Skip Lists

Skip List

• Question:
  – Can we create a structure that adds the best properties of Array and Linked list Data Structure?
  • Query: O(log n) in sorted Arrays
  • Insert/Removal: O(1) in Linked List

What is a Skip List

• A skip list for a set S of distinct (key, element) items is a series of lists S_0, S_1, …, S_h such that
  – Each list S_i contains the special keys +∞ and −∞
  – List S_0 contains the keys of S in nondecreasing order
  – Each list is a subsequence of the previous one, i.e.,
    \[ S_0 \supseteq S_1 \supseteq \cdots \supseteq S_h \]
  – List S_h contains only the two special keys
• We show how to use a skip list to implement the dictionary ADT

S_0
S_1
S_2
S_3
S_4
We search for a key $x$ in a skip list as follows:
- We start at the first position of the top list
- At the current position $p$, we compare $x$ with $y \leftarrow \text{key(after(p))}$
  - If $x = y$, we return $\text{element(after(p))}$
  - If $x > y$, we “scan forward”
  - If $x < y$, we “drop down”
- If we try to drop down past the bottom list, we return $\text{NO_SUCH_KEY}$

Example: search for 78

To insert an item $(x, o)$ into a skip list, we use a randomized algorithm:
- We repeatedly toss a coin until we get tails, and we denote with $i$ the number of times the coin came up heads
- If $i \geq h$, we add to the skip list new lists $S_{h+1}, \ldots, S_{i+1}$, each containing only the two special keys
- We search for $x$ in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with largest key less than $x$ in each list $S_0, S_1, \ldots, S_i$
- For $j \leftarrow 0, \ldots, i$, we insert item $(x, o)$ into list $S_j$ after position $p_j$

Example: insert key 15, with $i = 2$

To remove an item with key $x$ from a skip list, we proceed as follows:
- We search for $x$ in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key $x$, where position $p_j$ is in list $S_j$
- We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, \ldots, S_i$
- We remove all but one list containing only the two special keys

Example: remove key 34
Implementation

- We can implement a skip list with quad-nodes
- A quad-node stores:
  - item
  - link to the node before
  - link to the node after
  - link to the node below
  - link to the node above
- Also, we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them

Quad-node

Space Usage

- The space used by a skip list depends on the random bits used by each invocation of the insertion algorithm
- We use the following two basic probabilistic facts:
  Fact 1: The probability of getting $i$ consecutive heads when flipping a coin is $1/2^i$
  Fact 2: If each of $n$ items is present in a set with probability $p$, the expected size of the set is $np$
- Consider a skip list with $n$ items
  - By Fact 1, we insert an item in list $S_i$ with probability $1/2^i$
  - By Fact 2, the expected size of list $S_i$ is $n/2^i$
- The expected number of nodes used by the skip list is
  \[
  \sum_{i=2}^{\infty} \frac{n}{2^i} = \sum_{i=2}^{\infty} \frac{1}{2^{i-1}} < 2n
  \]
- Thus, the expected space usage of a skip list with $n$ items is $O(n)$

Height

- The running time of the search an insertion algorithm is affected by the height $h$ of the skip list
- We show that with high probability, a skip list with $n$ items has height $O(\log n)$
- We use the following additional probabilistic fact:
  Fact 3: If each of $n$ events has probability $p$, the probability that at least one event occurs is at most $np$
- Consider a skip list with $n$ items
  - By Fact 1, we insert an item in list $S_i$ with probability $1/2^i$
  - By Fact 3, the probability that list $S_i$ has at least one item is at most $n/2^i$
  - By picking $i = 3\log n$, we have the probability that $S_{3\log n}$ has at least one item is at most $n/2^{3\log n} = n/n^3 = 1/n^2$
- Thus a skip list with $n$ items has height at most $3\log n$ with probability at least $1 - 1/n^2$
Search and Update Times

- The search time in a skip list is proportional to:
  - the number of drop-down steps, plus
  - the number of scan-forward steps.
- The drop-down steps are bounded by the height of the skip list and thus are $O(\log n)$ with high probability.
- To analyze the scan-forward steps, we use yet another probabilistic fact:
  Fact 4: The expected number of coin tosses required in order to get tails is 2.
- When we scan forward in a list, the destination key does not belong to a higher list.
  - A scan-forward step is associated with a former coin toss that gave tails.
- By Fact 4, in each list the expected number of scan-forward steps is 2.
  - Thus, the expected number of scan-forward steps is $O(\log n)$.
- We conclude that a search in a skip list takes $O(\log n)$ expected time.
- The analysis of insertion and deletion gives similar results.

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with $n$ items:
  - The expected space used is $O(n)$.
  - The expected search, insertion, and deletion time is $O(\log n)$.
- Using a more complex probabilistic analysis, one can show that these performance bounds also hold with high probability.
- Skip lists are fast and simple to implement in practice.

Sorting Lower Bound
Comparison-Based Sorting

Many sorting algorithms are comparison based.
- They sort by making comparisons between pairs of objects
- Examples: bubble-sort, selection-sort, insertion-sort, heap-sort, merge-sort, quick-sort, ...
- Let us therefore derive a lower bound on the running time of any algorithm that uses comparisons to sort \( n \) elements, \( x_1, x_2, \ldots, x_n \).

Counting Comparisons
- Let us just count comparisons then.
- Each possible run of the algorithm corresponds to a root-to-leaf path in a decision tree
- Example: \( x_a, x_b, x_c \)

Decision Tree Height
- The height of this decision tree is a lower bound on the running time
- Every possible input permutation must lead to a separate leaf output.
  - If not, some input \( \ldots 4 \ldots 5 \ldots \) would have same output ordering as \( \ldots 5 \ldots 4 \ldots \), which would be wrong.
- Since there are \( n! = 1 \times 2 \times \ldots \times n \) leaves, the height is at least \( \log(n!) \).
The Lower Bound

- Any comparison-based sorting algorithms takes at least \( \log (n!) \) time.
- Therefore, any such algorithm takes time at least

\[
\log (n!) \geq \log \left( \frac{n}{2} \right)^{\frac{n}{2}} = (n/2) \log (n/2).
\]

- That is, any comparison-based sorting algorithm must run in \( \Omega(n \log n) \) time.