CSE 100
Ternary Tries and Skip Lists
Multi-way tries: Efficient finding of keys by their sequence

Build the trie to store the following numbers:

8
1234
59
123
8775
80

Is there a way to find whether all keys contained in a sequence of digits are present in the trie?
A. Yes
B. No
Properties of tries

Build the trie to store the following numbers:

If you stored the same N D-digit keys in a Binary Search Tree, what would be the worst case height of the tree?

A. N  B. $\lg(10^D)$  C. $\lg(N)$  D. $\lg(D)$  E. Other
Properties of tries

Build the trie to store the following numbers:

Consider storing the full $10^D$ keys. We know that on average the height of a BST will be $\lg(10^D)$. Which is smaller: $D$ or $\lg(10^D)$?

A. $D$  
B. $\lg(10^D)$  
C. They are the same
Properties of tries

Build the trie to store the following numbers:

8
1234
59
123
8775
80

So what is the main drawback of tries?
A. They are difficult to implement
B. They (usually) waste a lot of space
C. They are slow
D. There is no drawback of tries
Ternary search trees to the rescue!

- Tries combine binary search trees with tries.
- Each node contains the following:
  - A key **digit** for search comparison
  - Three pointers:
    - **left** and **right**: for when the digit being considered is less than and greater than (respectively) the digit stored in the node (the BST part)
    - **middle**: for when the digit being considered is equal to the digit stored in the node (the trie part)
  - An **end** bit to indicate we’ve completed a key stored in the tree.
List all the words (strings) you can find in this TST

Are the following in the tree? (A=yes, B=no)
• get
• if
• gif
• its
• gacar
• tsem
Draw the ternary tree for the following (in this order)

i
just
met
this
is
crazy
call
me
maybe

Does the structure of the tree depend on the order in which keys were inserted? A. Yes   B. No
Draw the ternary tree for the following (in this order)

i
just
met
this
is
crazy
call
me
maybe

Does the tree have a strong ordering property (i.e. keys in the left subtree are always less than trees in the right subtree?)
A. Yes  B. No
Algorithms for insert and find (in TSTs and MWTs)

- In your reading and/or in Paul Kube’s slides
Skip Lists: Motivation

- Which data structure is faster in the worst case for finding elements, assuming the elements are sorted?
  A. An array
  B. A linked list
  C. They can both be made equally fast

2 5 9 29 35 42 47 55 58 60 65
Skip Lists: Motivation

Which data structure is faster in the worst case for inserting elements, assuming the elements are sorted?

A. An array
B. A linked list
C. They can both be made equally fast
Toward Skip lists

- Adding forward pointers in a list can vastly reduce search time, from $O(N)$ to $O(\log N)$ in the worst case.
Toward Skip lists

- Costly to maintain a deterministic structure

Skip lists fix this problem by using randomness to randomly determine where pointers go
The max level of a node is equal to the number of forward pointers it has.

Which of the following depicts a node with max-level 2 in a skip list?
Node levels

- The following depicts a level 2 node

Shorthand

Picture of the book’s implementation
Find in a Skip List

Highlight the pointers that are followed in a find for the element 12. Annotate the order in which they are followed.
Find in a Skip List

Which of the following pointers are followed in a find for the element 35?

A. Red only
B. Red and blue only
C. Red, blue and purple only
D. Red, blue, purple and black
E. Some other combination of pointers
SkipList find, pseudocode

• To find key $k$:
  1. Let $x$ bet list header (root). Let $i$ be the highest non-null index pointer in $x$
  2. While pointer $x[i]$ is not null and points to a key smaller than $k$, let $x = x[i]$ (follow the pointer)
  3. If the pointer $x[i]$ points to a key equal to $k$ return true
  4. If the pointer $x[i]$ points to a key larger than $k$, decrement $i$ (drop down a level in $x$)
  5. If $i < 0$ return false. Else go to 2.

Assumes index pointers are 1 less than level
SkipList insert: slow motion
SkipList insert: slow motion

root

prev

curr

lvl ind: 3
Because prev[3] is null start at root again.
SkipList insert: slow motion
SkipList insert: slow motion

prev

root

lvl: 1

3 5 6 21 25 26
SkipList insert: slow motion
SkipList insert: slow motion

root

prev

curr

lvl: 0

3

5

6

21

25

26
SkipList insert: slow motion

prev  lvl: -1  newNode

curr

root
SkipList insert: slow motion

prev

root

curr

lvl: -1

newNode

3

5

6

21

25

26
SkipList insert: slow motion

root

prev

lvl: -1

newNode

prev

prev

prev

prev
SkipList insert: slow motion
Why Skip Lists?

- Why use a skip list? Discuss with your group.
Why Skip Lists?

- Why use a skip list? Discuss with your group.
  - Simpler to implement
  - Simple in-order traversal
  - Fast find (comparable to Balanced Binary Tree in the average case)
  - Min $O(1)$, delete-min $O(1)$
  - Amenable to concurrent modification (*changes are quite local*)