## 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

9:39 2018

CS61B: Lecture #31 2

9:39 2018

## CS61B Lecture #31

ed search structures (DS(IJ), Chapter 9

om Numbers (*DS(IJ), C*hapter 11)

### Adding Item to a Trie



#### The Trie: Example

abash, abate, abbas, axolotl, axe, fabric, facet
show paths followed for "abash" and "fabric"
I node corresponds to a possible prefix.
n path to node = that prefix.



Scrunching Example					
(unrelated to Tries on preceding slides)					
rrays, each indexed 09					
4 5 6 7 8 9 trout pike A2: 0 1 2 3 4 5 6 7 8 9 A2: ghee milk oil					
4 5 6 7 8 9 cumin mace					
them, but keep track of original index of each item:					
A3: 0 1* 2 3 4 5* 6 7 8 9* A2: 0 1 2* 3 4 5* 6 7* 8 9 1: 0* 1 2 3 4 5* 6 7* 8 9 0 -1 1 -1 2 5 5 7 6 7 9 B: $0 -1 1 -1 2 5 5 7 6 7 9$ bass trout pike milk oil mace					
9:39 2018 C5618: Lecture #31 6					

#### A Side-Trip: Scrunching

bvious implementation for internal nodes is array inaracter.

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!

9:39 2018

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

arkers to tell which entries belong to which array.

9:39 2018

CS61B: Lecture #31 1

9:39 2018

CS61B: Lecture #31 3

CS61B: Lecture #31 4

CS61B: Lecture #31 5

## 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 nen 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.

hore useful for representing large, sparse, fixed tables

number of children in trie tends to drop drastically

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

rrays for the first few levels, which are likely to have

hes fast with high probability.

Practicum

bd if we want to expand our trie.

s a few levels down from the root.

ing idea is cute, but

rows and columns.

plicated.

9:39 2018

CS61B: Lecture #31 8

CS61B: Lecture #31 7

## 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 nen 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.

9:39 2018

CS61B: Lecture #31 10

## 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:

9:39 2018



tart at top layer on left, search until next step would nen 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.

CS61B: Lecture #31 9

## 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



tart at top layer on left, search until next step would nen 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.

CS61B: Lecture #31 12

## 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:

9:39 2018

9:39 2018



tart at top layer on left, search until next step would nen 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.

en...

### 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



## 

tart at top layer on left, search until next step would nen 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.

9:39 2018

CS61B: Lecture #31 14

CS61B: Lecture #31 13

## 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 nen 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.

9:39 2018

CS61B: Lecture #31 16

## 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



tart at top layer on left, search until next step would nen 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.

CS61B: Lecture #31 18

## 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 nen 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.

9:39 2018

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

ple:

9:39 2018



in be thought of as a kind of n-ary search tree in which

robabilistic Balancing: Skip Lists

put the keys at "random" heights.

tart at top layer on left, search until next step would nen 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.

CS61B: Lecture #31 15

## 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:

9:39 2018



tart at top layer on left, search until next step would nen 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.

9:39 2018

Summary	tructures the	at Implement Abstractions		
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- uplement. I for interesting ideas: probabilistic balance, random- structures. 9:39 2018 CSGIB: Lecture #31 20	linked lists, circ et Queue: heaps Set: binary se arrays or linked d Set: hash table b: red-black tree 9:39 2018	cular buffers earch trees, red-black trees, B-trees, l lists le es, B-trees, sorted arrays or linked lists		
Example: Adding and deleting m initial list: r, we add 126 and 127 (choosing random heights for move 20 and 40:	mmary of Contains, iter	Blue: Java has corresponding interface Green: Java has no corresponding interface Unordered Set Sorted Set Sorted Set	C ist ist G S d S c J S c S c S c S c S c t s t S c t s t s t s t s t s t s t s t s t s t	Corresponding Classes in Java tion) t, LinkedList, Stack, ArrayBlockingQueue, t Queue: PriorityQueue et (SortedSet): TreeSet Set: HashSet p: HashMap (SortedMap): TreeMap
9:39 2018 C5618: Lecture #31 19	P:39 2018	Ordered Map CS618: Lecture #31 21	9:39	9 2018 C5618: Lecture #31 23