Tries

- tries
- TSTs
- applications

References:
Algorithms in Java, Chapter 15
`http://www.cs.princeton.edu/algs4/62trie`

Algorithms in Java, 4th Edition · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · April 9, 2008 8:45:36 AM

Review: summary of the performance of symbol-table implementations

Frequency of operations.

implementation	typical case			ordered iteration?	operations on keys
	search	insert	delete		
BST	1.39 lg N	1.39 lg N	?	yes	`compareTo()`
randomized BST	1.39 lg N	1.39 lg N	1.39 lg N	yes	`compareTo()`
red-black tree	lg N	lg N	lg N	yes	`compareTo()`

hashing	1 †	1 †	1 †	no	`equals()`
					`hashcode()`

† assumes random hash code

Q. Can we do better?

A. Yes, if we can avoid examining the entire key, as with radix sorting.

Digital keys (review)

Digital key. Sequence of digits over fixed alphabet.

Radix. Number of digits in alphabet.

Applications.
- 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. `String` of ASCII characters.

String set API

String set. Collection of distinct strings.

```
public class StringSET
```

	`StringSET()`	*create an empty set of strings*
`void`	`add(String key)`	*add a string to the set*
`boolean`	`contains(String key)`	*is key in the set?*

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

dedup client

Remark. Same idea extends to `StringST`.

String set implementations cost summary

	typical case			dedup	
implementation	search hit	insert	space	`moby.txt`	`actors.txt`
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

* only reads in data

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

file	size	words	distinct
`moby.txt`	1.2 MB	210 K	32 K
`actors.txt`	82 MB	11.4 M	900 K

