String Sets and Symbol Tables

Symbol Table Review

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

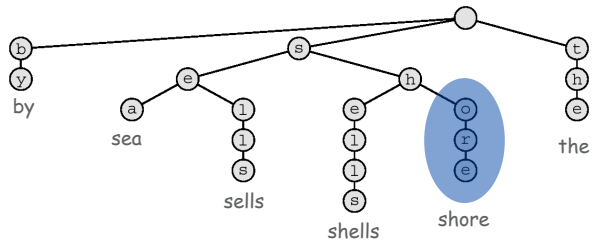
Q. Are key comparisons necessary? No.
Q. Is time proportional to key length required? No.
Best possible. Examine O(log N) bits.

This lecture. Specialized symbol table/set for string keys.
- Faster than hashing.
- More flexible than BST.

Tries

Tries. [from retrieval, but pronounced "try"]
- Store characters in internal nodes, not keys.
- Store records in external nodes.
- Use the characters of the key to guide the search.

Ex: `sells sea shells by the sea` **`shore`**



Applications

Applications.
- Spell checkers.
- Auto-complete.
- Data compression. [stay tuned]
- Computational biology.
- Inverted index of Web.
- Routing tables for IP addresses.
- Storing and querying XML documents.
- T9 predictive text input for cell phones.

String Set: Operations

Operations.

- `st.add(s)`: insert string `s` into the set.
- `st.contains(s)`: is string `s` in the set?

goal: implement this efficiently

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

Removes duplicates from input stream

Keys

Key = sequence of "digits."

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

This lecture: key = string over ASCII alphabet.

String Set: 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.

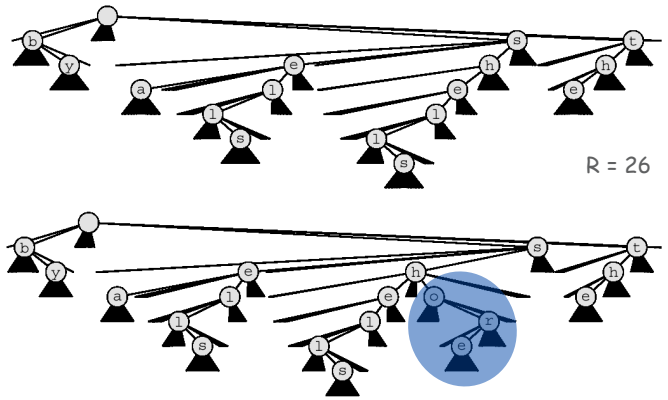
N = number of strings
L = size of string
C = number of characters in input
R = radix

* only reads in data

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

R-Way Trie: Example

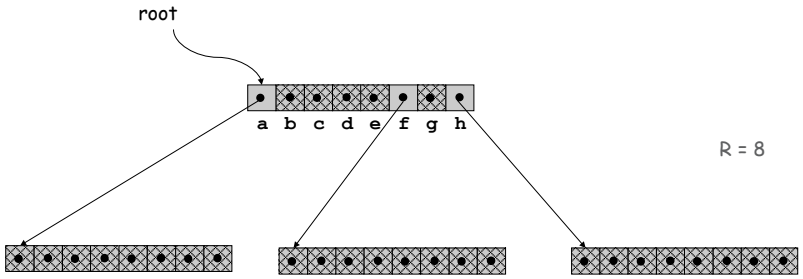
Ex: `sells sea shells by the sea shore`



R-Way Trie: Java Implementation

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

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



R-Way Trie: Java Implementation

```
public class StringSET {
   private static final int R = 128;   ← ASCII
   private Node root new Node();

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

   public boolean contains(String s) {
      return contains(root, s, 0);
   }

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


R-Way Trie: Java Implementation

```
public void add(String s) {
   add(root, s, 0);
}

private Node add(Node x, String s, int i) {
   if (x == null) x = new Node();
   if (i == s.length()) x.end = true;
   else {
      char c = s.charAt(i);
      x.next[c] = add(x.next[c], 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 for large R.

Challenge. Use less space.

