Chapter 12
Digital Search Structures

- Digital Search Trees
- Binary Tries and Patricia
- Multiway Tries

Digital Search Tree

- A digital search tree is a binary tree in which each node contains one element.
- Assume fixed number of bits.
- Not empty
  - Root contains one dictionary pair (any pair).
  - All remaining pairs whose key begins with a 0 are in the left subtree.
  - All remaining pairs whose key begins with a 1 are in the right subtree.
  - Left and right subtrees are digital subtrees on remaining bits.
Example of Digital Search Tree

- Start with an empty digital search tree and **insert a pair whose key is 0110**.
- Now, **insert a pair whose key is 0010**.
- Now, **insert a pair whose key is 1001**.

Example

- Now, **insert a pair whose key is 1011**.
- Now, **insert a pair whose key is 0000**.
Search/Insert/Delete

- Complexity of each operation is $O(\#\text{bits in a key})$.
- \#key comparisons = $O(\text{height})$.
- Expensive when keys are very long.

Applications of Digital Search Trees

- Analog of radix sort to searching.
- Keys are binary bit strings.
  - **Fixed length** – 0110, 0010, 1010, 1011.
  - **Variable length** – 01, 00, 101, 1011.
  - IPv4 – 32 bit IP address.
  - IPv6 – 128 bit IP address.
**Binary Trie**

- Information Retrieval.
- At most one key comparison per operation, search/insert/delete.
- A Binary trie (pronounced *trie*) is a binary tree that has two kinds of nodes: branch nodes and element nodes. For fixed length keys,
  - **Branch nodes**: Left and right child pointers. No data field(s).
  - **Element nodes**: No child pointers. Data field to hold dictionary pair.

---

**Example of Binary Trie**

At most one key comparison for a search.
Variable Key Length

- Left and right child fields.
- Left and right pair fields.
  - Left pair is pair whose key terminates at root of left subtree or the single pair that might otherwise be in the left subtree.
  - Right pair is pair whose key terminates at root of right subtree or the single pair that might otherwise be in the right subtree.
  - Field is null otherwise.

Example of Variable Key Length

At most one key comparison for a search.
Fixed Length Insert

Insert 0111.

Zero compares.

Now, Insert 1101
Fixed Length Insert

Insert 1101

Inserted 1101.

One compare.
Fixed Length Delete

Now, Delete 0111.

Delete 0111.

One compare.
Fixed Length Delete

Now, Delete 1100.

Delete 1100.
Delete 1100.

Fixed Length Delete

Delete 1100.
Fixed Length Delete

Delete 1100. One compare.

Fixed Length Join(S,m,B)

- Insert \( m \) into \( B \) to get \( B' \).
- \( S \) empty \( \Rightarrow \) \( B' \) is answer; done.
- \( S \) is element node \( \Rightarrow \) insert \( S \) element into \( B' \); done;
- \( B' \) is element node \( \Rightarrow \) insert \( B' \) element into \( S \); done;
- If you get to this step, the roots of \( S \) and \( B' \) are branch nodes.
**Fixed Length Join(S,m,B)**

- **S has empty right subtree.**
  - \( J(S, B') = J(a, b) + J(S, B') \)
  - \( J(X, Y) \) = join \( X \) and \( Y \), all keys in \( X < \) all in \( Y \).

- **S has nonempty right subtree.**
  - Left subtree of \( B' \) must be empty, because all keys in \( B' > \) all keys in \( S \).

  \[ \begin{align*}
  &J(S, B') = J(a, b) + J(b, c) \\
  \text{Complexity} = O(\text{height}).
  \end{align*} \]

**Compressed Binary Tries**

- No branch node whose degree is 1.
- Add a **bit#** field to each branch node.
- **bit#** tells you which bit of the key to use to decide whether to move to the left or right subtrie.
**Example: Binary Trie**

- Bit# field shown in black outside branch node.

**Example: Compressed Binary Trie**

- Bit# field shown in black outside branch node.
- #branch nodes = n – 1.
Now, Insert 0010.

Example: After Inserting 0010

Now, Insert 0100.
Example: Insert 0100

Delete

Now, Delete 0010.
Example: After Deleting 0010

Now, Delete 1001.

Example: After Deleting 1001
Patricia

- Practical Algorithm To Retrieve Information Coded In Alphanumeric.
- All nodes in *Patricia* structure are of the same data type (binary tries use branch and element nodes).
  - Pointers to only one kind of node.
  - Simpler storage management.
- Uses a header node that has zero bitNumber. Remaining nodes define a trie structure that is the left subtree of the header node. (right subtree is not used)
- Trie structure is the same as that for the compressed binary trie.

Node Structure

<table>
<thead>
<tr>
<th>bit#</th>
<th>LC</th>
<th>Pair</th>
<th>RC</th>
</tr>
</thead>
</table>

- bit# = bit used for branching
- LC = left child pointer
- Pair = dictionary pair
- RC = right child pointer
Move each element into an ancestor or header node.

Example:
Compressed Binary Trie To Patricia
Insert

- Insert 0000101

- Insert 0000000

- Now, Insert 0000010

- Inserted 0000010
Delete

- Let $p$ be the node that contains the dictionary pair that is to be deleted.
  - Case 1: $p$ has one self pointer.
  - Case 2: $p$ has no self pointer.

