Tries

R-way tries

Ternary search tries

Reference: Chapter 12, Algorithms in Java, 3rd Edition, Robert Sedgewick.

Symbol Table Review

Symbol table: key-value pair abstraction.

- Insert a value with specified key.
- Search for value given key.
- Delete value with given key.
- Balanced trees use log N key comparisons.
- Hashing uses O(1) probes, but probe proportional to key length.

Are key comparisons necessary? No.
Is time proportional to key length required? No.
Best possible. Examine lg N bits.

This lecture: specialized symbol table for string keys.

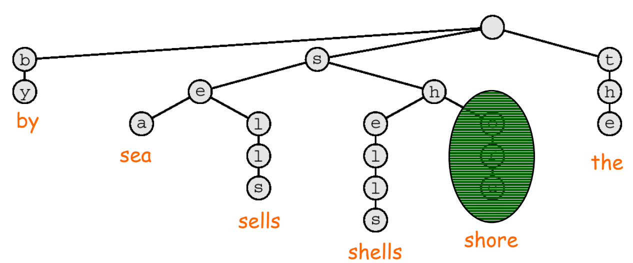
- Faster than hashing.
- More flexible than BST.

Tries

Tries.

- Store characters in internal nodes, not keys.
- Store records in external nodes.
- Use the characters of the key to guide the search.
- NB: from retrieval, but pronounced "try."
- You can get at anything if its organized properly in 40 or 100 bits!

Example: `sells sea shells by the sea shore`



Applications

Applications.

- Spell checkers.
- Data compression. stay tuned
- Princeton U-CALL.
- Computational biology.
- Routing tables for IP addresses.
- Storing and querying XML documents.
- Associative arrays, associative indexing.

Modern application: inverted index of Web.

- Insert each word of every web page into trie, storing URL list in leaves.
- Find query keywords in trie, and take intersection of URL lists.
- Use Pagerank algorithm to rank resulting web pages.

Existence Symbol Table: Operations

Existence symbol table: set of keys.

say, strings over ASCII alphabet

Operations.

- `st.add(key)` inserts a key.
- `st.contains(key)` checks if the key is in the symbol table.

```
ExistenceTable st = new ExistenceTable();
while (!StdIn.isEmpty()) {
   String key = StdIn.readString();
   if (!st.contains(key)) {
      st.add(key);
      System.out.println(key);
   }
}
```

Removes duplicates from input stream

Keys

Key = sequence of "digits."

- DNA: sequence of a,c, g, t.
- Protein: sequence of 20 amino acids A, C, ..., Y.
- IPv6 address: sequence of 128 bits.
- English words: sequence of lowercase letters.
- International words: sequence of UNICODE characters.
- Credit card number: sequence of 16 decimal digits.
- Library call numbers: sequence of letters, numbers, periods.

This lecture: key = string.

- We assume over ASCII alphabet.
- We also assume that character `'\0'` never appears.

Existence Symbol Table: Implementations Cost Summary

	Typical Case			Dedup	
Implementation	Search hit	Insert	Space	Moby	Actors
Input *	L	L	L	0.26	15.1
Red-Black	L + log N	log N	C	1.40	97.4
Hashing	L	L	C	0.76	40.6

Actor: 82MB, 11.4M words, 900K distinct.
Moby: 1.2MB, 210K words, 32K distinct.

N = number of strings
L = size of string
C = number of characters in input
R = radix

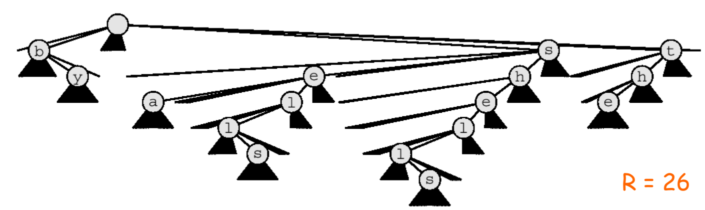
* only reads in data

Challenge: As fast as hashing, as flexible as BST.

