CS 240 — Lecture 12

Bartosz Antczak *Instructor: Eric Schost* *February 9, 2017*

Midterm Info

Tries will be the last topic covered that will be on the midterm. Advice to do well on the midterm:

- Understand everything on the assignments. Practice them.
- Understand all of the data structures we've gone through.

12.1 Tries

A **trie** (pronounced "try", even though that's a ridiculous pronunciation personally) is a dictionary for binary strings, structured as a radix tree. A trie is structured such that:

- the left child corresponds to a 0 bit
- the right child corresponds to a 1 bit

A node in the tree is *flagged* if the path from the root to the node is a binary string that's present in the dictionary.

Example 12.1.1. *A trie for a particular set of binary strings (flagged nodes are in pink)*

$S = \{00, 0001, 01001, 011, 01101, 01111, 110, 1101, 111\}$

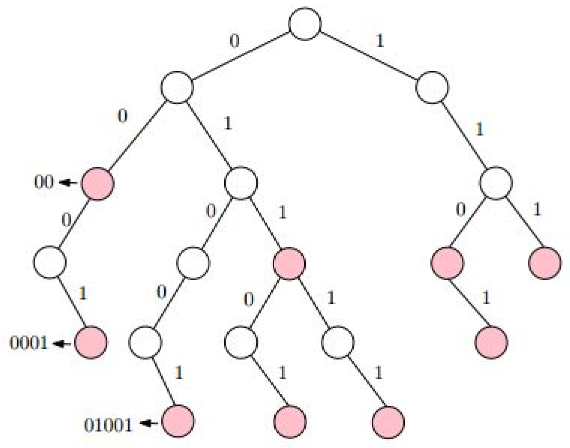


Figure 12.1: Courtesy of the CS 240 lecture slides.

12.1.1 Search

To find a particular string x in the trie:

- Start at the root.
- If the current bit in x is 0, take the right link; otherwise, take the left link. Return failure if the link is missing.
- If there are no extra bits in x left and the current node is flagged, then return success (i.e., x is found). If the node is not flagged, return failure.
- Recurse.

12.1.2 Insert

To insert a binary string x in the trie:

- Search for x, and suppose we finish at node v (we finish at a node if the next bit value is not a child of the current node).
- If x still has extra bits, then expand the trie from node v by adding necessary nodes that correspond to extra bits of x; flag the last one.

12.1.3 Delete

To delete a binary string x in the trie:

- Search for x.
- If x is found at an internal flagged node (i.e., not a leaf), then unflag the node.
- If x is 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.

12.1.4 Operation Time Complexity

All three of these operations have a runtime of $\Theta(|x|)$, where $|x|$ is the length of the binary string (i.e., the number of bits in x).

12.2 Compressed Tries

Also called *Patricia Tries.* Patricia stands for Practical Algorithm To Retrieve Information Coded In Alphanumeric (I wonder how long it took to think of that). This data structure reduces storage requirements by eliminating unflagged nodes with only one child.
Each node stores a number representing the index value in the string that will be tested next (0 for the first bit, 1 for the second bit, etc.; just like an index for an array). A compressed trie storing n keys always has at most $n-1$ internal (non-leaf) nodes.