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

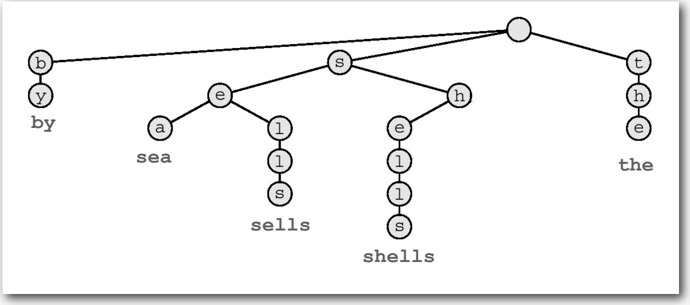
- tries
- TSTs
- applications

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`

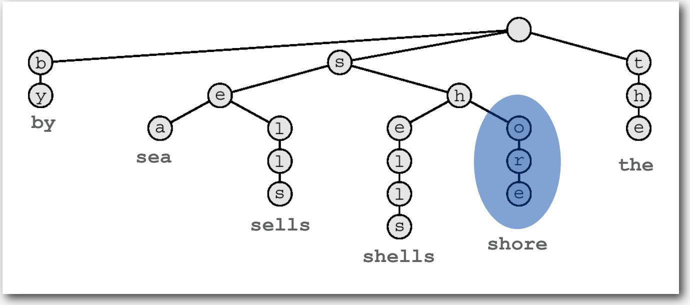


Tries

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`



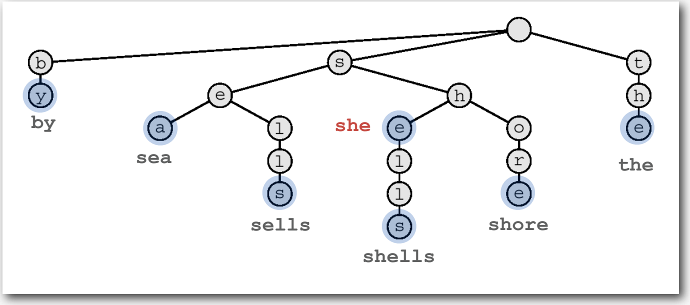
Tries

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`

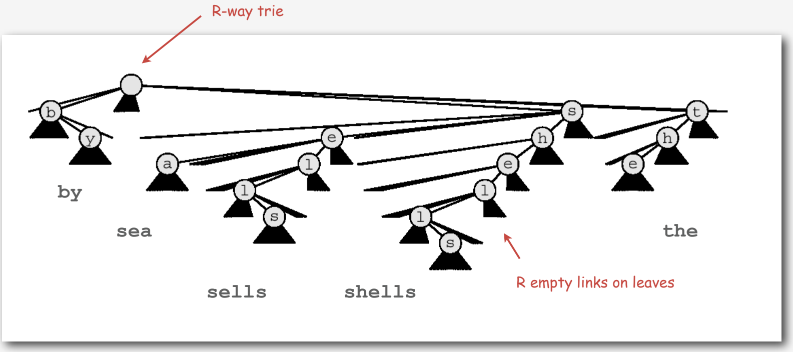


Branching in tries

Q. How to branch to next level?

A. One link for each possible character.

Ex. `sells sea shells by the sea`

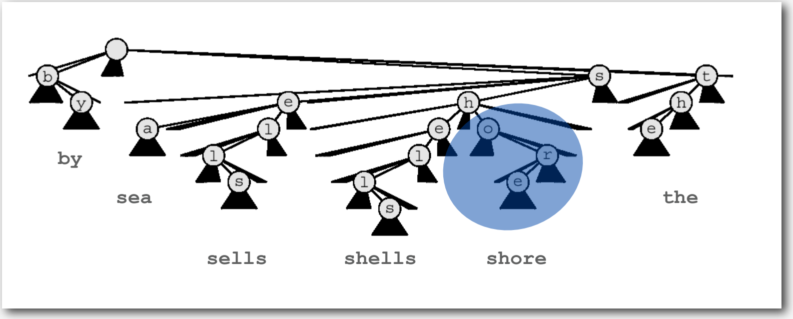


Branching in tries

Q. How to branch to next level?

A. One link for each possible character.

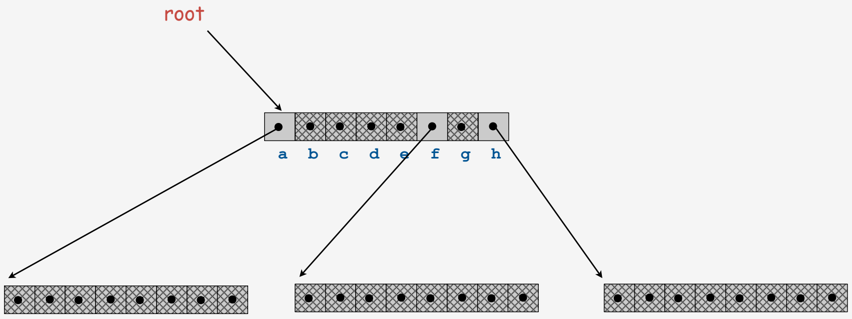
Ex. `sells sea shells by the sea shore`



R-way existence trie: Java implementation

Node. References to R nodes.

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



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

R-way existence trie: Java implementation

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

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

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

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

ASCII

empty trie

current digit

R-way existence trie: Java implementation (cont)

