CSE 100
Ternary Tries and Skip Lists
Announcements

• Staff will be unavailable during Thanksgiving break.

• Please plan to get help today (in person or via Piazza)
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.

Given: Digit `d`

```
d < digit
  d == digit
  d > digit
```
List all the words (strings) you can find in this TST

Are the following in the tree? (A=yes, B=no)
- get  yes
- if   yes
- gif  no   g→e but i has no right could
- its  yes
- gacar no
- tsem no
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 \( O(\log n) \)
  B. A linked list \( O(n) \)
  C. They can both be made equally fast
Skip Lists: Motivation

- Which data structure is faster in the worst case for inserting elements, assuming the elements are sorted?
  A. An array \( O(N) \) because of shifts.
  B. A linked list \( O(N) \) because of find!
  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? B
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.

1. Start here, look ahead
2. Look ahead again
3. Look ahead again
4. Find 5 < 12, look ahead from 5
5. Found 12!

How would the search proceed if we were looking for the key '8'?
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

At this point we cannot be sure that 35 doesn’t exist because we are not at a leaf node.

Root
SkipList find, pseudocode

• To find key $k$:
  1. Let $x$ be 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

prev[i] should point to the node whose level i pointer needs to be modified to point to the new node.

curr[i] should point to the same node that the new node’s level i pointer has to point to after it is inserted.
SkipList insert: slow motion
SkipList insert: slow motion

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

root

prev

**lvl ind:** 2

curr

prev

curr

root
SkipList insert: slow motion

root

prev

lvl: 1

25

26

1

3

5

6

21

.
SkipList insert: slow motion

prev

root

curr

lvl: 0

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

prev

root

lvl: -1

curr

newNode

3 5 6 21 25 26
SkipList insert: slow motion

prev

root

lvl: -1

curr

newNode
SkipList insert: slow motion
SkipList insert: slow motion

root

3 5 6 21 22 25 26
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*)
Happy Thanksgiving

Make sure you take a break!
Play lots of Boggle 😊