of the inserted words have values 0. For example, since we did not insert the word “car” its value in the trie is 0.

In[251]:=
```
TrieForm[aaTrie]
```

Out[251]=



Here are examples of retrieval of associated values:

In[252]:=
```
TrieRetrieve[aaTrie, "car"]
```

Out[252]=
```
{r, 0}
```

In[253]:=
```
TrieRetrieve[aaTrie, "bars"]
```

Out[253]=
```
{s, srab}
```

In[254]:=
```
TrieRetrieve[aaTrie, "bard"]
```

Out[254]=
```
{d, drab}
```

In[255]:=
```
TrieRetrieve[aaTrie, "train"]
```

Out[255]=
```
{}
```

Example with the U.S. constitution

Load some text:

In[256]:=
```
text = ExampleData[{"Text", "USConstitution"}];
```

Split the text into words:

In[257]:=
```
words = ToLowerCase /@ StringSplit[text,
    {",", ".", ";", ":", "!", "?", " ", "-", "\n", "(", ")"}];
words //
 Length
```

Out[258]=
```
9166
```

Drop the words that are too short:

In[259]:=
```
words = Select[words, StringLength[#] ≥ 1 &];
words // Length
```

Out[260]=
```
8115
```

The number of unique words is

In[261]:=
```
Length[Union[words]]
```

Out[261]=
```
1273
```

Create a trie:

In[262]:=
```
usctrie = TrieCreate[words];
```

Convert the trie nodes values into probabilities:

In[263]:=
```
usctriep = TrieNodeProbabilities[usctrie];
```

Search for the prefix “universe”:

In[264]:=
```
pos = TriePosition[usctrie, "universe"]
```

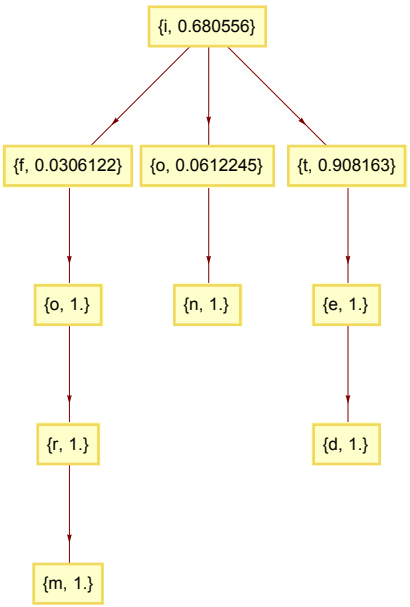
Out[264]=
```
{31, 4, 8}
```

We can see that only the prefix “uni” is in the trie.

Here is the subtree of the trie with the last letter of “uni” as a root:

In[265]:=
```
TrieForm[usctriep[[Sequence @@ pos]]]
```

Out[265]=



Find the probabilities and positions of words that starts with “uni” and finish with “m”, “n”, and “d”:

In[266]:=
```
probs = TrieLeafProbabilitiesWithPositions[usctriep[[Sequence @@ pos]]];
probs[[All, 2]] = probs[[All, 2]] / usctriep[[Sequence @@ pos, 1, 2]];
probs
```

Out[268]=
```
{{m, 0.0306122, {2, 2, 2, 2, 1}},
 {n, 0.0612245, {3, 2, 1}}, {d, 0.908163, {4, 2, 2, 1}}}
```

Here are the corresponding words:

In[269]:=
```
Function[{ppos}, "uni" <> Apply[StringJoin,
     Map[usctriep[[Sequence @@ pos]][[Sequence @@ Join[#, {1, 1}]]] &,
      FoldList[Append, {First[#]}, Rest[#]] &@Most[ppos]]]] /@ probs[[All, 3]]
```

Out[269]=
```
{uniform, union, united}
```

Let us find the most frequently appearing words in the text. First we find the probabilities to reach each of the leafs of the trie:

In[270]:=
```
probs = SortBy[TrieLeafProbabilitiesWithPositions[usctriep], -#[[2]] &];
probs // Length
```

Out[271]=
```
1273
```

Here are the top 40 most frequent words in the text:

In[272]:=
```
With[{ntop = 40}, t = Transpose[
    {Function[{ppos}, StringJoin @@ Map[usctriep[[Sequence @@ Join[#, {1, 1}]]] &,
         FoldList[Append, {First[#]}, Rest[#]] &@Most[ppos]]] /@
       probs[[1 ;; ntop, 3]], probs[[1 ;; ntop, 2]]}];
 Magnify[Grid[Flatten /@ Transpose[Partition[t, Length[t] / 2]],
   Alignment → Left, Spacings → {{Automatic, Automatic, 2}, Automatic}], 0.8]]
```

