Tries

- review
- tries
- TSTs
- applications
- refinements

References: Algorithms in Java, Chapter 15
Intro to Algs and Data Structs, Section 6.2

Review: summary of the performance of searching (symbol-table) algorithms

Frequency of execution of instructions in the inner loop:

iteration?	guarantee search	guarantee insert	guarantee delete	average case search	average case insert	average case delete	ordered iteration?	operations on keys
BST	N	N	N	1.38 lg N	1.38 lg N	?	yes	`Comparable`
randomized BST	7 lg N	7 lg N	7 lg N	1.38 lg N	1.38 lg N	1.38 lg N	yes	`Comparable`
red-black tree	2 lg N	2 lg N	2 lg N	lg N	lg N	lg N	yes	`Comparable`
hashing	1*	1*	1*	1*	1*	1*	no	`equals()` `hashcode()`

Q: Can we do better?

Review

Symbol tables.
- Associate a value with a key.
- Search for value given key.

Balanced trees
- use between lg N and 2 lg N key comparisons
- support ordered iteration and other operations

Hash tables
- typically use 1-2 probes
- require good hash function for each key type

Radix sorting
- some keys are inherently digital
- digital keys give linear and sublinear sorts

This lecture. Symbol tables for digital keys.

Digital keys (review)

Many commonly-use key types are inherently digital (sequences of fixed-length characters)

Examples
- Strings
- 64-bit integers

interface

```
interface Digital
{
   public int charAt(int k);
   public int length(int);
   static int R();
}
```

This lecture:
- refer to fixed-length vs. variable-length strings
- `R` different characters
- key type implements `charAt()` and `length()` methods
- code works for `String` and key types that implement `Digital`.

Digital keys in applications

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 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 ADT

String set. Unordered collection of distinct strings.

API for **`StringSET`**.

- `add(key)` insert the key into the set
- `contains(key)` is the given key in the set?

Typical client: **`Dedup`** (remove duplicate strings from input)

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

This lecture: focus on **`StringSET`** implementation

Same ideas improve STs with wider API

`StringSET` implementation 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

`actors` 82MB, 11.4M words, 900K distinct.
`moby` 1.2MB, 210K words, 32K distinct.

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

* only reads in data

Challenge. Efficient performance for long keys (large L).

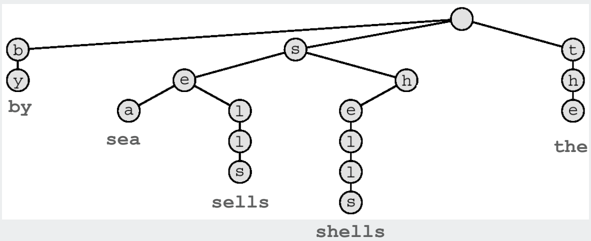
tries
TSTs
applications
refinements

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**

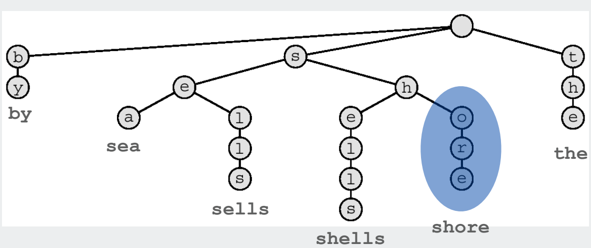


Tries

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**



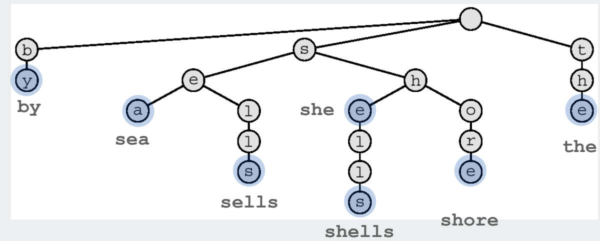
Tries

Q. How to handle case when one key is a prefix of another?

A1. Append sentinel character `'\0'` to every key so it never happens.

A2. Store extra bit to denote which nodes correspond to keys.

