An advanced approach towards linked lists…

SKIP LISTS

Presented By. Soumadip Biswas
Roll: 10IT60R12

Under the guidance of:
Prof Arobindo Gupta

OUTLINE

- INTODUCTION
- REVIEW
- NEW APPROACH
- COMPLEXITY ANALYSIS
- VARIATION
- APPLICATIONS
- SUMMARY

INTRODUCTION

REVIEW – ARRAY & LINKED LISTS

- Arrays – Ordered Arrays
  - Search time is O(log n)
  - Insertion and Deletion takes O(n) time
  - Fixed size
- Linked lists
  - Sorted and doubly linked-list can do min, max, predecessor etc operation in O(1) time
  - But insert and delete can take up to O(n) time
  - Goal is to make them O(log n)

REVIEW — BALANCED TREES

- AVL tree
  - Search, Insert, Delete operation take O(log n) time
  - Use *rotation* to restructure tree
- Red-black trees:
  - Binary search trees augmented with node color
  - Operations designed to guarantee that the height $h = O(\log n)$
- These guarantee $h = O(\log n)$
- Operations on red-black trees
  - Only tricky operations: insert, delete
  - The operations are complex to implement

NEW APPROACH — SKIP LISTS

- A Skip List is a *Data Structure* based on parallel linked lists "*discovered*" by *William Pugh* in 1990.

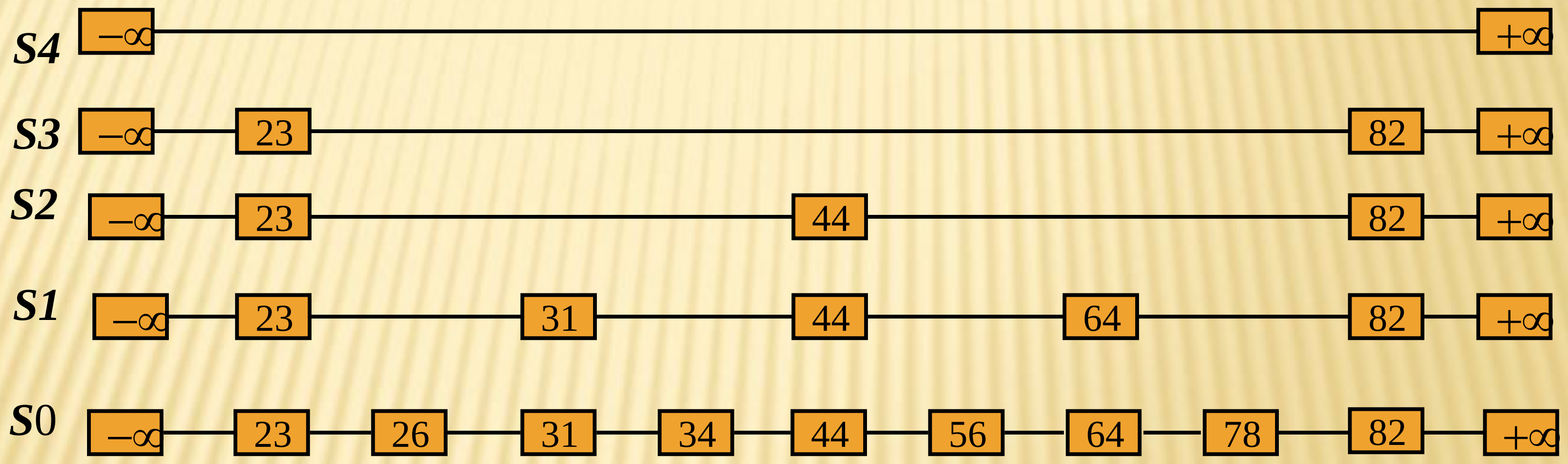
- It's major advantage is that it provides expected *O(log n)* time for all operations.

- At a high level, a skip list is just a sorted, singly linked list with some "*shortcuts*" (additional pointers that allow to travel faster.