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

private Node add(Node x, String s, int d)
{
   if (x == null) x = new Node();
   if (d == s.length()) x.end = true;
   else
   {
      char c = s.charAt(d);
      x.next[c] = add(x.next[c], s, d+1);
   }
   return x;
}
```

Sublinear search miss with tries

Tries enable user to present string keys one `char` at a time.

Search miss.

- Could have mismatch on first character.
- Typical case: examine only a few characters.

Search hit.

- Need to examine all L characters for equality.
- Can present possible matches after a few characters.

auto-complete
(stay tuned)

Space. R empty links at each leaf.

Bottom line. Fast search hit, sublinear-time search miss.

String set implementations cost summary

	typical case			dedup	
implementation	search hit	insert	space	`moby.txt`	`actors.txt`
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

R-way trie.

- Method of choice for small R.
- Too much memory for large R.

Challenge. Use less memory, e.g., 65,536-way trie for Unicode!

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.

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

> " *512-bit words ought to be enough for anybody.* "
> *— Kevin Wayne, 2003*

A modest proposal

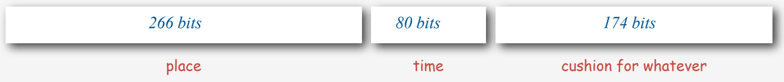
Number of atoms in the universe (estimated). ≤ 2^{266}.

Age of universe (estimated). 20 billion years ~ 2^{50} seconds ≤ 2^{80} nanoseconds.

Q. How many bits address every atom that ever existed?

A. Use a unique 512-bit address for every object.

Observation. 512 bits ought to be enough.



Use trie to map to current location.

- Represent location as 64 8-bit chars (512 bits).
- 256-way trie wastes 255/256 actual memory.
- Need better use of memory.

‣ tries
‣ TSTs
‣ applications

Ternary search tries

TST. [Bentley-Sedgewick, 1997]

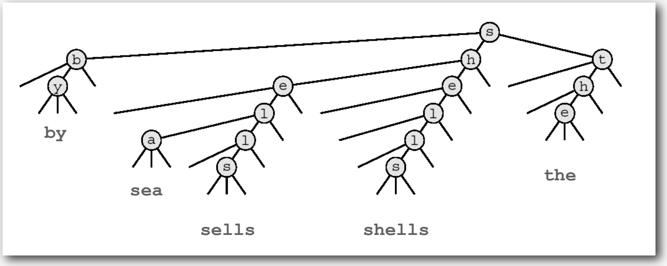
- Store characters in internal nodes, records in external nodes.
- Use the characters of the key to guide the search
- Each node has three children: smaller (left), equal (middle), larger (right).

Ternary search tries

TST. [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: smaller (left), equal (middle), larger (right).

Ex. `sells sea shells by the sea`



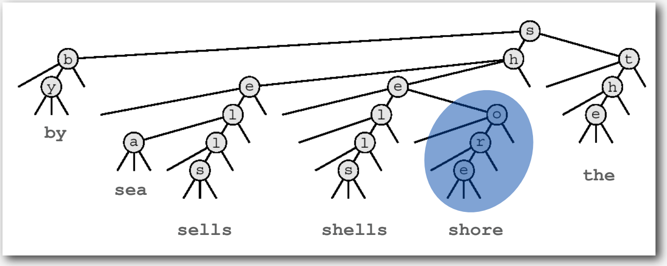
Observation. Only three null links in leaves!

Ternary search tries

TST. [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: smaller (left), equal (middle), larger (right).

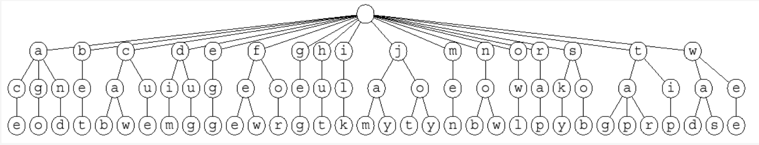
Ex. `sells sea shells by the sea shore`



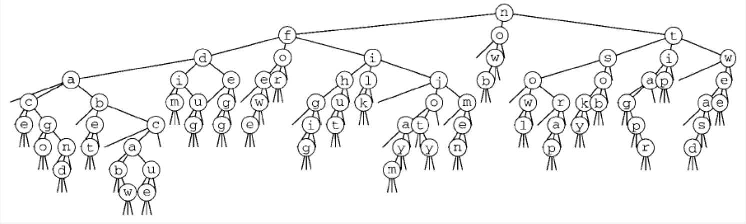
Observation. Only three null links in leaves!

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)

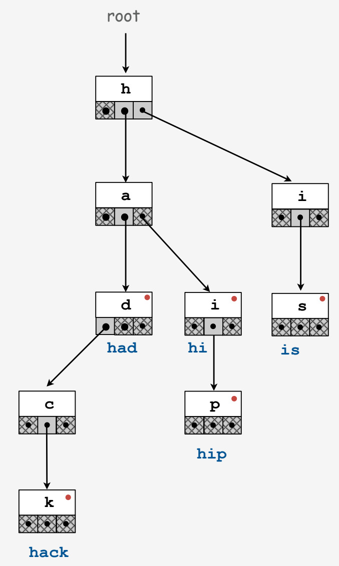
```
now
for
tip
ilk
dim
tag
jot
sob
nob
sky
hut
ace
bet
men
egg
few
jay
owl
joy
rap
gig
wee
was
cab
wad
caw
cue
fee
tap
ago
tar
jam
dug
and
```

TST representation

A TST 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.

```
class Node
{
   char c;
   Node left, mid, right;
   boolean end;
}
```



TST: Java implementation

```
public class TST
{
   private class Node
   {  /* see previous slide */  }

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

   private boolean contains(Node x, String s, int d)
   {
      if (x == null) return false;
      char c = s.charAt(d);
      if      (c < x.c)          return contains(x.left,  s, d);
      else if (c > x.c)          return contains(x.right, s, d);
      else if (d < s.length()-1) return contains(x.mid,   s, d+1);
      else                       return x.end;
   }

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

TST: Java implementation (cont)

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

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

String set implementation cost summary

	typical case			dedup	
implementation	search hit	insert	space	moby.txt	actors.txt
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	3 C	0.72	38.7

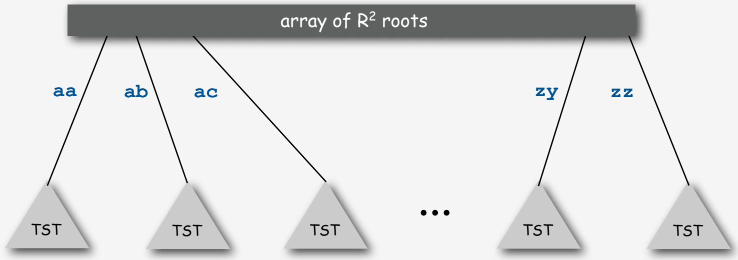
Remark. Can build balanced TSTs via rotations to achieve L + log N worst-case guarantees.

Bottom line. TST is as fast as hashing (for string keys), space efficient.

TST with R^2 branching at root

Hybrid of R-way and TST.

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



Note. Need special test for one- and two-letter words.

String set implementation cost summary

	typical case			dedup	
implementation	search hit	insert	space	`moby.txt`	`actors.txt`
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	0.72	38.7
TST with R^2	L	L	3C + R^2	0.51	32.7

Bonus. TST performance even better with nonuniform keys.