$p$ Has One Self Pointer

- $p = \text{header} \Rightarrow$ trie is now empty.
  - Set trie pointer to null.
- $p \neq \text{header} \Rightarrow$ remove node $p$ and update pointer to $p$. 

![Diagram](image1.png)
Let q be the node that has a back pointer to p.
Node q was determined during the search for the pair with the delete key k.

Blue pointer could be red or black.

Use the key y in node q to find the unique node r that has a back pointer to node q.
p Has No Self Pointer

- Copy the pair whose key is \( y \) to node \( p \).

- Change back pointer to \( q \) in node \( r \) to point to node \( p \).
**p Has No Self Pointer**
- Change forward pointer to q from parent(q) to child of q.

Node q now has been removed from trie.

**Multiway Tries**
- Key = Social Security Number.
  - 441-12-1135
  - 9 decimal digits.
- 10-way trie (order 10 trie).

Height <= 10.
Social Security Trie

- **10-way trie**
  - Height \( \leq 10 \).
  - Search: \( \leq 9 \) branches on digits plus 1 compare.

- **100-way trie**
  - 441-12-1135
  - Height \( \leq 6 \).
  - Search: \( \leq 5 \) branches on digits plus 1 compare.

Social Security AVL & Red-Black

- **Red-black tree**
  - Height \( \leq 2\log_2 10^9 \approx 60 \).
  - Search: \( \leq 60 \) compares of 9 digit numbers.

- **AVL tree**
  - Height \( \leq 1.44\log_2 10^9 \approx 40 \).
  - Search: \( \leq 40 \) compares of 9 digit numbers.

- **Best binary tree.**
  - Height \( = \log_2 10^9 \approx 30 \).
Compressed Social Security Trie

- **char#** = character/digit used for branching.
  - Equivalent to **bit#** field of compressed binary trie.
  - **#ptr** = # of nonnull pointers in the node.

**Branch Node Structure**

<table>
<thead>
<tr>
<th>char#</th>
<th>#ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**Insert**

- Insert 012345678.
  
  ![Insert 012345678 diagram]

- Insert 015234567.
  
  ![Insert 015234567 diagram]

3: The 3rd digit is used for branching
Null pointer fields not shown.
Insert

Insert 012345618.

Insert 011917352.
Insert

Delete

Delete 011917352.
Delete

Delete 012345678.

Delete 015231671.

Delete 012345678.
Variable Length Keys

- Problem arises only when one key is a (proper) prefix of another.

Insert 0123
**Variable Length Keys**

- Add a special end of key character (#) to each key to eliminate this problem.
- Insert 0123

---

End of key character (#) not shown.
Tries With Edge Information

- Add a new field (element) to each branch node.
- New field points to any one of the element nodes in the subtree.
- Use this pointer on way down to figure out skipped-over characters.

Example

- element field shown in blue.
### Trie Characteristics

- Expected height of an order $m$ trie is $\sim \log_m n$.
- Limit height to $h$ (say 6). Level $h$ branch nodes point to buckets that employ some other search structure for all keys in subtrie.
- Switch from trie scheme to simple array when number of pairs in subtrie becomes $\leq s$ (say $s=6$).
- Expected # of branch nodes for an order $m$ trie when $n$ is large and $m$ and $s$ are small is $n/(s \ln m)$.
- Sample digits from right to left (instead of from left to right) or using a pseudorandom number generator so as to reduce trie height.

### Multibit Tries

- Variant of binary trie in which the number of bits (stride) used for branching may vary from node to node.
- Proposed for Internet router applications.
  - Variable length prefixes.
  - Longest prefix match.
- Limit height by choosing node strides.
  - Root stride = 32 $\Rightarrow$ height = 1.
  - Strides of 16, 8, and 8 for levels 1, 2, and 3 $\Rightarrow$ only 3 levels.
### Multibit Trie Example

- **S = 1**: Node whose stride is 1 uses 1 bit for branching.
  - Node has $2^1$ children and $2^1$ element/prefix fields.
  - Prefixes that end at a node are stored in that node.
  - Short prefixes are expanded to length represented by node.
  - When root stride is 3, prefixes whose length is < 3 are expanded to length 3.
  - P = 00* expands to P0 = 000* and P1 = 001*.
  - If Q = 000* already exists P0 is eliminated because Q represents a longer match for any destination.

- **S = 2**: Node whose stride is 2 uses 2 bits for branching.
  - Node has $2^2$ children and $2^2$ element/prefix fields.
  - Prefixes that end at a node are stored in that node.
  - Short prefixes are expanded to length represented by node.
  - When root stride is 3, prefixes whose length is < 3 are expanded to length 3.
  - P = 00* expands to P0 = 000* and P1 = 001*.
  - If Q = 000* already exists P0 is eliminated because Q represents a longer match for any destination.

- **S = 3**: Node whose stride is 3 uses 3 bits for branching.
  - Node has $2^3$ children and $2^3$ element/prefix fields.
  - Prefixes that end at a node are stored in that node.
  - Short prefixes are expanded to length represented by node.
  - When root stride is 3, prefixes whose length is < 3 are expanded to length 3.
  - P = 00* expands to P0 = 000* and P1 = 001*.
  - If Q = 000* already exists P0 is eliminated because Q represents a longer match for any destination.