THE IDEA …

- The idea behind skip list is can we reduce the time further?
- The skip list in the beginning can be identified as a better and easy implementation of BST
- An ideal skip list is nothing but to maintain multiple linked lists S0,S1,S2,…Si… which points in the following manner

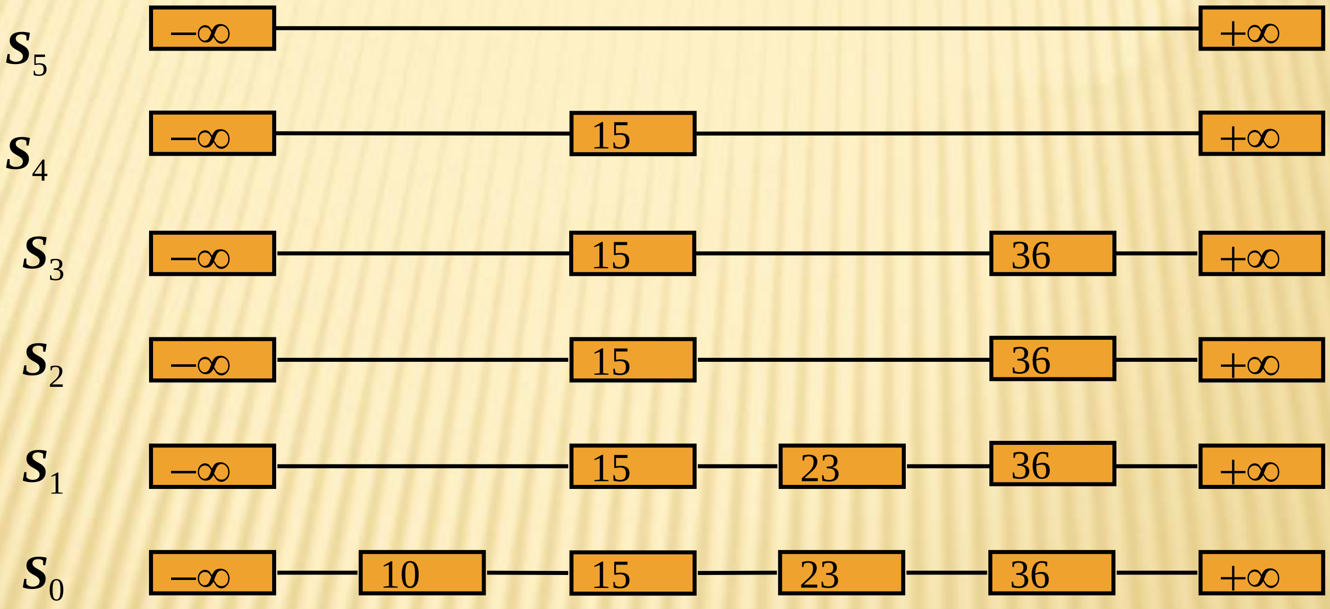
DEMONSTRATION



THERE COMES THE NEXT CONCEPT

- Instead of making a replica of BST comes the concept of incorporating probability
- As if a node will have no of forward references is being determined by some probability
  - Here coin toss probability is used
- This can be something like…