Ex. **she sells sea shells by the sea shore**

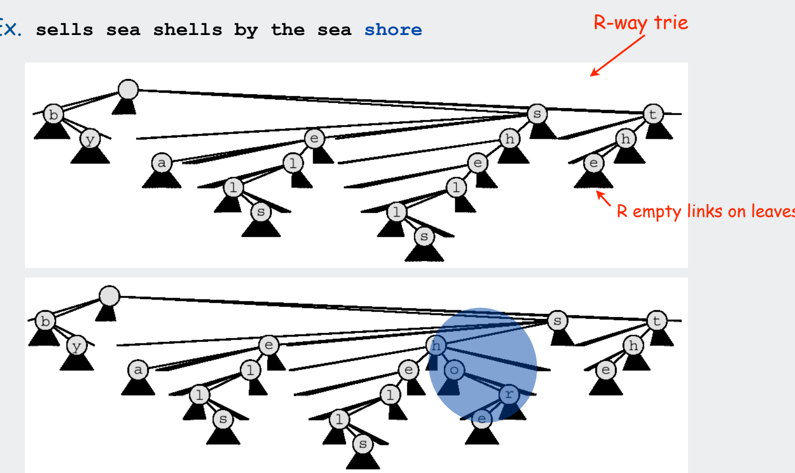


Branching in tries

Q. How to branch to next level?

A. One link for each possible character

Ex. **sells sea shells by the sea shore**

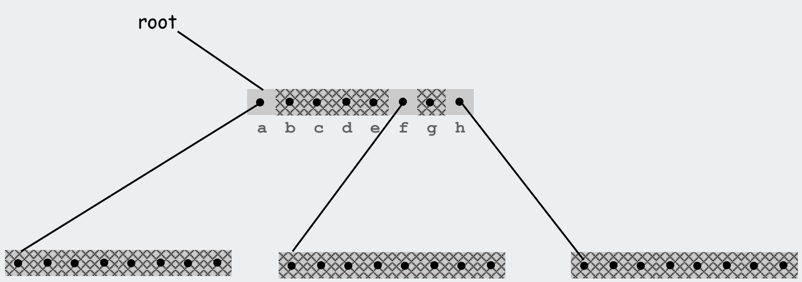


R-Way Trie: Java Implementation

R-way existence trie. A node.

Node. Reference to R nodes.

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



8-way trie that represents {**a**, **f**, **h**}

R-way trie implementation of `StringSET`

empty trie

```
public class StringSET
{
   private static final int R = 128;
   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);
   }

   public void add(String s)
   // see next slide
}
```

current digit

R-way trie implementation of `StringSET` (continued)

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


R-way trie performance characteristics

Time
- examine one character to move down one level in the trie
- trie has ~$\log_R$ N levels (not many!)
- need to check whole string for search hit (equality)
- search miss only involves examining a few characters

Space
- R empty links at each leaf
- 63356-way branching for Unicode impractical

Bottom line.
- method of choice for small R
- you use tries every day
- stay tuned for ways to address space waste

Sublinear search with tries

Tries enable user to present keys one char at a time

Search hit
- can present possible matches after a few digits
- need to examine all L digits for equality

Search miss
- could have mismatch on first character
- typical case: mismatch on first few characters

Bottom line: sublinear search cost (only a few characters)

Further help
- object equality test
- cached hash values

`StringSET` implementation 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	RN + C	1.12	out of memory

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

R-way trie
- faster than hashing for small R
- too much memory if R not small

65536-way trie for Unicode??

Challenge. Use less memory!

Digression: Out of memory?

"640 K ought to be enough for anybody."
- attributed to Bill Gates, 1981
(commenting on the amount of RAM in personal computers)

"64 MB of RAM may limit performance of some Windows XP features; therefore, 128 MB or higher is recommended for best performance." - Windows XP manual, 2002

"64 bit is coming to desktops, there is no doubt about that. But apart from Photoshop, I can't think of desktop applications where you would need more than 4GB of physical memory, which is what you have to have in order to benefit from this technology. Right now, it is costly." - Bill Gates, 2003

Digression: Out of memory?

