Radix tree

From Wikipedia, the free encyclopedia

In computer science, a radix tree is a space-optimized trie data structure where each node with only one child is merged with its child. The result is that every internal node has at least two children. Unlike in regular tries, edges can be labeled with sequences of characters as well as single characters. This makes them much more efficient for small sets (especially if the strings are long) and for sets of strings that share long prefixes.

It supports the following main operations, all of which are \( O(k) \), where \( k \) is the maximum length of all strings in the set:

- **Lookup**: Determines if a string is in the set. This operation is identical to tries except that some edges consume multiple characters.
- **Insert**: Add a string to the tree. We search the tree until we can make no further progress. At this point we either add a new outgoing edge labeled with all remaining characters in the input string, or if there is already an outgoing edge sharing a prefix with the remaining input string, we split it into two edges (the first labeled with the common prefix) and proceed. This splitting step ensures that no node has more children than there are possible string characters.
- **Delete**: Delete a string from the tree. First, we delete the corresponding leaf. Then, if its parent only has one child remaining, we delete the parent and merge the two incident edges.
- **Find predecessor**: Locates the largest string less than a given string, by lexicographic order.
- **Find successor**: Locates the smallest string greater than a given string, by lexicographic order.

A common extension of radix trees uses two colors of nodes, 'black' and 'white'. To check if a given string is stored in the tree, the search starts from the top and follows the edges of the input string until no further progress can be made. If the search-string is consumed and the final node is a black node, the search has failed; if it is white, the search has succeeded. This enables us to add a large range of strings with a common prefix to the tree, using white nodes, then remove a small set of "exceptions" in a space-efficient manner by inserting them using black nodes.
Applications

As mentioned, radix trees are useful for constructing associative arrays with keys that can be expressed as strings. They find particular application in the area of IP routing, where the ability to contain large ranges of values with a few exceptions is particularly suited to the hierarchical organization of IP addresses. They are also used for inverted indexes of text documents in information retrieval.

History

Donald R. Morrison first described what he called "Patricia trees" in 1968; the name comes from the acronym PATRICIA, which stands for "Practical Algorithm To Retrieve Information Coded In Alphanumeric". Gernot Gwehenberger independently invented and described the data structure at about the same time.

Comparison to other data structures

(In the following comparisons, it is assumed that the keys are of length \(k\) and the data structure contains \(n\) elements.)

Unlike balanced trees, radix trees permit lookup, insertion, and deletion in \(O(k)\) time rather than \(O(\log n)\). This doesn't seem like an advantage, since normally \(k \geq \log n\), but in a balanced tree every comparison is a string comparison requiring \(O(k)\) worst-case time, many of which are slow in practice due to long common prefixes. In a trie, all comparisons require constant time, but it takes \(m\) comparisons to look up a string of length \(m\). Radix trees can perform these operations with fewer comparisons and require many fewer nodes.

Radix trees also share the disadvantages of tries, however: as they can only be applied to strings of elements or elements with an efficiently reversible mapping (injection) to strings, they lack the full generality of balanced search trees, which apply to any data type with a total ordering. A reversible mapping to strings can be used to produce the required total ordering for balanced search trees, but not the other way around. This can also be problematic if a data type only provides a comparison operation, but not a (de)serialization operation.

Hash tables are commonly said to have expected \(O(1)\) insertion and deletion times, but this is only true when considering computation of the hash of the key to be a constant time operation. When hashing the key is taken into account, hash tables have expected \(O(k)\) insertion and deletion times, but may take longer in the worst-case depending on how collisions are handled. Radix trees have worst-case \(O(k)\) insertion and deletion. The successor/predecessor operations of radix trees are also not implemented by hash tables.
### Variants

The **HAT-trie** is a radix tree based cache-conscious data structure that offers efficient string storage and retrieval, and ordered iterations. Performance, with respect to both time and space, is comparable to the cache-conscious hashtable. [4] [5]

### References

2. ^ Morrison, Donald R. Practical Algorithm to Retrieve Information Coded in Alphanumeric

### External links

- Algorithms and Data Structures Research & Reference Material: PATRICIA, by Lloyd Allison, Monash University
- Patricia Tree, NIST Dictionary of Algorithms and Data Structures
- Crit-bit trees, by Daniel J. Bernstein
- Radix Tree API in the Linux Kernel, by Jonathan Corbet
- Kart (key alteration radix tree), by Paul Jarc

### Implementations

- GNU C++ Standard library has a trie implementation
- A heavily commented dictionary implementation with a binary Radix Tree, by Herbert Glamer (in Linoleum, a cross platform assembler)
- Java implementation of Radix Tree, by Tahseen Ur Rehman
- C# implementation of a Radix Tree
- Practical Algorithm Template Library, a C++ library on PATRICIA tries (VC++ >=2003, GCC G++ 3.x), by Roman S. Klyuykov
- Patricia Trie C++ template class implementation, by Radu Gruian
- Haskell standard library implementation "based on big-endian patricia trees". Web-browsable source code
- Patricia Trie implementation in Java, by Roger Kapsi and Sam Berlin
- Crit-bit trees forked from C code by Daniel J. Bernstein
- Patricia Trie implementation in C, in libcprops
<table>
<thead>
<tr>
<th>Binary trees</th>
<th>Binary search tree (BST) · Cartesian tree · Top Tree · T-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-balancing binary search trees</td>
<td>AA tree · AVL tree · Red-black tree · Scapegoat tree · Splay tree · Treap</td>
</tr>
<tr>
<td>B-trees</td>
<td>B+ tree · B*-tree · B*-tree · UB-tree · 2-3 tree · 2-3-4 tree · (a,b)-tree · Dancing tree · Htree</td>
</tr>
<tr>
<td>Tries</td>
<td>Suffix tree · Radix tree · Ternary search tree</td>
</tr>
<tr>
<td>Binary space partitioning (BSP) trees</td>
<td>Quadtree · Octree · kd-tree (implicit) · VP-tree</td>
</tr>
<tr>
<td>Non-binary trees</td>
<td>Exponential tree · Fusion tree · Interval tree · PQ tree · Range tree · SPQR tree · Van Emde Boas tree</td>
</tr>
<tr>
<td>Spatial data partitioning trees</td>
<td>R-tree · R+ tree · R* tree · X-tree · M-tree · Segment tree · Fenwick tree · Hilbert R-tree</td>
</tr>
<tr>
<td>Other trees</td>
<td>Heap · Hash tree · Finger tree · Metric tree · Cover tree · BK-tree · Doubly-chained tree · iDistance · Link-cut tree</td>
</tr>
</tbody>
</table>

Categories: Trees (structure) | String data structures

This page was last modified on 24 March 2011 at 07:51.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.