Example 12.2.1. *A trie with its respective compressed trie*

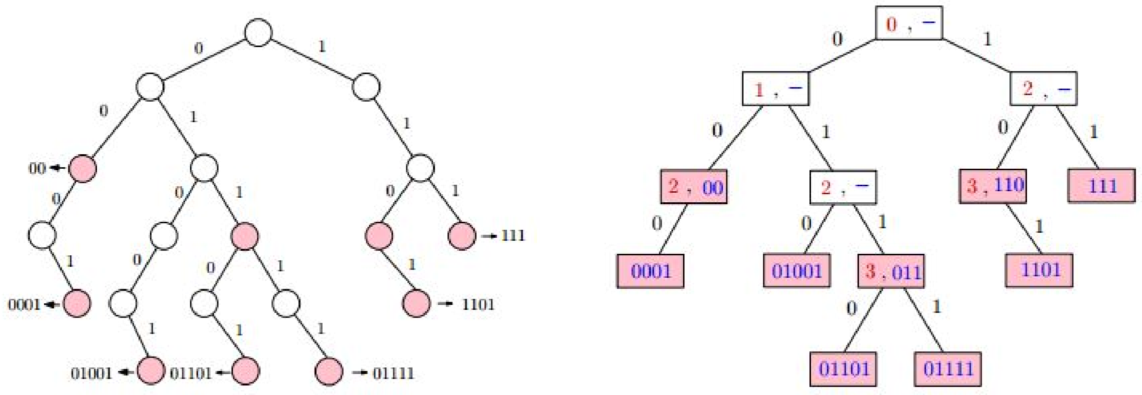


Figure 12.2: Courtesy of the CS 240 lecture slides.

Observe that the right child of the root has been reduced. The right child node contains an index number '2', which means that rather than checking the 1st index of the string, skip directly to the second index and check what value it is.

12.2.1 Claim 1

In a compressed trie with n keys and m internal nodes $m \leq n - 1$

Subclaim 1

In a complete (i.e., every node has 0 or 2 children) binary tree with n leaves, there are $n - 1$ internal nodes

The proof of this claim is done by induction and is left as an exercise.

Define $n = n_1 + n_2 + n_3$ n1 are leaves, n2 have both children, n3 have one child. Delete the n3 nodes (keys with one child), what is left is a complete binary tree with n1 leaves. It must jave n1 - 1 internal nodes. We used to have m internal nodes but deleted n3 of those, so we have m-n3 internal nodes left.

$$m - n_3 = n_1 - 1$$

So m = n1 + n3 - 1 $\leq n_1 + n_2 + n_3 - 1 = n - 1$.

12.2.2 Search in a Compressed Trie

To search for a binary string x in a compressed trie:

- Follow the proper path from the root down in the tree to a leaf.
- If the search ends in an unflagged node, it is unsuccessful.

- If the search ends in a flagged node, we need to check if the key stored in indeed x.

12.2.3 Delete

To delete a string x in a compressed trie:

- Perform `search(x)`.
- If the search ends in an internal node:
  - If the node has two children, then unflag the node and delete the key;
  - otherwise, delete the node and make their only child, the child of its parent
- If the search ends in a leaf, then delete the leaf, and
- if its parent is unflagged, then delete the parent.

12.2.4 Insert

This one's a doozy. To insert a string x into a compressed trie:

- 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 (i.e., they're not the same bit value) 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 less than i and one child of N is either L or an existing node on the path from the root to L that has index greater than 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.

Example 12.2.2. *Insert 1110 into an existing trie.*

- Search, arrive at 1100.
- The first index where 1100 and 1110 differ is at index 2 $\rightarrow$ so create a new node to test index 2.

Example 12.2.3. *Insert 001000 into an existing trie.*

- Search for 001000. Arrive at 001010.
- The first index where 001000 and 001010 differ is at index 4. So we create a new node at index 4.