A short (approximate) history

		address bits	addressable memory	typical actual memory	cost
PDP-8	1960s	12	6K	6K	$16K
PDP-10	1970s	18	256K	256K	$1M
IBM S/360	1970s	24	4M	512K	$1M
VAX	1980s	32	4G	1M	$1M
Pentium	1990s	32	4G	1 GB	$1K
Xeon	2000s	64	enough	4 GB	$100
??	future	128+	enough	enough	$1

A modest proposal

Number of atoms in the universe: < 2^{266} (estimated)
Age of universe (estimated): 20 billion years ~ 2^{50} secs < 2^{80} nanoseconds

How many bits address every atom that ever existed ?

A modest proposal: use a unique 512-bit address for every object

512 bits is enough:

266 bits	80 bits	174 bits
place	time	cushion for whatever

current plan:

128 bits	64 bits
place (ipv6)	place (machine)

Use trie to map to current location. 64 8-bit chars
- wastes 255/256 actual memory (only good for tiny memories or Gates)
- need better use of memory

tries
TSTs
extensions
refinements

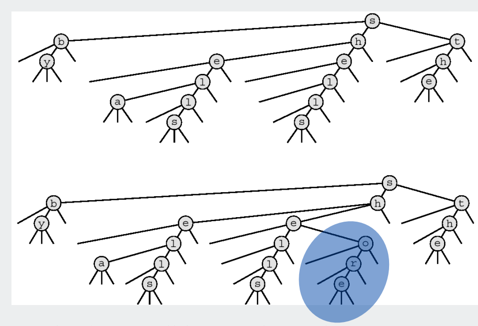
Ternary Search Tries (TSTs)

Ternary search tries. [Bentley-Sedgewick, 1997]
- Store characters in internal nodes, records in external nodes.
- Use the characters of the key to guide the search
- Each node has three children
- Left (smaller), middle (equal), right (larger).

Ternary Search Tries (TSTs)

Ternary search tries. [Bentley-Sedgewick, 1997]
- Store characters in internal nodes, records in external nodes.
- Use the characters of the key to guide the search
- Each node has three children:
  left (smaller), middle (equal), right (larger).

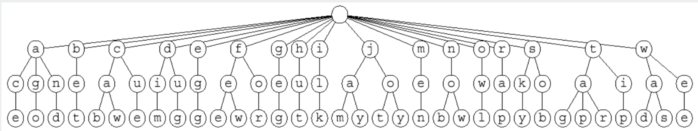
Ex. **sells sea shells by the sea shore**



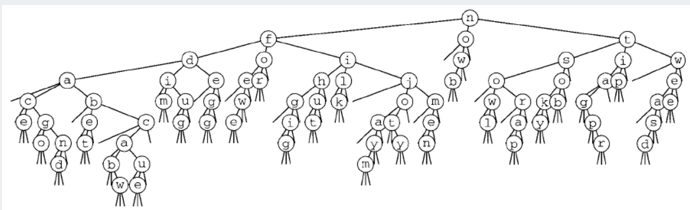
Observation. Only three null links in leaves!

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

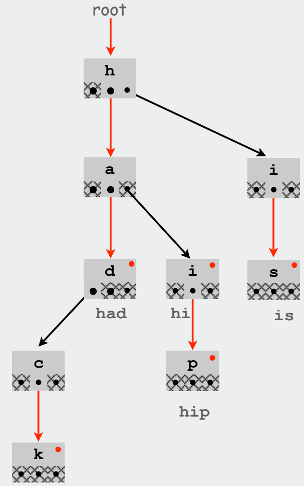
TST representation

A TST string set is a TST node.

A TST node is five fields:

- a character c.
- a reference to a left TST. [smaller]
- a reference to a middle TST. [equal]
- a reference to a right TST. [larger]
- a bit to indicate whether this node is the last character in some key.

```
private class Node
{
   char c;
   Node l, m, r;
   boolean end;
}
```



TST implementation of contains() for StringSET

Recursive code practically writes itself!

```
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 implementation of add() for StringSET