Out[272]=

the	0.0941466	such	0.00640789
of	0.0644486	may	0.00566852
shall	0.0377079	which	0.00542206
and	0.0338879	not	0.00542206
to	0.025878	all	0.00529883
be	0.0225508	no	0.0051756
or	0.019963	on	0.00505237
in	0.0184843	from	0.00492914
states	0.0166359	law	0.00480591
by	0.0150339	amendment	0.00468269
president	0.0147874	office	0.00455946
a	0.0121996	vice	0.00443623
united	0.0109673	this	0.00443623
for	0.0103512	house	0.00418977
state	0.00985829	person	0.00418977
congress	0.00973506	but	0.00406654
any	0.00973506	constitution	0.00382009
as	0.00825632	other	0.00382009
have	0.0077634	one	0.00369686
section	0.0069008	article	0.00369686

Using tries in data mining

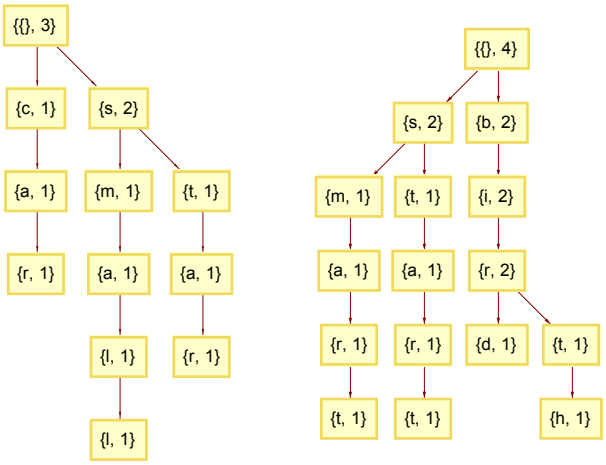
Implementation notes

Merging of tries

Consider these two tries:

In[273]:=
```
t1 = TrieCreate[{"car", "small", "star"}];
t2 = TrieCreate[{"smart", "start", "bird", "birth"}];
Grid[
 {{TrieForm[t1, ImageSize → 120], TrieForm[t2, ImageSize → 150]}}, Spacings → 3]
```

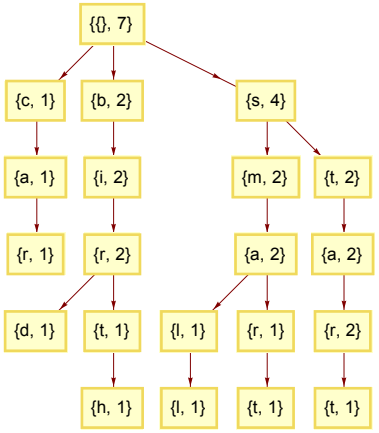
Out[275]=



We can merge these two tries with `TrieMerge` and get the trie:

In[276]:=
```
TrieForm[TrieMerge[t1, t2], ImageSize → 200]
```

Out[276]=



The initial version of `TrieCreate` used only `TrieInsert` with `Fold` over the argument list. The current, faster version of the function `TrieCreate` is implemented with `TrieMerge`.

Details on trie visualization

The package has the function `TrieToRules` that converts a trie into a list of rules:

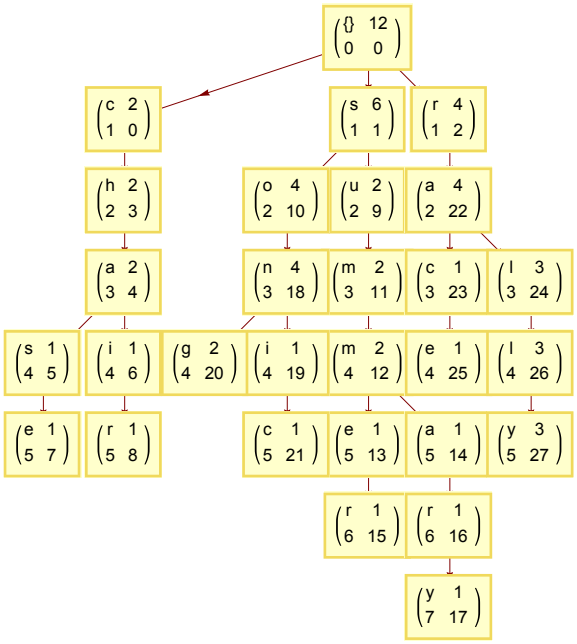
In[277]:=
```
Magnify[Grid[List /@ TrieToRules[mytrie], Alignment → Left], 0.6]
```

