Randomized Search Structures

Today we will discuss two randomized search structures:
- Treaps
- Skip lists

We shall see that these structures are both very efficient and very simple.

Running times are measured in expectation over all random choices made.

Note that expected time does not depend on the distribution of keys or the order of operations.
Recall that if $n$ keys are inserted into a (standard) binary search tree, the expected height is $O(\log n)$

**Treap** - A binary tree that behaves “as if” keys were inserted in random order

**Intuition:**
- Label each item with its *insertion time*
- These timestamps are ordered like a *heap*

---

![Binary Search Tree Diagram](image.png)

**Insertion order:** k, e, b, o, f, h, w, m, c, a, s

**Binary search tree (a)**

**With timestamps (b)**

---

**Timestamp**

**Key**
A treap is a binary search tree, where every node $p$ stores a priority $p.priority$

- Priority values are chosen randomly when the key is inserted, and do not change
- The tree is structured as if keys were inserted in priority order

**Theorem:** Expected height is $O(\log n)$

**Ordering:**
- Keys - inorder
- Priorities - heap order
Treap Insertion

- Apply the standard insertion process - create node where we fall out of tree
- Assign a random priority value to the new node
- Apply rotations up the tree until it is in proper heap order
- **Note:** Inorder is maintained throughout
Treap Deletion

- Find the node to be deleted
- Set its priority value to $\infty$
- Rotate it down to the leaf level and unlink
Skip List

- A “better” linked list
- **Intuition:** “Ideal” skip list
  - Store keys in a sorted link list (level 0)
  - Promote every other key from level \( i - 1 \) to level \( i \)
  - Number of levels is \( O(\log n) \)
(Randomized) Skip List
- Each node at level $i$ tosses a coin
- If the coin comes up heads (probability = $\frac{1}{2}$) extend this node to level $i + 1$
- Expected number of levels is $O(\log n)$
Skip List: Find(x)

- Start at the **topmost level** \((i = \text{maxLevel})\)
- Walk through level \(i\) until finding the **rightmost** node \(p\) such that \(p.key \leq x\)
  - if \(p.key == x\), found it!
  - else if \(i > 0\), drop to next lower level \((i = i-1)\)
  - else not found
Skip List - Randomized Analysis

- Let $E(i)$ be the expected number of nodes visited at level $i$ and lower.
- **Backwards analysis**: Walk backwards along the search path.
- Suppose we are currently at level $i - 1$.
- If current node contributes to next higher level (with prob $\frac{1}{2}$) search goes up a level ($i$), else we stay at current level ($i - 1$). Thus:
  \[
  E(i) = 1 + \frac{1}{2} E(i-1) + \frac{1}{2} E(i)
  \]
- **Conclusion**: $E(i) = 2 + E(i - 1) = 2 \cdot i$ (i.e., two nodes per level).
- **Theorem**: Expected search time is $O(\log n)$.
Skip List Insertion

\[ \text{insert}(x, v) \]

- Walk through structure, as if doing find(x)
- Keep track of the last node visited before dropping down a level
- When we find where to insert x at level 0, apply coin flipping to determine level

\[
k = 0 \\
\text{while (} k \leq \text{maxLevel} \&\& \text{Math.random()} \% 2 == 0 \text{) } k++; \\
\text{generate node of level } k, \text{ with key } x \text{ and value } v
\]
- Link this node into levels 0 through \( k \)
Skip List - Insertion

![Skip List Diagram]

1. The process starts with a skip list containing elements 2, 10, 11, 13, 19, 22, and 25.
2. An element 24 is inserted into the list.
3. The element 24 is inserted into the list at the correct level, maintaining the skip list property.
4. After insertion, the skip list contains elements 2, 10, 11, 13, 19, 22, 24, and 25.
Summary

- Randomized search structures
- Treaps
- Skip lists