Public Service Announcement

erimental Social Science Laboratory (Xlab) invites
cipate in social science studies! Experiments con-
lab (located in Hearst Gym, Suite 2) are computer-
n-making studies such as tasks, surveys, and games.
asionally offer remote online and mobile studies that
leted anywhere. Participants earn $15/hour on av-
time they participate. For more information, visit
y.edu. To sign up, visit berkeley.sona-systems.com."

CS61B Lecture #31

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

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

lly Efficient Use of Keys: the Trie

lent 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

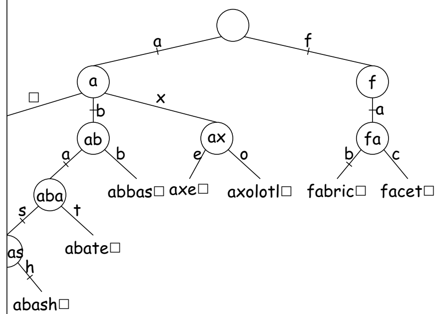
The Trie: Example

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

show paths followed for "abash" and "fabric"

l node corresponds to a possible prefix.

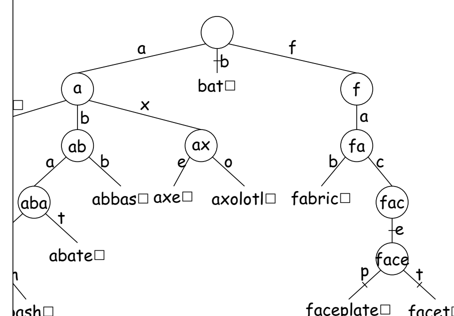
n path to node = that prefix.



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

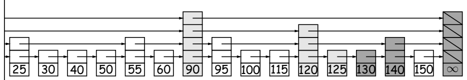
arkers to tell which entries belong to which array.

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
hen go down one layer and repeat.

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

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

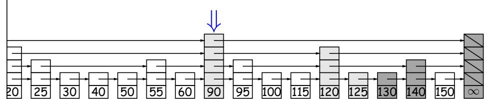
hes fast *with high probability.*

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
hen go down one layer and repeat.

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

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

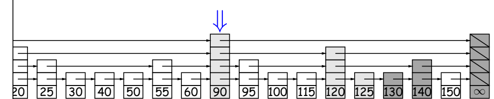
hes fast *with high probability.*

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
hen go down one layer and repeat.

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

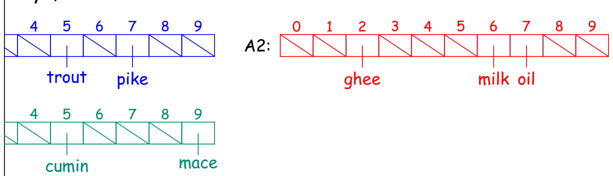
he nodes were chosen randomly so that there are about
nodes that are $> k$ high as there are that are k high.

hes fast *with high probability.*

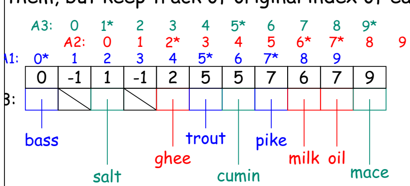
Scrunching Example

(unrelated to Tries on preceding slides)

rrays, each indexed 0..9



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

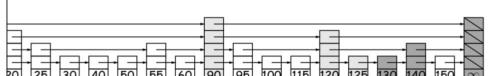


robabilistic Balancing: Skip Lists

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
hen go down one layer and repeat.

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

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

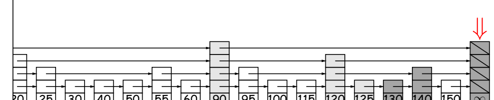
hes fast *with high probability.*

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
hen go down one layer and repeat.

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

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

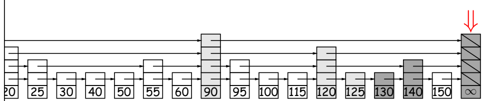
hes fast *with high probability.*

robabilistic Balancing: Skip Lists

an 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
hen go down one layer and repeat.

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

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

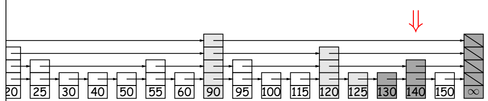
hes fast *with high probability*.

robabilistic Balancing: Skip Lists

an 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
hen go down one layer and repeat.

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

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

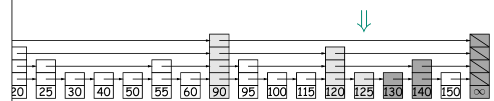
hes fast *with high probability*.

robabilistic Balancing: Skip Lists

an 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
hen go down one layer and repeat.

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

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

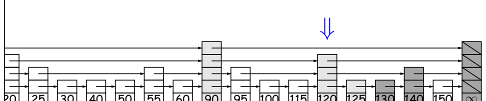
hes fast *with high probability*.

robabilistic Balancing: Skip Lists

an 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
hen go down one layer and repeat.

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

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

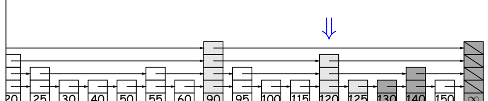
hes fast *with high probability*.

robabilistic Balancing: Skip Lists

an 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
hen go down one layer and repeat.

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

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

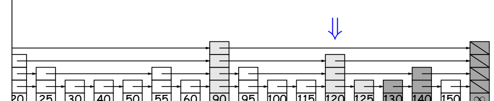
hes fast *with high probability*.

robabilistic Balancing: Skip Lists

an 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
hen go down one layer and repeat.

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

he 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*.

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
tions

) 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.

tructures that Implement Abstractions

linked lists, circular buffers

et

Queue: heaps

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

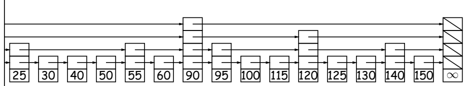
d Set: hash table

ap: hash table

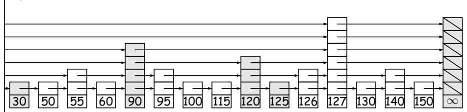
: red-black trees, B-trees, sorted arrays or linked lists

Example: Adding and deleting

m initial list:

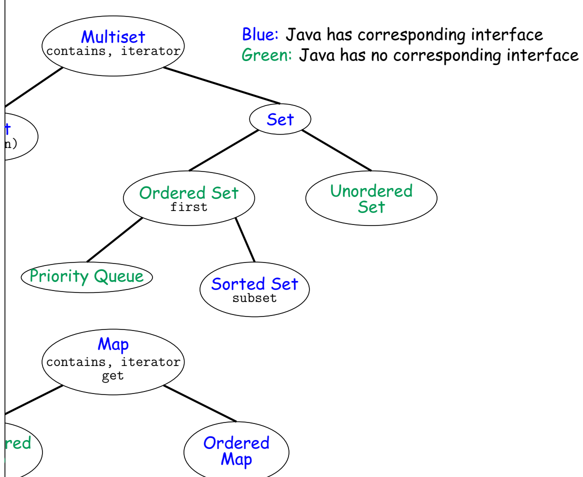


r, we add 126 and 127 (choosing random heights for
emove 20 and 40:



s here have been modified.

mmary of Collection Abstractions



Corresponding Classes in Java

ction)

ist, LinkedList, Stack, ArrayBlockingQueue,

et

Queue: PriorityQueue

Set (SortedSet): TreeSet

d Set: HashSet

ap: HashMap

(SortedMap): TreeMap