Randomized Data Structures

Store a set $S$ of $n$ numbers to support:

- **Search**: given $q$, is $q \in S$? (membership)
  - given $q$, find predecessor/successor in $S$
    - (may not be in $S$)

- **Insert**
- **Delete**

**Known:**
- **Static**: $O(n)$ space, $O(n \log n)$ preprocessing
  - $O(\log n)$ query time
- **Dynamic**: $O(\log n)$ query & update time
  - $O(n)$ space

**by**
- AVL trees
- Red-black trees
- 2-3 trees, 2-3-4 trees,
- $BB(\alpha)$ trees
- Splay tree
- AA trees

**Random Method 1:** Skip Lists (Pugh ’90)

**Idea:** random sampling!

$S_0 = S$

$S_1$

$S_2$

$S_3$

**Expected space:** $O(n + \frac{n}{2^{k+1}} + \ldots) = O(n)$. 

1 4 5 6 9 11 13
\[ S = S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots \text{ till } S_k = \emptyset. \]

Let \( S_0 = S \)

\[ \forall x \in S_i, \text{ we put } x \in S_{i+1} \text{ w. prob. } \frac{1}{2} \text{ by indep coin flip.} \]

Store each \( S_i \) in a sorted linked list

Odd pts's between \( S_i \) & \( S_{i+1} \).

\[ \text{Obs} \quad \# \text{ levels } = O(\log n) \text{ w.h.p.} \]

**Pf:** Fix an elem \( x \).

level of \( x \) is geom distrib. w. prob. \( \frac{1}{2} \)

\[ \Rightarrow \text{ mean } 2 \]

\[ \Rightarrow P(\text{level of } x = i) = \left( \frac{1}{2} \right)^i \cdot \frac{1}{2} = 2^{-i} \]

\[ \Rightarrow P(\text{level of } x \geq \log n) \]

\[ \leq O\left( \frac{1}{n^c} \right) \]

\[ = O\left( \frac{1}{n^c} \right). \]

**pred-search** \((S_i, q)\):

\[ x = \text{pred-search} \left( S_{i+1}, q \right) \]

Do linear search in \( S_i \) from \( x \)

\[ \Rightarrow \text{ query time at level } i \text{ is geom distrib. w. prob. } \frac{1}{2} \]

\[ \Rightarrow E \left[ \text{query time per level} \right] = O(1) \]

\[ \Rightarrow E \left[ \text{query time} \right] = O(\log n). \]

**insert** \((S_i, x, p)\): \# given ptr \( p \) to pred of \( x \)

flip coin

if heads \{
flip coin
if heads {
    do linear search in $S_i$
    from $p$ to find pred $p'$ in $S_i$,
    \begin{align*}
    \text{insert} & (S_i, x, p') \\
    \end{align*}
}

\begin{align*}
\Rightarrow & \quad E[\text{insert time at level } i \mid \text{level}(x) \geq i] \\
& = O(1) \quad \text{geometric dist. w. prob } \frac{1}{2} \\
\Rightarrow & \quad \Pr[\text{level}(x) \geq i] = O\left(\frac{1}{2^i}\right). \\
\Rightarrow & \quad E[\text{insert time}] = \sum_i \frac{1}{2^i} = O(1) \\
& \text{if given pred} \\
& \text{if not, } O(\log n).
\end{align*}

Same for delete.

**Rand. Method 2: Trees** *(Seidel-Aragon '96)*

**Idea:**
- back to binary search tree
- pick root "randomly"

**How?**
- assign each elem
- a random priority value in $[0,1]$ (indep).

- for each subtree
- choose elem w. lowest priority value as its root.

**Keys:** $\{2, 4, 6, 7, 9, 11, 12\}$
**Priorities:** $(0.7, 0.5, 0.8, 0.1, 0.4, 0.2, 0.8)$

([[Insert figure here]]

**Note:** simultaneously, binary search tree (in the key values)

**Heap** (in the priority values)
Pred-search: same as in standard binary search tree
query time $O(\log n)$ w.h.p.
(equiv. to recur. depth of rand. quicksort)

Insert: pick rand. priority value
fix problems by rotations

$O(1)$ expected time if pred/succ given
(if omitted)

Similar for delete

no messy cases
no extra ptrs.

Question: can we do better than $O(\log n)$ query time?
no for comp.-based algns
but yes for membership queries for integers!
Assume all elms are in $\{0, 1, \ldots, U-1\}$. 
Easy Method 0:

- Use bit vector of size \( U \)
- Query time \( O(1) \)
- Insert \( O(1) \)
- Delete \( O(1) \)

but space is \( O(U) \).

Next: hashing...