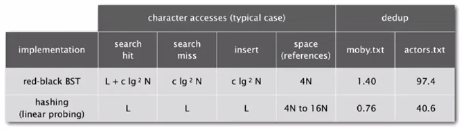
R-way Tries

We can beat algorithm performance of Red-Black BSTs and hash tables if we avoid examining entire keys

String symbol table API

Public class StringST<Value>

StringST() : create an empty symbol table  
void put(String key, Value val) : put key-value pair into the symbol table  
Value get(String key) : return value paired with given key  
void delete(String key) : delete key and corresponding value

String symbol table implementation cost summaries

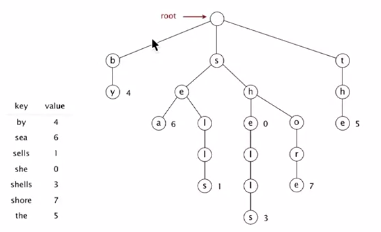
C:\Users\Zach\AppData\Local\Microsoft\Windows\INetCache\Content.Word\inputdeets.pngParameters:

* N = number of strings
* L = length of string
* R = radix

Tries

* Store characters in nodes (not keys)
* Each node has R children, one for each possible character (for now, we don’t draw null links)
* Store values in nodes corresponding to last characters in keys

Trie visualized (not showing any null links)



* Search in a trie:

Follow links corresponding to each character in the key

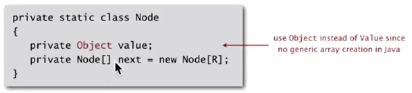
* Search hit: node where search ends has a non-null value
* Search miss: reach null link or node where search ends has a null value
* Insertion into a trie:

Follow links corresponding to each character in the key

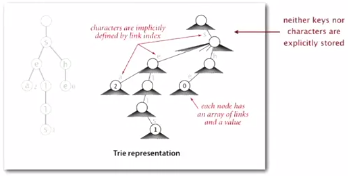
* Encounter a null link: create a new node
* Encounter the last character of the key: set value in that node

Trie representation

Node: a value, plus references to R nodes

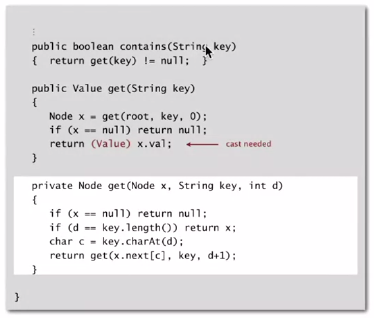
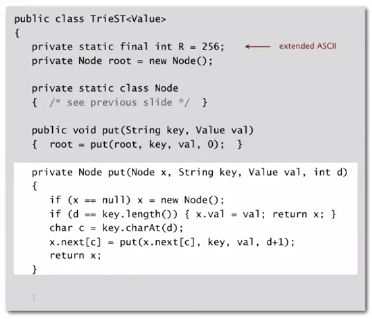


Node links are represented with indexed arrays with one entry for each possible characters



Characters implicitly defined by link indices

Trie implementation in Java



Performance:

Search hit: Need to examine all L characters for equality (length of string).

Search miss:

* Could have a mismatch on first character
* Typical case: only examine a few characters (sublinear)

Space: R null links at each leaf.  
*but sublinear space possible if many short strings share common prefixes*

Bottom line: Fast search hit and even faster search miss, but wastes a lot of space.

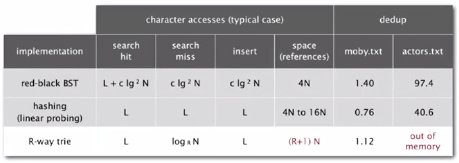
Application: a good data structure to perform efficient spell checking (26-way trie).

Deletion in an R-way trie

To delete a key-value pair:

* Find the node corresponding to key and set value to null
* If node has null value and all null links, remove that node (and recur)

Trie performance



R-way trie:

* Good for small R
* Too much memory for large R