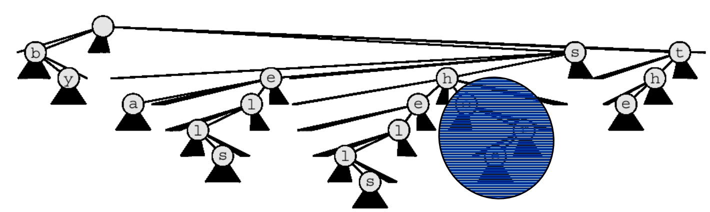
R-Way Existence Trie: Example

Assumption: no string is a prefix of another string.

Ex: `sells sea shells by the sea shore`



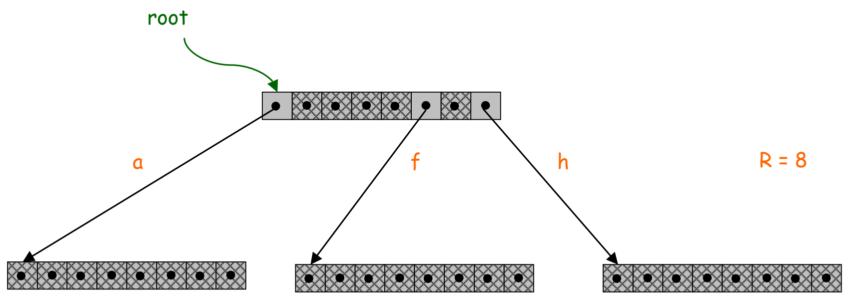
R = 26



R-Way Existence Trie: Java Implementation

R-way existence trie: a node.
Node: reference to R nodes.

```
private static class Node {
   Node[] next = new Node[R];
}
```



R-Way Existence Trie: Implementation

Code is short and sweet.

```
public class RwayExistenceTable {
   private static final int  R   = 128;     ← ASCII
   private static final char END = '\0';    ← sentinel
   private Node root;

   private static class Node {
      Node[] next = new Node[R];
   }


   public boolean contains(String s) {
      return contains(root, s + END, 0);
   }                      ↖ensure no string is a prefix of another

   private boolean contains(Node x, String s, int i) {
      char d = s.charAt(i);
      if (x == null) return false;
      if (d == END)  return (x.next[END] != null);
      return contains(x.next[d], s, i+1);
   }
```


R-Way Existence Trie: Implementation

```
   public void add(String s) {
      root = add(root, s + END, 0);
   }                      ⬆
                  ensure no string is a prefix of another

   private Node add(Node x, String s, int i) {
      char d = s.charAt(i);
      if (x == null) x = new Node();
      if (d == END && x.next[END] == null)
         x.next[END] = new Node();
      if (d == END) return x;
      x.next[d] = insert(x.next[d], s, i+1);
      return x;
   }
}
```


Existence Symbol Table: Implementations Cost Summary

	Typical Case			Dedup	
Implementation	Search hit	Insert	Space	Moby	Actors
Input	L	L	L	0.26	15.1
Red-Black	L + log N	log N	C	1.40	97.4
Hashing	L	L	C	0.76	40.6
R-Way Trie	L	L	R N + C	1.12	Memory
				R = 128	R = 256

R-way trie: Faster than hashing for small R, but slow and wastes memory if R is large.

Goal: Use less space.

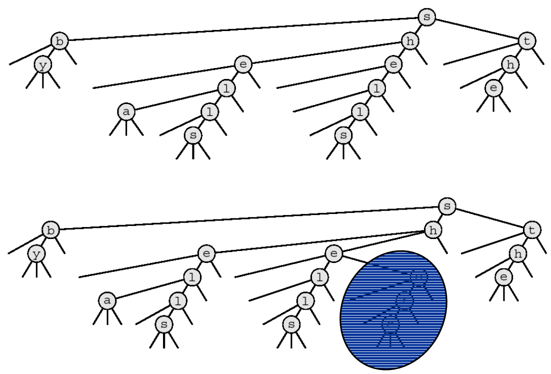
Existence TST

Ternary search trie. Bentley-Sedgewick

- Each node has 3 children:
- Left (smaller), middle (equal), right (larger).