Out[277]=
```
{{{}, 12}, {0, 0}} → {{c, 2}, {1, 0}}
{{{}, 12}, {0, 0}} → {{s, 6}, {1, 1}}
{{{}, 12}, {0, 0}} → {{r, 4}, {1, 2}}
{{c, 2}, {1, 0}} → {{h, 2}, {2, 3}}
{{h, 2}, {2, 3}} → {{a, 2}, {3, 4}}
{{a, 2}, {3, 4}} → {{s, 1}, {4, 5}}
{{a, 2}, {3, 4}} → {{i, 1}, {4, 6}}
{{s, 1}, {4, 5}} → {{e, 1}, {5, 7}}
{{i, 1}, {4, 6}} → {{r, 1}, {5, 8}}
{{s, 6}, {1, 1}} → {{u, 2}, {2, 9}}
{{s, 6}, {1, 1}} → {{o, 4}, {2, 10}}
{{u, 2}, {2, 9}} → {{m, 2}, {3, 11}}
{{m, 2}, {3, 11}} → {{m, 2}, {4, 12}}
{{m, 2}, {4, 12}} → {{e, 1}, {5, 13}}
{{m, 2}, {4, 12}} → {{a, 1}, {5, 14}}
{{e, 1}, {5, 13}} → {{r, 1}, {6, 15}}
{{a, 1}, {5, 14}} → {{r, 1}, {6, 16}}
{{r, 1}, {6, 16}} → {{y, 1}, {7, 17}}
{{o, 4}, {2, 10}} → {{n, 4}, {3, 18}}
{{n, 4}, {3, 18}} → {{i, 1}, {4, 19}}
{{n, 4}, {3, 18}} → {{g, 2}, {4, 20}}
{{i, 1}, {4, 19}} → {{c, 1}, {5, 21}}
{{r, 4}, {1, 2}} → {{a, 4}, {2, 22}}
{{a, 4}, {2, 22}} → {{c, 1}, {3, 23}}
{{a, 4}, {2, 22}} → {{l, 3}, {3, 24}}
{{c, 1}, {3, 23}} → {{e, 1}, {4, 25}}
{{l, 3}, {3, 24}} → {{l, 3}, {4, 26}}
{{l, 3}, {4, 26}} → {{y, 3}, {5, 27}}
```

The resulting list of rules can be used in `GraphPlot` and `LayeredGraphPlot`:

In[278]:=
```
LayeredGraphPlot[TrieToRules[mytrie], VertexLabeling → True]
```

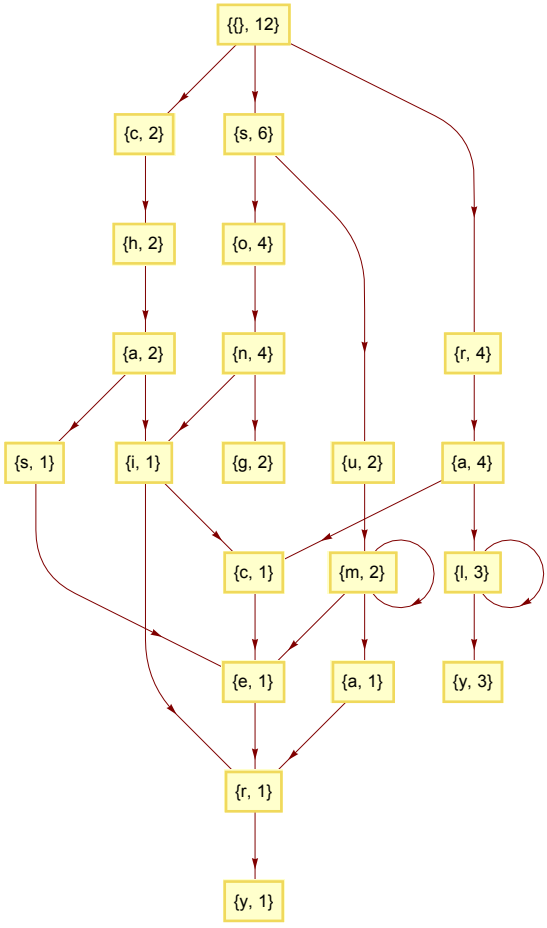
Out[278]=



In order to eliminate ambiguity in the graph plot rules each node has a traversal order attached with it. For example, if we remove the traversal order in the nodes of the graph we get the graph

In[279]:=
```
LayeredGraphPlot[
 Map[Most[#][[1]] &, TrieToRules[mytrie], {2}], VertexLabeling → True]
```

Out[279]=



which is misleading. `TrieForm` uses the lower level function `TrieToRules` which is explained later. (See the code of [1] for details.)

`TrieRemove` has two options “HowManyTimes” and “FrequencyValues” with default values `Automatic`.

In[301]:=
```
Options[TrieRemove]
```

Out[301]=
```
{HowManyTimes → Automatic, FrequencyValues → Automatic}
```

The option "HowManyTimes" takes a number or `Automatic` and it is still under development (its functionality is partially correct). The option "FrequencyValues" takes (`True | False | Automatic`) and it is used for the interpretation of sub-tree removal. If the values are frequencies the removal process is more straightforward. If the values are probabilities the removal process has to take into account that the values on the same level of the sub-tree are conditional probabilities that sum to 1. TrieRemove has two different implementations one for tries with frequencies and one for tries with probabilities. The

option setting "FrequencyValues"->True|False is used to redirect to the correct implementation. When "FrequencyValues"->Automatic is given a simple heuristic is used to determine is the trie argument is a trie with frequencies or a trie with probabilities.

References

[1] Anton Antonov, Tries with frequencies *Mathematica* package, source code at GitHub, https://github.-com/antononcube/MathematicaForPrediction, package TriesWithFrequencies.m, (2013).

[2] Wikipedia, Trie, http://en.wikipedia.org/wiki/Trie .