1 Content

The major content of the lecture:

- Last classes: AVL trees, splay trees
  - detailed implementation
  - $O(\log n)$ guaranteed all ops
  - simple randomized
  - $O(\log n)$ w.h.p. expected
- Treaps
- Skip Lists

2 Heap and Treap

Tree: all left descendants of a node are smaller than that node and all right descendants are larger.

**Definition 1.** Heap: A specialized tree-based data structure, with each node is smaller than all its descendants.

2.1 Treap

To maintain a dynamic set of ordered keys and allow binary searches among the keys, introduce data structure treap.

**Definition 2.** Treap: A tree in which each key is given a randomly chosen numeric priority. The structure of the tree should be determined by the priority in heap-ordered.

**Example 3.** A treap example is in Figure 2.1 which has letters as keys and random number as priorities. Keys are ordered as BST and priorities are ordered as heap.
Insert(key):
1. Choose a random “priority” that is paired with key, i.e. (key, priority).
2. Insert keys like normal BST
3. Rotate up until the priority obeys heap priority

**Example 4.** Insert (C,9) in above example.

**Proposition 5.** Treap: Same as if we initially ordered elements by priority and inserted into BST.

The algorithm to delete keys can be summarized as:
Delete(key)
1. Find key
2. Rotate until leaf
3. Once leaf delete
2.2 Analysis

Assume

\[ A_i^k = \begin{cases} 1 & \text{if } x_i \text{ is ancestor of } x_k \\ 0 & \text{else} \end{cases}, \]

then

\[ \text{depth}(x_k) = \sum_{i=1}^{n} A_i^k. \]

**Theorem 6.** \( x_i \) is ancestor of \( x_k \) iff \( x_i \) has the smallest priority in \( \{x_i, \cdots, x_k\} \) or \( \{x_k, \cdots, x_i\} \).

**Proposition 7.** The depth of treap is \( O(\ln n) \) w.h.p.

Proof:

\[ P(A_i^k = 1) = \frac{1}{k - i + 1} \text{ or } \frac{1}{i - k + 1} \]

\[ E[\text{depth}(x_k)] = \sum_{i=1}^{n} E[A_i^k] = \sum_{i=1}^{k-1} \frac{1}{k - i + 1} + \sum_{i=k+1}^{n} \frac{1}{i - k + 1} \]

\[ = \sum_{j=2}^{k} \frac{1}{j} + \sum_{j=2}^{n} \frac{1}{j} \]

\[ E[\text{depth}(x_k)] = \sum_{j=1}^{k-1} \frac{1}{j} + \sum_{j=1}^{n-1} \frac{1}{j} \]

\[ = H(k - 1) + H(n - k - 1) \]

\[ \leq \ln (k - 1) + \ln (n - k - 1) + 2 \leq 2 \ln n + 2 \]

**Theorem 8.** Operations based on treap take \( O(\ln n) \) w.h.p.

3 Skip List

**Proposition 9.** Link list takes \( O(n) \) to realize find, insert, delete etc.

**Definition 10.** Skip list is a data structure for storing a sorted list of items using a hierarchy of linked lists than connect increasingly sparse subsequences of the items.
Example 11. A practical example is the Number 1 subway line in NYC. A local train will go through:

14 → 23 → 34 → 42 → 50 → 59 → 66 → 72 → 79

however, the express only goes through:

14 → 34 → 42 → 72

which is a skip list of the original link list.

Assume there are \( L_1 \) elements in list 1 (number of local stops), there are \( L_2 \) elements in list 2 (number of express stops). If the express stations are relatively evenly scattered, then the stations between express stops are \( \frac{L_1}{L_2} \). Thus the maximum number of stops to reach some specific station is

\[
L_2 + \frac{L_1}{L_2}
\]

to minimize this number, assume \( L_1 = n \), then

\[
\min \left( L_2 + \frac{n}{L_2} \right) \Rightarrow L_2 = \sqrt{n} \Rightarrow L_2 + \frac{L_1}{L_2} = 2\sqrt{n}
\]

The steps for implementing insert are:

1. Find where goes
2. Insert into bottom list

Algorithm:

\[
\text{whilt (coin flip = heads)}
\text{ insert the element into next list }
\]

Then, we have

\[
P(\text{X is at level } \geq l) = \frac{1}{2^l}
\]

\[
P(\text{Any element level } \geq l) \leq \frac{n}{2^l}
\]
assume

\[ l = c \log n \]

thus

\[ P(\text{Any element level} \geq l) \leq \frac{n}{2^c \log n} = \frac{1}{n^{c-1}} = \frac{1}{n^\alpha}. \]

**Theorem 12.** There are \( O(\log n) \) levels in skip list w.h.p.