Ex: `sells sea shells by the sea shore`

Observation: Few wasted links!



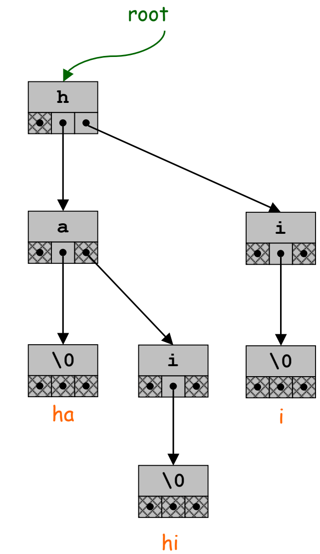
Existence TST: Implementation

Existence TST: a node.

Node: four fields:

- Character d.
- Reference to left TST. smaller
- Reference to middle TST. equal
- Reference to right TST. larger

```
private class Node {
   char d;
   Node l, m, r;
}
```



Existence TST: Java Implementation

```
private boolean contains(Node x, String s, int i) {
   char d = s.charAt(i);
   if (x == null) return false;
   if (d == END && x.d == END) return true;
   if      (d  < x.d) return contains(x.l, s, i);
   else if (d == x.d) return contains(x.m, s, i+1);
   else               return contains(x.r, s, i);
}


private Node add(Node x, String s, int i) {
   char d = s.charAt(i);
   if (x == null) {
      x = new Node();
      x.d = d;
   }
   if (d == END && x.d == END) return x;
   if      (d  < x.d) x.l = add(x.l, s, i);
   else if (d == x.d) x.m = add(x.m, s, i+1);
   else               x.r = add(x.r, s, i);
   return x;
}
```


Existence Symbol Table: Implementations Cost Summary

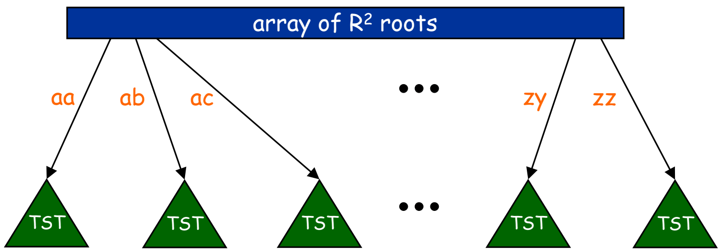
	Typical Case			Dedup	
Implementation	Search hit	Insert	Space	Moby	Actors
Input	L	L	L	0.26	15.1
Red-Black	L + log N	log N	C	1.40	97.4
Hashing	L	L	C	0.76	40.6
R-Way Trie	L	L	R N + C	1.12	Memory
TST	L + log N	L + log N	C	0.72	38.7

↑ no arithmetic

Existence TST With R^2 Branching At Root

Hybrid of R-way and TST.

- Do R-way or R^2-way branching at root.
- Each of R^2 root nodes points to a TST.



Q. What about one letter words?

Existence Symbol Table: Implementations Cost Summary

	Typical Case			Dedup	
Implementation	Search hit	Insert	Space	Moby	Actors
Input	L	L	L	0.26	15.1
Red-Black	L + log N	log N	C	1.40	97.4
Hashing	L	L	C	0.76	40.6
R-Way Trie	L	L	R N + C	1.12	Memory
TST	L + log N	L + log N	C	0.72	38.7
TST with R^2	L + log N	L + log N	C	0.51	32.7

Existence TST Summary

Advantages.

- Very fast search hits.
- Search misses even faster. examine only a few digits of the key!
- Linear space.
- Adapts gracefully to irregularities in keys.
- Supports even more general symbol table ops.

Bottom line: more flexible than BST and can be faster than hashing.

especially if lots of search misses

Existence TST: Other Operations

Delete. Delete key from the symbol table.
Sort. Examine the keys in ascending order.
Find i^{th}. Find the i^{th} largest key.
Range search. Find all elements between k_1 and k_2.

conventional BST ops

Partial match search.