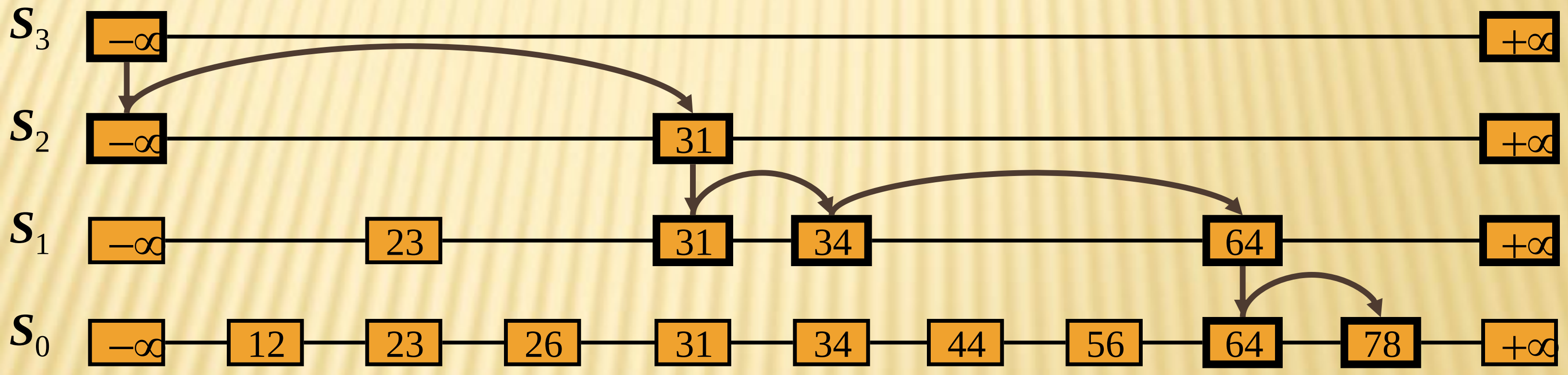
DEMONSTRATION



OPERATIONS ON SKIP LIST

SEARCH

Example: search for 78

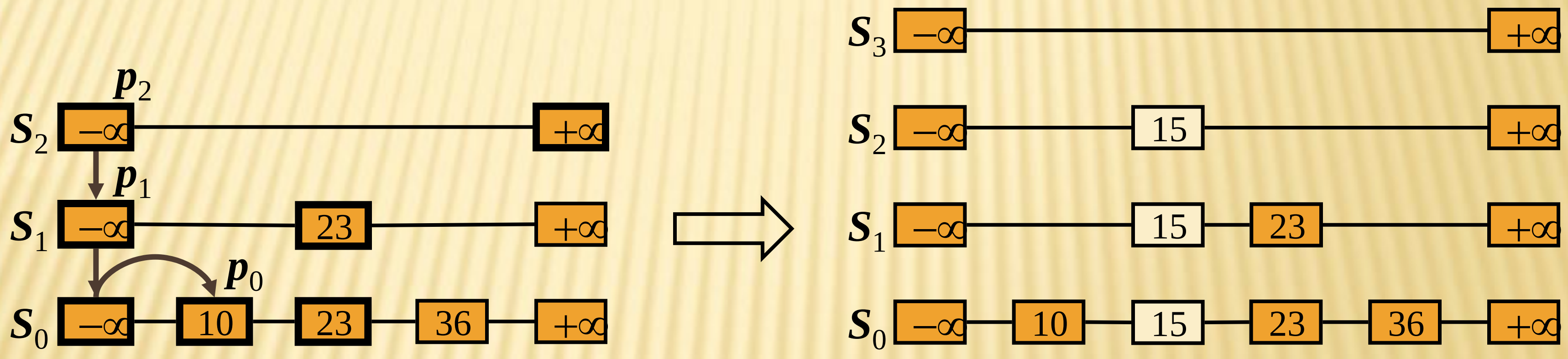


SEARCH

- 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 $y \leftarrow key(next(p))$
    - $x = y$: we return *element*(*next*(*p*))
    - $x > y$: we “scan forward”
    - $x < y$: we “drop down”
  - If we try to drop down past the bottom list, we return *null*