```
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;
}
```

StringSET implementation 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	RN + C	1.12	out of memory
TST	L	L	3C	.72	38.7

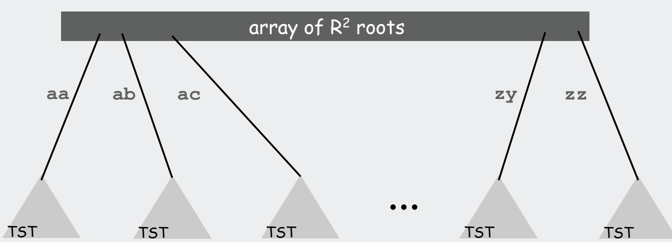
N = number of strings
L = size of string
C = number of characters in input
R = radix

TST
- faster than hashing
- space usage independent of R
- supports extended APIs (stay tuned)
- Unicode no problem

Space-efficient trie: challenge met.

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.



Note. Need special test for one-letter words.

StringSET implementation 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	RN + C	1.12	out of memory
TST	L	L	3C	.72	38.7
TST with R^2	L	L	3C + R^2	.51	32.7

TST performance even better with nonuniform keys

Ex. Library call numbers

```
WUS-------10706-----7---10
WUS-------12692-----4---27
WLSOC------2542----30
LTK--6015-P-63-1988
LDS---361-H-4
 ...
```

TSTs 5 times faster than hashing

Faster than hashing

Hashing.
- need to examine entire key
- hits and misses cost about the same.
- need good hash function for every key type
- no help for ordered-key APIs

TSTs.
- need to examine just enough key characters
- search miss may only involve a few characters
- works only for keys types that implement charAt()
- can handle ordered-key APIs
- more flexible than search trees [stay tuned]

Ex. Library call numbers

```
WUS-------10706-----7---10
WUS-------12692-----4---27
WLSOC------2542----30
LTK--6015-P-63-1988
LDS---361-H-4
 ...
```

TSTs 5 times faster than hashing

tries
TSTs
extensions
refinements

Extending the `StringSET` API

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

equals()

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

Comparable

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

Digital

Longest Prefix Match

Find string in set with longest prefix matching given key.

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 implementation of longest prefix match operation

Find string in set with longest prefix matching a given key.

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

private int prefix(Node x, String s, int i)
{
   if (x == null) return 0;
   int length = 0;
   if (x.end) length = i;
   if (i == s.length()) return length;
   char c = s.charAt(i);
   return Math.max(length, 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 implementation of wildcard match operation

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.

for printing out matches (use StringBuilder for long keys)

```
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);
}
```


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.
- 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`

www.t9.com

A Letter to t9.com

```
To: info@t9support.com
Date: Tue, 25 Oct 2005 14:27:21 -0400 (EDT)

Dear T9 texting folks,

I enjoyed learning about the T9 text system
from your webpage, and used it as an example
in my data structures and algorithms class.
However, one of my students noticed a bug
in your phone keypad

   http://www.t9.com/images/how.gif

Somehow, it is missing the letter s. (!)

Just wanted to bring this information to
your attention and thank you for your website.

Regards,

Kevin
```

where's the "s" ??

A world without "s" ??

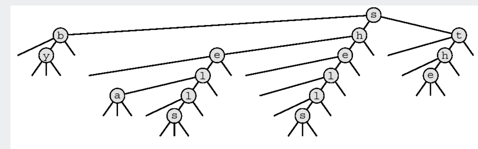
```
To: "'Kevin Wayne'" <wayne@CS.Princeton.EDU>
Date: Tue, 25 Oct 2005 12:44:42 -0700

Thank you Kevin.

I am glad that you find T9 o valuable for your
cla.  I had not noticed thi before.  Thank for
writing in and letting u know.

Take care,

Brooke nyder
OEM Dev upport
AOL/Tegic Communication
1000 Dexter Ave N. uite 300
eattle, WA 98109