- Use . to match any character.
- `co....er` `.c...c.`

Near neighbor search.

- Find all strings in ST that differ in $\leq$ P characters from query.
- Application: spell checking for OCR.

Longest prefix match.

- Find string in ST with longest prefix match to query.
- Application: search IP database for longest prefix matching destination IP, and route packets accordingly.

additional ops

TST: Partial Matches

Partial match in a TST.
- Search as usual if query character is not a period.
- Go down all three branches if query character is a period.

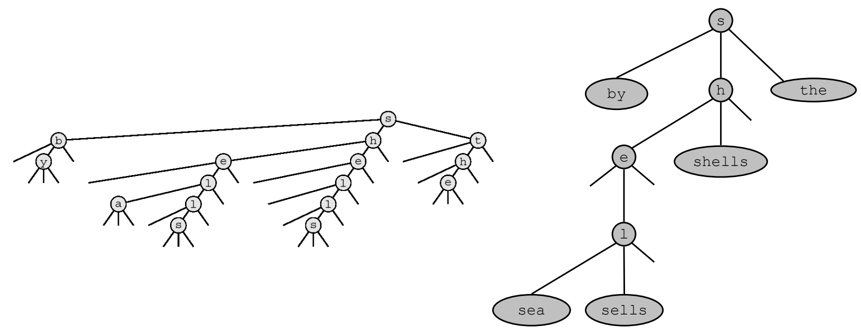
```
private void match(Node x, String s, int i, String prefix) {
   char d = s.charAt(i);
   if (x == null) return;
   if (d == END && x.d == END) System.out.println(prefix);
   if (d == END) return;
   if (d == '.' || d  < x.d) match(x.l, s, i,   prefix);
   if (d == '.' || d == x.d) match(x.m, s, i+1, prefix + x.d);
   if (d == '.' || d  > x.d) match(x.r, s, i,   prefix);
}

public void match(String s) {
      match(root, s + END, 0, "");
}
```

for printing out matches

or use explicit char array for efficiency

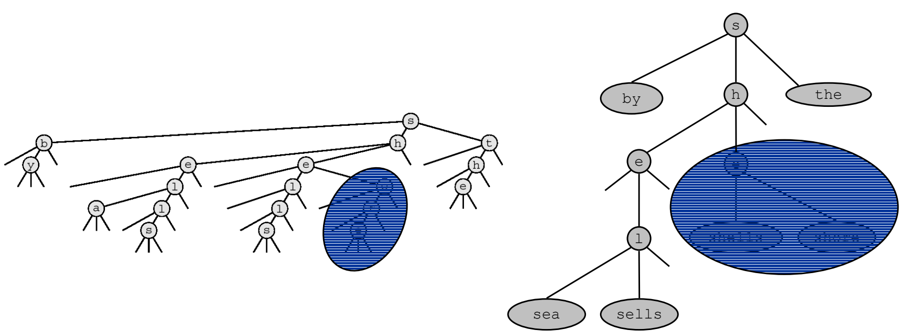
TST Symbol Table

TST implementation of symbol table ADT.
- Store key-value pairs in leaves of trie.
- Search hit ends at leaf with key-value pair; search miss ends at `null` or leaf with different key.
- Internal node stores `char`; external node stores key-value pair.
  - use separate internal and external nodes?
  - collapse (and split) 1-way branches at bottom?



TST Symbol Table

TST implementation of symbol table ADT.
- Store key-value pairs in leaves of trie.
- Search hit ends at leaf with key-value pair; search miss ends at `null` or leaf with different key.
- Internal node stores `char`; external node stores key-value pair.
  - use separate internal and external nodes?
  - collapse (and split) 1-way branches at bottom?



