Module 6: Tries

CS 240 - Data Structures and Data Management

Mark Petrick
Based on lecture notes by many previous cs240 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2017
Tries

- **Trie (Radix Tree):** A dictionary for binary strings
  - Comes from retrieval, but pronounced “try”
  - A binary tree based on **bitwise comparisons**
  - Similar to **radix sort:** use individual bits, not the whole key

- **Structure of trie:**
  - A left child corresponds to a 0 bit
  - A right child corresponds to a 1 bit

- Keys can have different number of bits

- Keys are not stored in the trie: a node $x$ is flagged if the path from root to $x$ is a binary string present in the dictionary
Tries

- Example: A trie for
  \[ S = \{00, 0001, 01001, 011, 01101, 01111, 110, 1101, 111\} \]
Tries: Search

\textbf{Search}(x):

- start from the root
- take the left link if the current bit in $x$ is 0 and take the right link if it is 1; return failure if the link is missing
- if there are no extra bits in $x$ left and the current node is flagged then - success ($x$ is found)
- recurse
Tries: Search

Example: Search(011)
Tries: Insert

- **Insert**(x)
  - Search for x, and suppose we finish at a node v
    - Note: x may have extra bits.
  - Expand the trie from the node v by adding necessary nodes that correspond to extra bits of x; flag the last one.
Tries: Insert

Example: Insert(101)
Tries: Insert

Example: Insert(0100)
Tries: Insert

Example: Insert(11101)
Tries: Delete

Delete($x$)

- Search for $x$
- if $x$ found at an internal flagged node, then unflag the node
- if $x$ found at a leaf $v_x$, delete the leaf and all ancestors of $v_x$ until
  * we reach an ancestor that has two children or
  * we reach a flagged node
Tries: Delete

Example: Delete(011)
Tries: Delete

Example: Delete(0001)
Tries: Delete

Example: Delete(01001)
Tries: Operations

- **Search**($x$)
- **Insert**($x$)
- **Delete**($x$)

**Time Complexity of all operations:** $\Theta(|x|)$

$|x|$: length of binary string $x$, i.e., the number of bits in $x$
Compressed Tries (Patricia Tries)

- **Patricia**: Practical Algorithm To Retrieve Information Coded in Alphanumeric
- Introduced by Morrison (1968)
- Reduces **storage requirement**: eliminate unflagged nodes with only one child
- Every path of one-child unflagged nodes is compressed to a single edge
- Each node stores an **index** indicating the next bit to be tested during a search (index= 0 for the first bit, index= 1 for the second bit, etc)
- A compressed trie storing $n$ keys always has at most $n - 1$ internal (non-leaf) nodes
Compressed Tries (Patricia Tries)

- Each node stores an **index** indicating the next bit to be tested during a search
- Example: A trie and the equivalent compressed trie
Compressed Tries: Operations

- **Search($x$):**
  - Follow the proper path from the root down in the tree to a leaf
  - If search ends in an unflagged node, it is unsuccessful
  - If search ends in a flagged node, we need to check if the key stored is indeed $x$
Compressed Tries: Operations

Example: Search(01001) - successful
Compressed Tries: Operations

Example: Search(11) - unsuccessful
Compressed Tries: Operations

Example: Search(101) - unsuccessful
Compressed Tries: Operations

**Delete(\(x\)):**

- Perform Search(\(x\))
- if search ends in an internal node, then
  - ★ if the node has two children, then unflag the node and delete the key
  - ★ else delete the node and make his only child, the child of its parent
- if search ends in a leaf, then delete the leaf and
- if its parent is unflagged, then delete the parent
Compressed Tries: Operations

Example: Delete(110)
Compressed Tries: Operations

Example: Delete(011)
Compressed Tries: Operations

Example: Delete(01101)
Compressed Tries: Operations

- **Insert**($x$):
  - Perform Search($x$)
  - If the search ends at a leaf $L$ with key $y$, compare $x$ against $y$.
  - If $y$ is a prefix of $x$, add a child to $y$ containing $x$.
  - Else, determine the first index $i$ where they disagree and create a **new node** $N$ with index $i$.
    Insert $N$ along the path from the root to $L$ so that the parent of $N$ has index $< i$ and one child of $N$ is either $L$ or an existing node on the path from the root to $L$ that has index $> i$.
    The other child of $N$ will be a **new leaf node** containing $x$.
  - If the search ends at an internal node, we find the key corresponding to that internal node and proceed in a similar way to the previous case.
Multiway Tries

- To represent **Strings** over any **fixed alphabet** $\Sigma$
- Any node will have at most $|\Sigma|$ children
- Example: A trie holding strings \{bear, bell, ben, soul, soup\}
Multiway Tries

- **Compressed** multi-way tries
- Example: A compressed trie holding strings \{bear, bell, be, so, soul, soup\}