CS61B Lecture #31

ed search structures (*DS(IJ)*, Chapter 9

om Numbers (*DS(IJ)*, Chapter 11)

lly Efficient Use of Keys: the Trie

much about cost of comparisons.

worst case is length of string.

hould throw extra factor of key length, L, into costs:

parisons really means $\Theta(ML)$ operations.

for key X, keep looking at same chars of X M times.

tter? Can we get search cost to be $O(L)$?

multi-way decision tree, with one decision per character

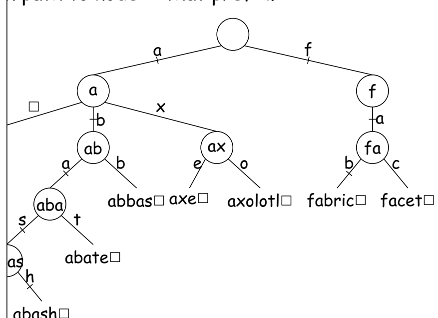
The Trie: Example

e, abash, abate, abbas, axolotl, axe, fabric, facet}

show paths followed for "abash" and "fabric"

l node corresponds to a possible prefix.

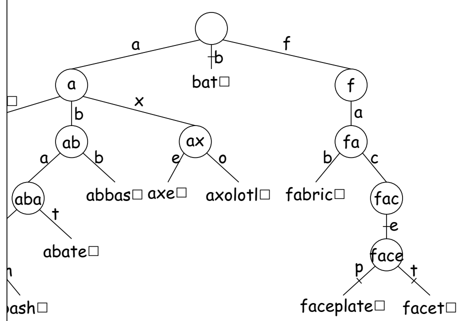
n path to node = that prefix.



Adding Item to a Trie

ding bat and faceplate.

icked.



A Side-Trip: Scrunching

bvious implementation for internal nodes is array in-
aracter.

erformance, L length of search key.

independent of N, number of keys. Is there a depen-

ays are *sparsely populated* by non-null values—waste of

arrays on top of each other!

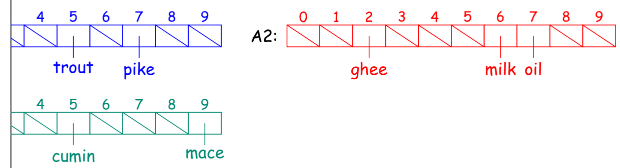
mpty) entries of one array to hold non-null elements of

arkers to tell which entries belong to which array.

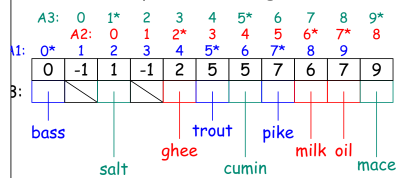
Scrunching Example

(unrelated to Tries on preceding slides)

rrays, each indexed 0..9



them, but keep track of original index of each item:



Practicum

ing idea is cute, but

od if we want to expand our trie.

plicated.

nore useful for representing large, sparse, fixed tables
rows and columns.

:, number of children in trie tends to drop drastically
's a few levels down from the root.

ce, might as well use linked lists to represent set of
en. . .

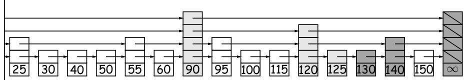
rays for the first few levels, which are likely to have
n.

robabilistic Balancing: Skip Lists

n be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
en go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

e nodes were chosen randomly so that there are about
nodes that are $> k$ high as there are that are k high.

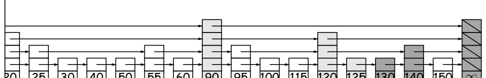
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

n be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
en go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

e nodes were chosen randomly so that there are about
nodes that are $> k$ high as there are that are k high.

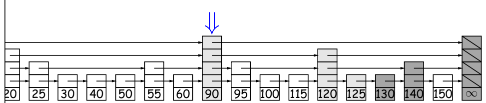
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

n be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
en go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

e nodes were chosen randomly so that there are about
nodes that are $> k$ high as there are that are k high.

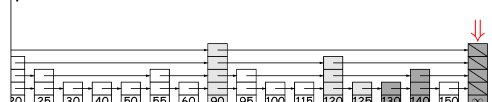
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

n be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
en go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

e nodes were chosen randomly so that there are about
nodes that are $> k$ high as there are that are k high.

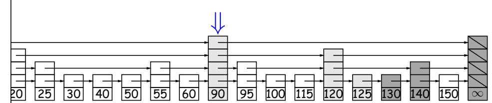
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

n be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
en go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

e nodes were chosen randomly so that there are about
nodes that are $> k$ high as there are that are k high.