Existence Symbol Table: Implementations Cost Summary

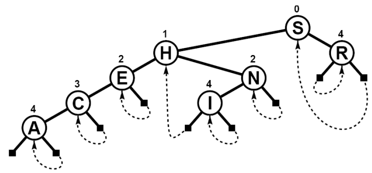
	Typical Case		
Implementation	Search hit	Insert	Space
Input	L	L	L
Red-Black	L + log N	log N	C
Hashing	L	L	C
R-Way Trie	L	L	R N + C
TST	L + log N	L + log N	C
TST with R^2	L + log N	L + log N	C
R-way collapse 1-way	$\log_R N$	$\log_R N$	RN + C
TST collapse 1-way	log N	log N	C

Search, insert time is independent of key length!
- Consequence: can use with very long keys.

PATRICIA Tries

Patricia tries. Practical Algorithm to Retrieve Information Coded in Alphanumeric.

- Collapse one-way branches in binary trie.
- Thread trie to eliminate multiple node types.

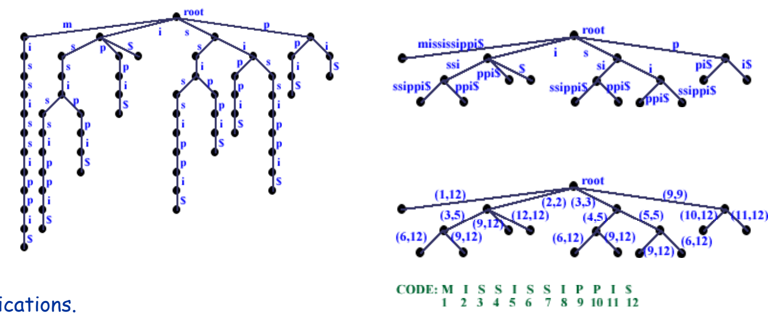


Applications.

- Database search.
- P2P network search.
- IP routing tables: find longest prefix match.
- Compressed quad-tree for N-body simulation.
- Efficiently storing and querying XML documents.

Suffix Tree

Suffix tree: PATRICIA trie of suffixes of a string.



Applications.

- Longest common substring.
- Longest repeated substring.
- Longest palindromic substring.
- Longest common prefix of two substrings.
- Computational biology databases (BLAST, FASTA).
- Search for music by melody.

Associative Arrays

Associative array.

- In Java, C, C++, arrays indexed by integers.
- In Perl, csh, PHP, Python: `president["Princeton"] = "Tilghman"`

```
# collect data
foreach student ($argv)
   foreach input (input100.txt input1000.txt input10000.txt)
      foreach program (worstfit bestfit)
         t[$student][$input][$program] = `time java $program < $input`
      end
   end
end

# compute statistics
. . .
```

Idealized excerpt from COS 226 timing script

Associative Indexing

Associative index.

- Given list of N strings, associate index 0 to N-1 with each string.
- Recall union-find where we assumed objects were labeled 0 to N-1.

Why useful?

- Using algorithm with strings is more useful.
- Running algorithm with indices (instead of ST lookup) is faster.

```
while (true) {
   int p = StdIn.readInt();
   int q = StdIn.readInt();
   ...
   uf.unite(p, q);
   ...
}
```

```
while (true) {
   String s = StdIn.readString();
   String t = StdIn.readString();
   int p = st.index(s);
   int q = st.index(t);
   ...
   uf.unite(p, q);
   ...
}
```

Associative Indexing: Application

Connectivity problem.

- N objects: 0 to N-1
- Find: is there a connection between A and B?
- Union: add a connection between A and B.

Fun version.

- N objects: "Kevin Bacon", "Kate Hudson", . . .
- Find: is there a chain of movies connecting Kevin to Kate?
- Union: Kevin and Kate appeared in "How To Lose a Guy in 10 Days" together, add connection

Real version.

- N objects: "www.cs.princeton.edu", "www.harvard.edu"
- Any graph processing application.

Symbol Table Summary

Hash tables: separate chaining, linear probing.

Binary search trees: randomized, splay, red-black.

Tries: R-way, TST.

Determine the needed ST ops for your application, and choose the best data structure.