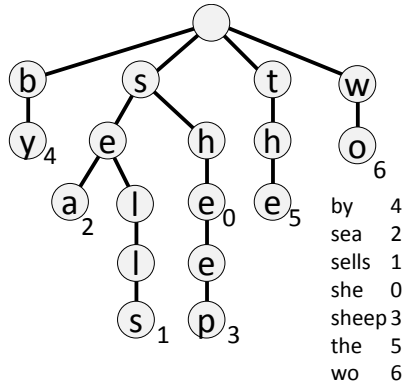
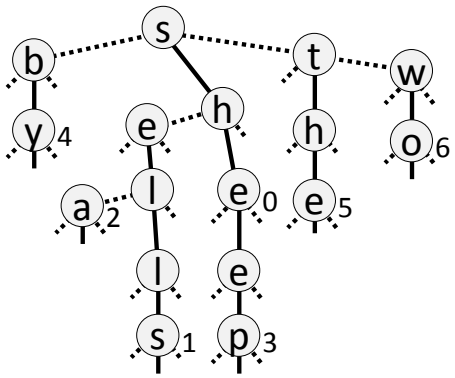
トライ木

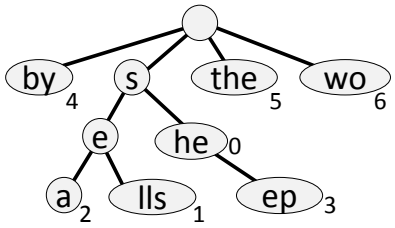
トライ木は、文字列の集合を表現する方法の一つである。他の方法に比べて特に検索時の速度の速さに優れる。具体的な例を図にしめす。ノードそれぞれが１文字を表し、それぞれのノードはアルファベットの数だけ子ノードへのポインタを持つ。子ノードが存在しない場合には、ポインタに null が入る。また、ノードにはラベルが付与可能であり、根ノードからラベルのついている各ノードまでの経路それぞれが、格納されている文字列の１つに対応する。連想記憶として利用する場合には、経路をキーとして、経路の終点ノードのラベルをバリューとして利用する。文字列を検索する場合ときは、単に根ノードから始めて、クエリ文字列を先頭から一文字ずつ見ながら木をたどっていくだけでよい。最後の文字に対応するノードにたどり着いた時点で、そのノードにラベルが付与されていれば検索は成功であり、そのノードにラベルが無い場合や探索の途中で null ポインタが出てきた場合には検索は失敗である。検索が成功した場合の計算量は O(w) (w は当該文字列の長さ)であり、これはハッシュの場合と同様に格納されている要素数によらない値であり、非常に高速である。トライに格納されていない文字列を探した場合（検索失敗）の計算の手間は、平均で O($\log_R N$)である（R はアルファベットの数で N は格納されている要素の数)。トライの欠点は、データを格納するためにメモリを大量に使用する点であり、RN～RNw のメモリを必要とする。これは特に R が大きい場合に問題となる。



トライ木



３分探索木



パトリシア木

３分探索木

トライ木の欠点であるメモリ消費量の問題を解決する手法として ３分探索木(ternary search tree)と呼ばれる方法がある。具体的な例を図に示す。トライ木の問題点は、各ノードがアルファベットの数だけリンクを持っているという点であり、これは null ポインタが多い場合に無駄が多い。たとえば、あるノードが子ノードを少数しか持っていないとすると、残りのポインタは無駄である。３分探索木では、親ノードが子ノードの数だけポインタを持つのでなく、親ノードは１つだけ子ノードへのポインタを持ち、子ノードが兄弟へのポインタを持つことで、上記の無駄をなくしている。

パトリシア木（基数木）

トライ木におけるメモリ消費量の問題を解決する別の手法として パトリシア木と呼ばれる方法がある。具体的な例を図に示す。パトリシア木では、トライ木における「子ノードを一つしかないノードの列」をまとめてひとつのノードにする。したがって、パトリシア木では、すべての中間ノードは２つ以上の子を持つ。