Ex. 5 times faster than hashing for library call numbers.

TST vs. 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.

- Works only for digital keys.
- Need to examine just enough key characters.
- Search miss may only involve a few characters.
- Can handle ordered-key APIs.

Bottom line. TSTs are faster than hashing (especially for search misses) and more flexible than red-black trees (stay tuned).

‣ tries
‣ TSTs
‣ applications

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 kth largest key.
Range search. Find all elements between k_1 and k_2.

compareTo()

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

charAt()

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

Easy to implement for R-way trie (below) or TST (see book).

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

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

Wildcard match

Use wildcard . to match any character in alphabet.

```
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.

Wildcard match: TST implementation

Search as usual if 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 d, String prefix)
{
   if (x == null) return;
   char c = s.charAt(i);
   if (c == '.' || c <  x.c) wildcard(x.left, s, d, prefix);
   if (c == '.' || c == x.c)
   {
      if (i < s.length() - 1)
         wildcard(x.mid, s, d+1, prefix + x.c);
      else if (x.end)
         StdOut.println(prefix + x.c);
   }
   if (c == '.' || c >  x.c) wildcard(x.right, s, d, prefix);
}
```

T9 texting

Goal. Type text messages on a phone keypad.

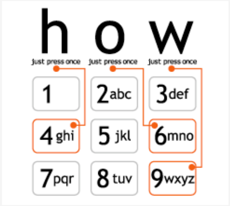
Multi-tap input. Enter a letter by repeatedly pressing a key until the desired letter appears.

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

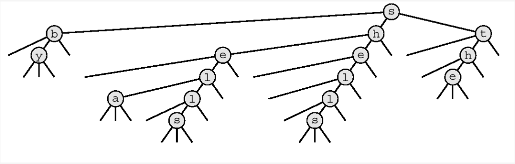
TST: collapsing 1-way branches

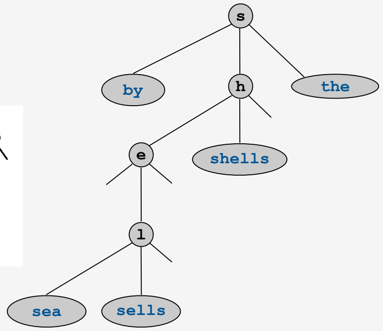
Collapsing 1-way branches at bottom.

- 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.





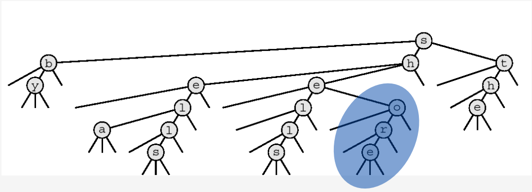
TST: collapsing 1-way branches

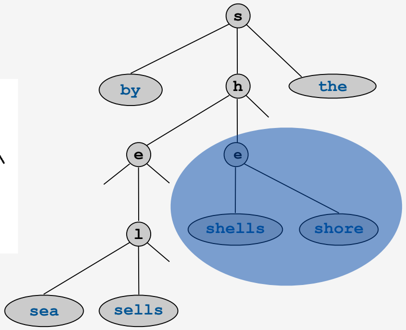
Collapsing 1-way branches at bottom.

- 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.





String set implementation cost summary

	typical case		
implementation	search hit	insert	space
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

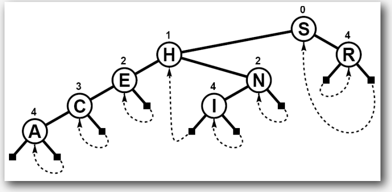
Challenge met.

- 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.



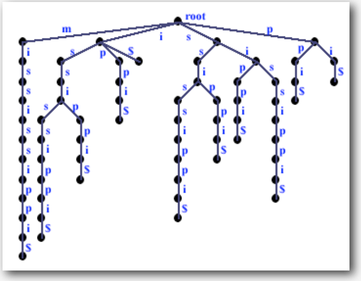
Applications.

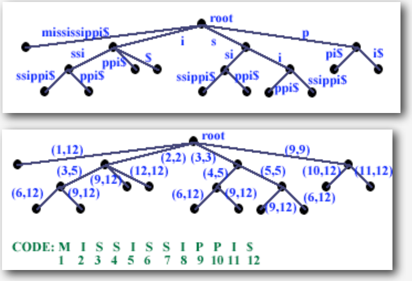
- Database search.
- P2P network search.
- IP routing tables: find longest prefix match.
- Compressed quad-tree for N-body simulation.
- Efficiently storing and querying XML documents.

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.

- Linear-time longest repeated substring.
- Computational biology databases (BLAST, FASTA).

Beyond the scope of COS 226.

Symbol tables summary

A success story in algorithm design and analysis.

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.
- 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 (!!!)