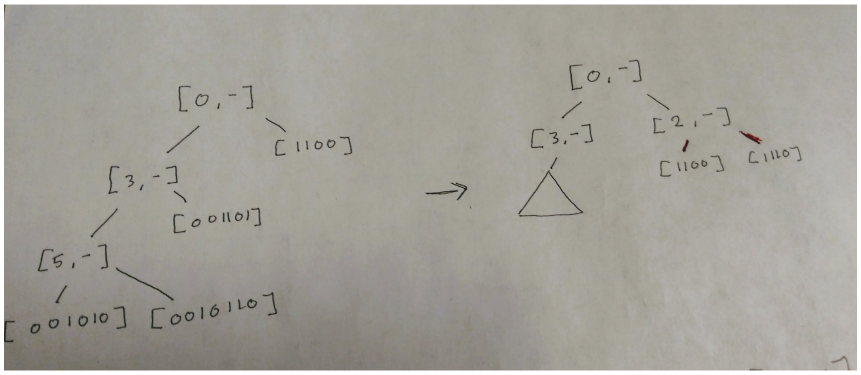


Figure 12.3: A visual example for example 12.2.2

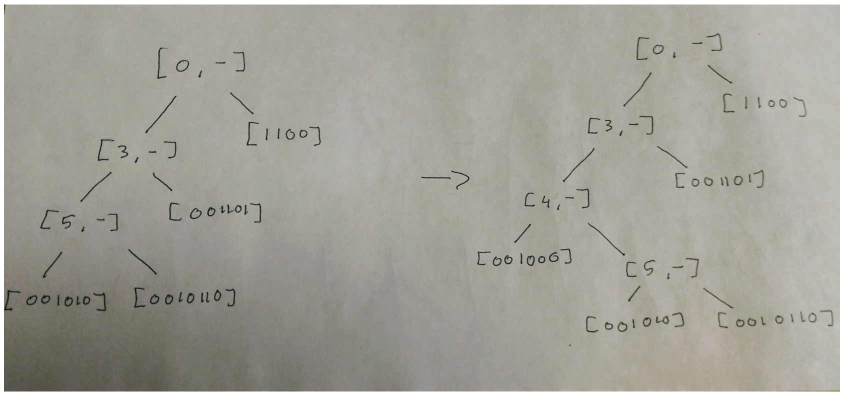
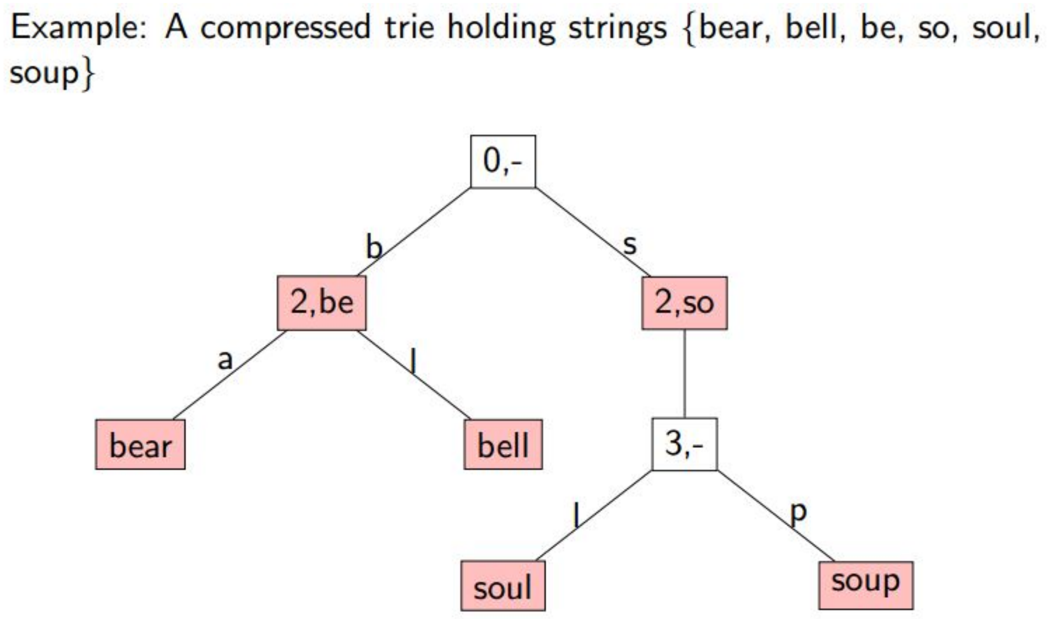


Figure 12.4: A visual example for example 12.2.3

12.3 Multiway Tries

Tries don't just have to represent binary strings. We can use **multiway tries** to represents any fixed alphabet Σ. In a trie, any node will have at most $|\Sigma|$ children. Of course, we can also implement a compressed multiway trie.

Example 12.3.1. *A compressed multiway trie (on the next page)*



Example: A compressed trie holding strings {bear, bell, be, so, soul, soup}

Figure 12.5: Courtesy of the CS 240 lecture slides. (I couldn't format this properly, so I'll just leave this here)