CH 9.4 : SKIP LISTS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN C++, GOODRICH, TAMASSIA AND MOUNT (WILEY 2004) AND SLIDES FROM NANCY M. AMATO AND JORY DENNY
A randomized algorithm controls its execution through random selection (e.g., coin tosses)

It contains statements like:

```plaintext
b ← randomBit()
if b = 0
    do something…
else // b = 1
    do something else…
```

Its running time depends on the outcomes of the “coin tosses”

Through probabilistic analysis we can derive the expected running time of a randomized algorithm

We make the following assumptions in the analysis:

- the coins are unbiased
- the coin tosses are independent

The worst-case running time of a randomized algorithm is often large but has very low probability (e.g., it occurs when all the coin tosses give “heads”)

We use a randomized algorithm to insert items into a skip list to insert in expected $O(\log n)$–time

When randomization is used in data structures they are referred to as probabilistic data structures
WHAT IS A SKIP LIST?

- A skip list for a set \( S \) of distinct (key, element) items is a series of lists \( S_0, S_1, \ldots, S_h \)
  - Each list \( S_i \) contains the special keys \( +\infty \) and \( -\infty \)
  - List \( S_0 \) contains the keys of \( S \) in non-decreasing 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
- Skip lists are one way to implement the Ordered Map ADT
  - [Java applet]
We can implement a skip list with quad-nodes

A quad-node stores:
- (Key, Value)
- links to the nodes before, after, below, and above

Also, we define special keys $+\infty$ and $-\infty$, and we modify the key comparator to handle them
We search for a key $k$ in a skip list as follows:

- We start at the first position of the top list
- At the current position $p$, we compare $k$ with $y \leftarrow p.next().key()$
  - $x = y$: we return $p.next().value()$
  - $x > y$: we scan forward
  - $x < y$: we drop down
- If we try to drop down past the bottom list, we return \texttt{NO\_SUCH\_KEY}

Example: search for 78
We search for a key $k$ in a skip list as follows:

- We start at the first position of the top list.
- At the current position $p$, we compare $k$ with $y ← p$.next().key()
  - $x = y$: we return $p$.next().value()
  - $x > y$: we scan forward
  - $x < y$: we drop down
- If we try to drop down past the bottom list, we return \textit{NO\_SUCH\_KEY}

Ex 1: search for 64: list the $(S_i, \text{node})$ pairs visited and the return value

Ex 2: search for 27: list the $(S_i, \text{node})$ pairs visited and the return value
To insert an item \((k, v)\) 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 \(k\) in the skip list and find the positions \(p_0, p_1, \ldots, p_i\) of the items with largest key less than \(k\) in each list \(S_0, S_1, \ldots, S_i\).
- For \(i \leftarrow 0, \ldots, i\), we insert item \((k, v)\) into list \(S_i\) after position \(p_i\).

Example: insert key 15, with \(i = 2\)
To remove an item with key $k$ from a skip list, we proceed as follows:

- We search for $k$ in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key $k$, where position $p_i$ is in list $S_i$.
- 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
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 $\frac{1}{2^i}$.
- 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 $\frac{1}{2^i}$.
- By Fact 2, the expected size of list $S_i$ is $\frac{n}{2^i}$.

The expected number of nodes used by the skip list is:

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

Thus the expected space is $O(2n)$. 
The running time of find\((k)\), put\((k,v)\), and erase\((k)\) operations are 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 \(\frac{1}{2^i}\).
- By Fact 3, the probability that list \(S_i\) has at least one item is at most \(\frac{n}{2^i}\).
- By picking \(i = 3 \log n\), we have that the probability that \(S_{3 \log n}\) has at least one item is at most \(\frac{n}{n^3} = \frac{1}{n^2}\).
- Thus a skip list with \(n\) items has height at most \(3 \log n\) with probability at least \(1 - \frac{1}{n^2}\).
SEARCH AND UPDATE TIMES

- The search time in a skip list is proportional to
  - the number of drop-down steps
  - the number of scan-forward steps
- The drop-down steps are bounded by the height of
  the skip list and thus are $O(\log n)$ expected time
- 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
You are working for ObscureDictionaries.com a new online start-up which specializes in sci-fi languages. The CEO wants your team to describe a data structure which will efficiently allow for searching, inserting, and deleting new entries. You believe a skip list is a good idea, but need to convince the CEO. Perform the following:

- Illustrate insertion of “X-wing” into this skip list. Randomly generated (1, 1, 1, 0).
- Illustrate deletion of an incorrect entry “Enterprise”
- Argue the complexity of deleting from a skip list
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.