INSERTION

Example: insert key 15, with $\boldsymbol{i} = 2$

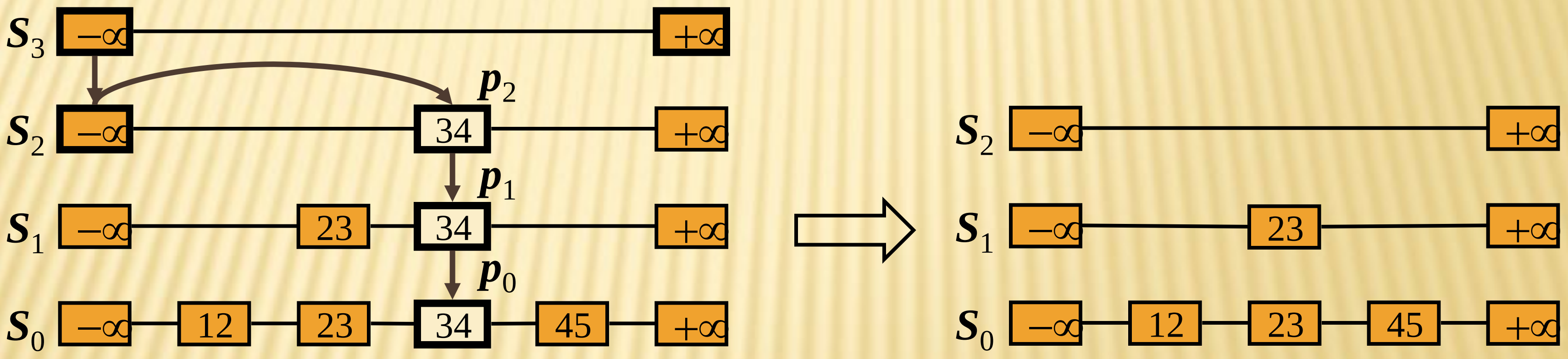


INSERTION

- To insert an entry (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

DELETION

Example: remove key 34



DELETION

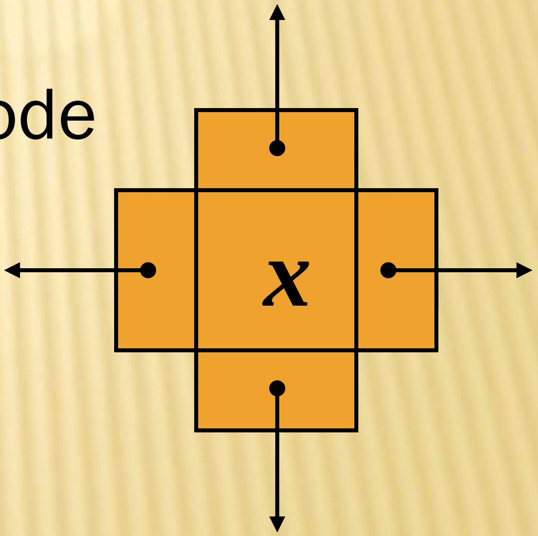
- To remove an entry with key $\boldsymbol{x}$ from a skip list, we proceed as follows:
  - We search for $\boldsymbol{x}$ in the skip list and find the positions $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_i$ of the items with key $\boldsymbol{x}$, where position $\boldsymbol{p}_j$ is in list $\boldsymbol{S}_j$
  - We remove positions $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_i$ from the lists $\boldsymbol{S}_0, \boldsymbol{S}_1, \ldots, \boldsymbol{S}_i$
  - We remove all but one list containing only the two special keys

ANALYSIS

IMPLEMENTATION

- We can implement a skip list with quad-nodes
- A quad-node stores:
  - entry
  - link to the node previous
  - link to the node next
  - link to the node below
  - link to the node above

quad-node



SPACE COMPLEXITY

- 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 entries is present in a set with probability p, the expected size of the set is np

SPACE COMPLEXITY – CONTD.

- Consider a skip list with n entries
  - By Fact 1, we insert an entry 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