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

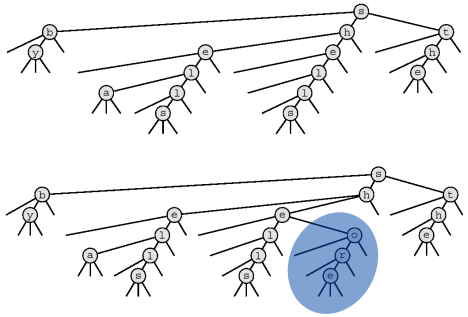
* only reads in data

Ternary Search Trie

Ternary search trie. [Bentley-Sedgewick]

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

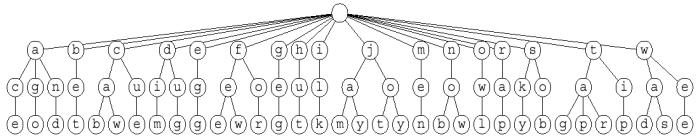
Ex: `sells sea shells by the sea shore`



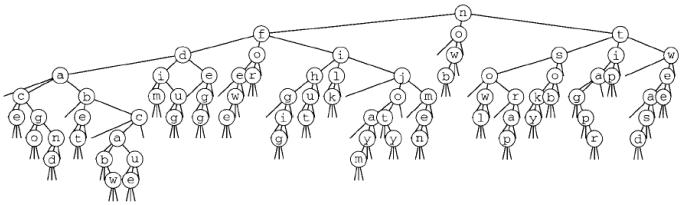
Observation: Few wasted links!

26-Way Trie vs. TST

TST. Collapses empty links in 26-way trie.



26-way trie (1035 null links, not shown)



TST (155 null links)

```
now
for
tip
ilk
dim
tag
jot
sob
nob
sky
hut
ace
bet
men
egg
few
jay
owl
joy
rap
gig
wee
was
cab
wad
caw
cue
fee
tap
ago
tar
jam
dug
and
```

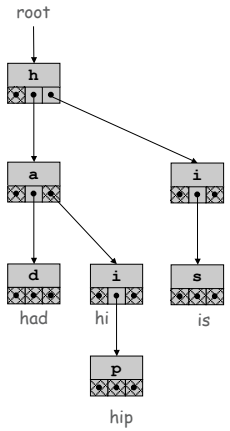

TST Implementation

TST String set: 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 c;
   Node l, m, r;
   boolean end;
}
```



TST: Java Implementation

```
public boolean contains(String s) {
   if (s.length() == 0) return false;
   return contains(root, s, 0);
}

private boolean contains(Node x, String s, int i) {
   if (x == null) return false;
   char c = s.charAt(i);
   if      (c < x.c)           return contains(x.l, s, i);
   else if (c > x.c)           return contains(x.r, s, i);
   else if (i < s.length()-1) return contains(x.m, s, i+1);
   else                        return x.end;
}
```

TST: Java Implementation

```
public void add(String s) {
   root = add(root, s, 0);
}

private Node add(Node x, String s, int i) {
   char c = s.charAt(i);
   if (x == null) x = new Node(c);
   if      (c < x.c)           x.l = add(x.l, s, i);
   else if (c > x.c)           x.r = add(x.r, s, i);
   else if (i < s.length()-1)  x.m = add(x.m, s, i+1);
   else                        x.end = true;
   return x;
}
```


String Set: 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

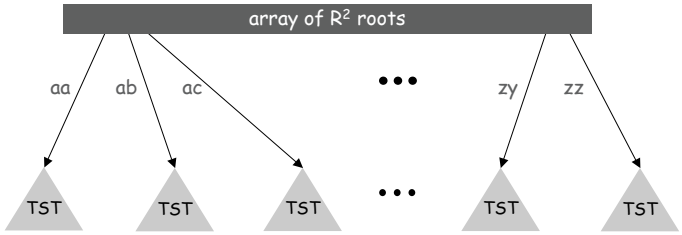
N = number of strings
L = size of string
C = number of characters in input
R = radix

* only reads in data

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?

String Set: 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

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

* only reads in data

TST Summary

Advantages.

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

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

(especially if lots of search misses)

Tries: Advanced Operations

Insert. Insert a key.
Contains. Check if given key in the set.
Delete. Delete key from the symbol table.

} ST ops

Sort. Iterate through the keys in ascending order.
Select. Find the k^{th} largest key.
Range search. Find all elements between k_1 and k_2.

} BST ops

Longest prefix match. Find longest prefix match.
Wildcard match. Allow wildcard characters.
Near neighbor search. Find strings that differ in ≤ P chars.

} Trie ops

(spell checking and OCR)

Longest Prefix Match

Longest prefix match. Find string in set with longest prefix match.

Ex: Search IP database for longest prefix matching destination IP, and route packets accordingly.

```
"128"
"128.112"
"128.112.136"
"128.112.055"
"128.112.055.15"
"128.112.155.11"
"128.112.155.13"
"128.222"
"128.222.136"

prefix("128.112.136.11") = "128.112.136"
prefix("128.166.123.45") = "128"
```


R-way Trie: Longest Prefix Match

Longest prefix match. Search, returning the length of longest prefix match seen so far.