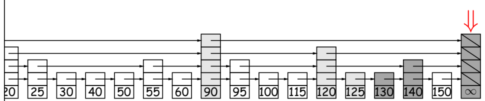
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

in be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
ien go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

ie nodes were chosen randomly so that there are about
nodes that are $>k$ high as there are that are k high.

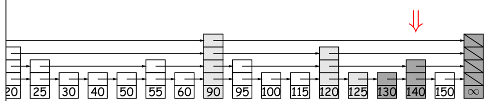
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

in be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
ien go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

ie nodes were chosen randomly so that there are about
nodes that are $>k$ high as there are that are k high.

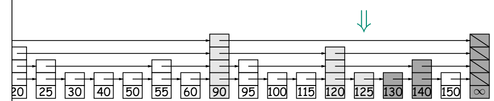
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

in be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
ien go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

ie nodes were chosen randomly so that there are about
nodes that are $>k$ high as there are that are k high.

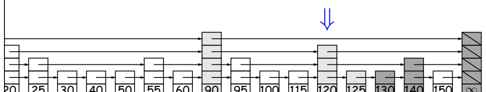
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

in be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
ien go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

ie nodes were chosen randomly so that there are about
nodes that are $>k$ high as there are that are k high.

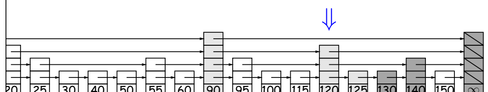
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

in be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
ien go down one layer and repeat.

, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

ie nodes were chosen randomly so that there are about
nodes that are $>k$ high as there are that are k high.

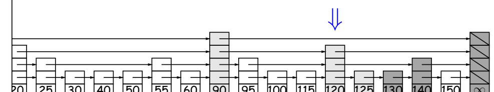
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

in be thought of as a kind of n-ary search tree in which
put the keys at "random" heights.

hought of as an ordered list in which one can skip large

ple:



tart at top layer on left, search until next step would
ien go down one layer and repeat.

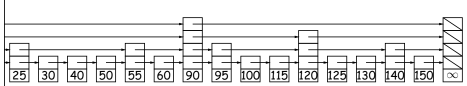
, we search for 125 and 127. Gray nodes are looked at;
nodes are overshoots.

ie nodes were chosen randomly so that there are about
nodes that are $>k$ high as there are that are k high.

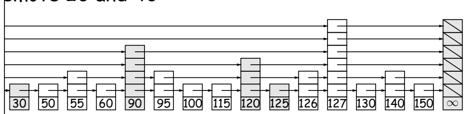
hes fast *with high probability.*

Example: Adding and deleting

m initial list:



r, we add 126 and 127 (choosing random heights for
emove 20 and 40:



s here have been modified.

Summary

arch trees allows us to realize $\Theta(\lg N)$ performance.

-black trees:

$N)$ performance for searches, insertions, deletions.

ood for external storage. Large nodes minimize # of
ations

) performance for searches, insertions, and deletions,
s length of key being processed.

to manage space efficiently.

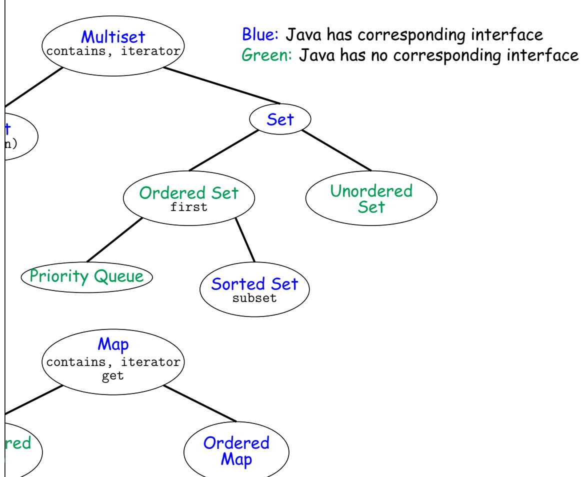
idea: scrunched arrays share space.

able $\Theta(\lg N)$ performace for searches, insertions, dele-

nplement.

l for *interesting ideas:* probabilistic balance, random-
structures.

mmary of Collection Abstractions



tructures that Implement Abstractions

linked lists, circular buffers

et

Queue: heaps

Set: binary search trees, red-black trees, B-trees,
arrays or linked lists

d Set: hash table

ap: hash table

: red-black trees, B-trees, sorted arrays or linked lists

Corresponding Classes in Java

ction)

ist, LinkedList, Stack, ArrayBlockingQueue,

et

Queue: PriorityQueue

Set (SortedSet): TreeSet

d Set: HashSet

ap: HashMap

(SortedMap): TreeMap