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

- Thus, the expected space usage of a skip list with n items is $O(n)$

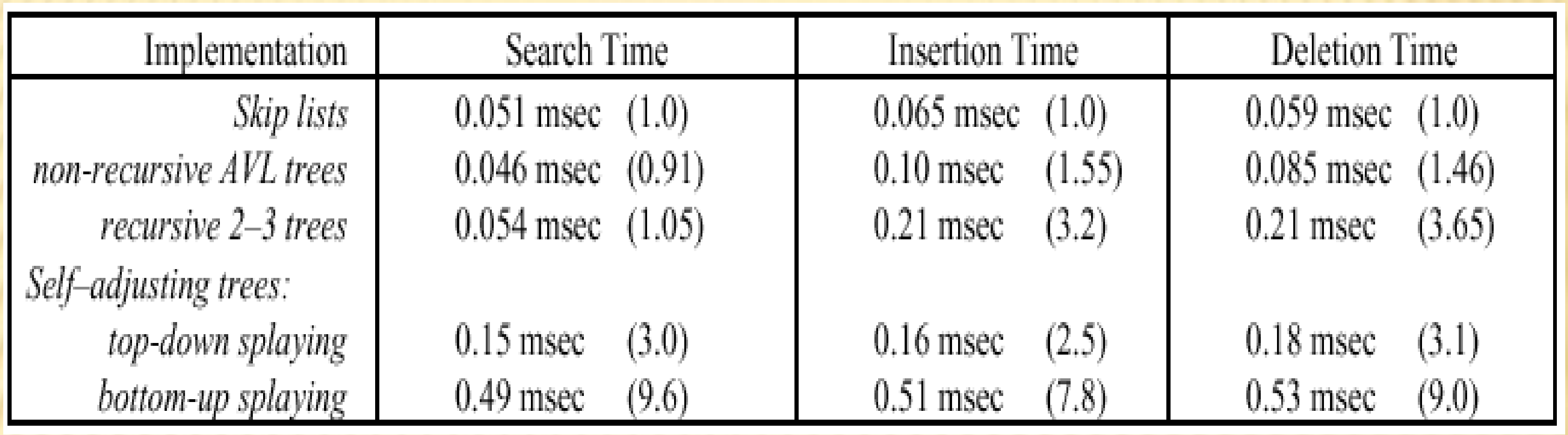
HEIGHT COMPLEXITY

- The running time of the search an insertion algorithms is affected by the height h of the skip list
- We can show that, a skip list with n items has height $O(\log n)$ with high probability.

TIME COMPLEXITY

- 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
- And we can show that, 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

SKIPLISTS VS OTHERS



Implementation	Search Time	Insertion Time	Deletion Time
Skip lists	0.051 msec (1.0)	0.065 msec (1.0)	0.059 msec (1.0)
non-recursive AVL trees	0.046 msec (0.91)	0.10 msec (1.55)	0.085 msec (1.46)
recursive 2–3 trees	0.054 msec (1.05)	0.21 msec (3.2)	0.21 msec (3.65)
Self–adjusting trees:			
top-down splaying	0.15 msec (3.0)	0.16 msec (2.5)	0.18 msec (3.1)
bottom-up splaying	0.49 msec (9.6)	0.51 msec (7.8)	0.53 msec (9.0)

From: “Skip Lists: A Probabilistic Alternative to Balanced Trees”

The times in this table reflect the CPU time on a Sun-3/60 to perform an operation in a data structure containing 216 elements with integer keys. The values in parenthesis show the results relative to the skip list time. The times for insertion and deletion do not include the time for memory management.

APPLICATIONS, PROS & CONS

APPLICATION

- Dictionaries
- Q-Map - The QMap class is a value-based template class that provides a dictionary
- Skip DB: It is an open-source database format using ordered key/value pairs.

OTHER APPROACHES

- Hashed Skip Lists
- Indexed Skip Lists
- Biased Skip Lists
- Skip Graph

ADVANTAGES

- The implementation it's direct and easier than the balanced tree's algorithm
- The memory requirements are less than the used for balanced trees
- Insertion and deletion don't need to be balanced again.
- Insertion, deletion, search and union operations support.

DISADVANTAGES

- Each node in a Skip list has an array of forward pointers that link the remote nodes.
- The number of pointers ahead in a node its determined by a probabilistic function during insertion; the order its randomized.
- We search for an element by traversing forward pointers that do not overshoot the node containing the element being searched for.

SUMMARY

- Though theoretically skip list has no need to be implemented as it has certain side effects compared to balanced trees
- But as the ease of design and the expected results are concerned skip lists are nowhere bound to only data structure class assignmens unlike balanced trees.

THANK YOU

- Bibliography:

  - http://www.lesliesanford.com/Programming/SkipList.shtml
  - **A Skip List Cookbook by** William Pugh, Institute for Advanced Computer Studies, July 1989.
  - ww3.algorithmdesign.net/handouts/**SkipLists**.pdf
  - http://en.wikipedia.org/wiki/Skip_list