```
public String prefix(String s) {
   int len = prefix(root, s, 0);
   return s.substring(0, len);
}

private int prefix(Node x, String s, int i) {
   if (x == null) return 0;
   int len = 0;
   if (x.end) len = i;
   if (i == s.length()) return len;
   char c = s.charAt(i);
   return Math.max(len, prefix(x.next[c], s, i+1));
}
```

Wildcard Match

Wildcard match. Use wildcard . to match any character.

```
coalizer
coberger
codifier
cofaster
cofather
cognizer
cohelper
colander
coleader
...
compiler
...
composer
computer
cowkeper
```

co....er

```
acresce
acroach
acuracy
octarch
science
scranch
scratch
scrauch
screich
scrinch
scritch
scrunch
scudick
scutock
```

.c...c.

TST: Wildcard Match

Wildcard match. Use wildcard . to match any character.

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

```
public void wildcard(String s) {
   wildcard(root, s, 0, "");
}

private void wildcard(Node x, String s, int i, String prefix) {
   if (x == null) return;
   char c = s.charAt(i);
   if (c == '.' || c <  x.c) wildcard(x.left,  s, i, prefix);
   if (c == '.' || c == x.c) {
      if (i < s.length() - 1)
         wildcard(x.mid, s, i+1, prefix + x.c);
      else if (x.end)
         System.out.println(prefix + x.c);
   }
   if (c == '.' || c >  x.c)  wildcard(x.right, s, i, prefix);
}
```

for printing out matches (use `StringBuilder` for efficiency)

T9 Texting

Goal. Type text messages on a phone keypad.

Multi-tap input. Enter a letter by repeatedly pressing a key until the desired letter appears.

T9 text input. "A much faster and more fun way to enter text."

- Find all words that correspond to given sequence of numbers. (sorted by frequency)
- Press 0 to see all completion options.

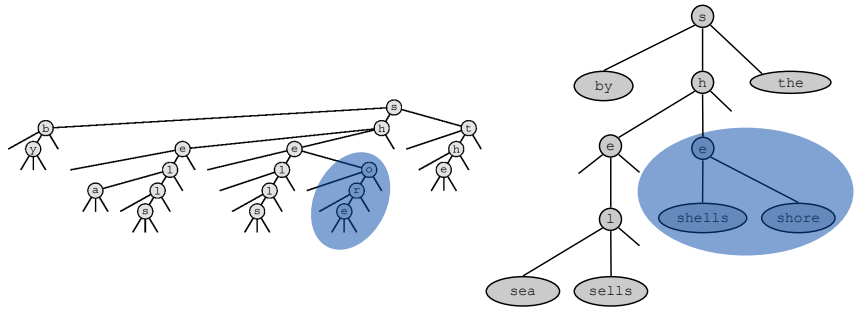
Ex: `hello`

- Multi-tap: `4 4 3 3 5 5 5 5 5 5 6 6 6`
- T9: `4 3 5 5 6`

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

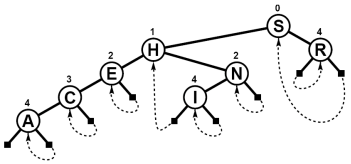
Implementation	Search hit	Insert	Space
Input	L	L	L
Red-Black	L + log N	log N	C
Hashing	L	L	C
R-Way Trie	L	L	R N + C
TST	L + log N	L + log N	C
TST with R^2	L + log N	L + log N	C
R-way collapse 1-way	$\log_R N$	$\log_R N$	RN + C
TST collapse 1-way	log N	log N	C

Key property. 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.

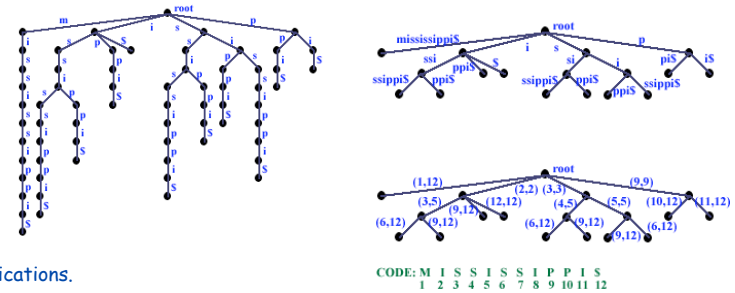


Applications.

- 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.

Symbol Table Summary

Hash tables. Separate chaining, linear probing.

Binary search trees. Randomized, splay, red-black.

Tries. R-way, TST.

Lessons.

- Determine the needed ST ops for your application, and choose the best data structure.
- You can get at anything (if organized properly) in 40 or 100 bits!