ALL INFORMATION CONTAINED IN THIS EMAIL IS CONIDERED
CONFIDENTIAL AND PROPERTY OF AOL/TEGIC COMMUNICATION
```

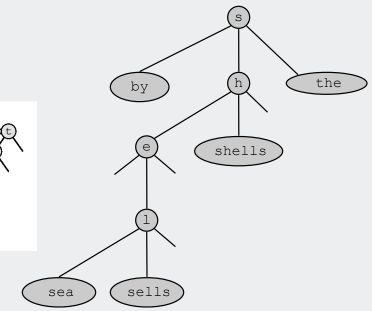

tries
TSTs
extensions
refinements

TST: Collapsing 1-Way Branches

Collapsing 1-way branches at bottom.
- internal node stores `char`; external node stores full key.
- append sentinel character `'\0'` to every key
- search hit ends at leaf with given key.
- search miss ends at `null` link or leaf with different key.

Collapsing interior 1-way branches
- keep char position in nodes
- need full compare at leaf

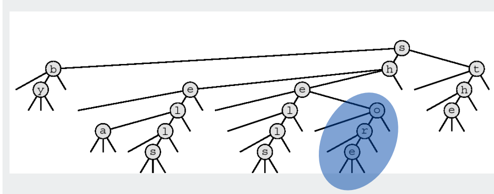


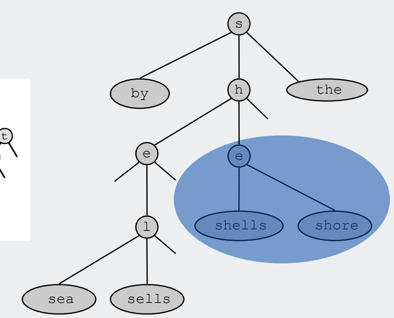


TST: Collapsing 1-Way Branches

Collapsing 1-way branches at bottom.
- internal node stores `char`; external node stores full key.
- append sentinel character `'\0'` to every key
- search hit ends at leaf with given key.
- search miss ends at `null` link or leaf with different key.

Collapsing interior 1-way branches
- keep char position in nodes
- need full compare at leaf





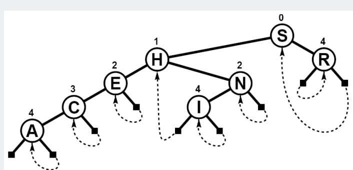
`StringSET` implementation cost summary

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	RN + C
TST	L	L	3C
TST with R^2	L	L	3C + R^2
R-way with no 1-way	$\log_R N$	$\log_R N$	RN + C
TST with no 1-way	log N	log N	C

Challenge met.
- Efficient performance for arbitrarily long keys.
- Search time is independent of key length!

A classic algorithm

Patricia tries. [Practical Algorithm to Retrieve Information Coded in Alphanumeric]
- Collapse one-way branches in binary trie.
- Thread trie to eliminate multiple node types.

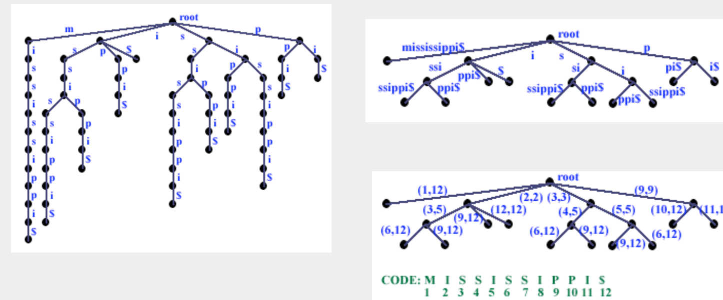


Applications.
- Database search.
- P2P network search.
- IP routing tables: find longest prefix match.
- Compressed quad-tree for N-body simulation.
- Efficiently storing and querying XML documents.

(Just slightly) beyond the scope of COS 226 (see Program 15.7)

Suffix Tree

Suffix tree.
Threaded trie with collapsed 1-way branching for string suffixes.



Applications.
- Longest common substring, longest repeated substring.
- Computational biology databases (BLAST, FASTA).
- Search for music by melody.
- ...

(Just slightly) beyond the scope of COS 226.

Symbol tables summary

A success story in algorithm design and analysis.
Implementations are a critical part of our computational infrastructure.

Binary search trees. Randomized, red-black.
- performance guarantee: log N compares
- supports extensions to API based on key order

Hash tables. Separate chaining, linear probing.
- performance guarantee: N/M probes
- requires good hash function for key type
- no support for API extensions
- enjoys systems support (ex: cached value for String)

Tries. R-way, TST.
- performance guarantee: log N characters accessed
- supports extensions to API based on partial keys

Bottom line: you can get at anything by examining 50-100 bits (!!!)