Linearity of Expectation & Skip Lists

COL 106

Random Variable

- Sample Space
- Probability distribution

- Expectation of a (numerical) random variable

Expectation

- I toss coin thrice. What is the expected number of heads.

Linearity of Expectation

Theorem 2.2. *Let $X_1, \ldots, X_n$ be any finite collection of discrete random variables and let $X = \sum_{i=1}^{n} X_i$. Then we have*

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i].$$

- I toss coin thrice. What is the expected number of heads?

Linearity of Expectation (Example 2)

- m balls are thrown into one of n bins independently and uniformly at random. What is expected number of balls in bin j?

If each of ***n*** items is present in a set with prob. ***p***, the expected size of the set is ***np***

Linearity of Expectation (Example 3)

- Same question as before. What is the expected number of empty bins?

Randomized Algorithms

- A randomized algorithm performs coin tosses (i.e., uses random bits) to control its execution
- It contains statements of the type:

$b \leftarrow random()$

if $b = 0$

  do A …

else $\{ b = 1 \}$

  do B …

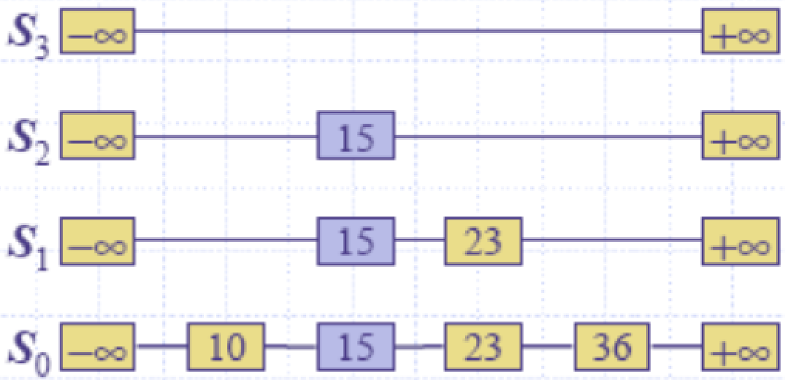
- Its running time depends on the outcomes of the coin tosses
- We analyze the expected running time of a randomized algorithm under the following assumptions:
  - the coins are unbiased, and
  - the coin tosses are independent
- The worst-case running time of a randomized algorithm is often large but has very low probability (e.g., it occurs when all the coin tosses give “heads”)

Randomized Quicksort

- Pick the pivot uniformly randomly from the array

- Expected Time Complexity of Randomized Qsort?

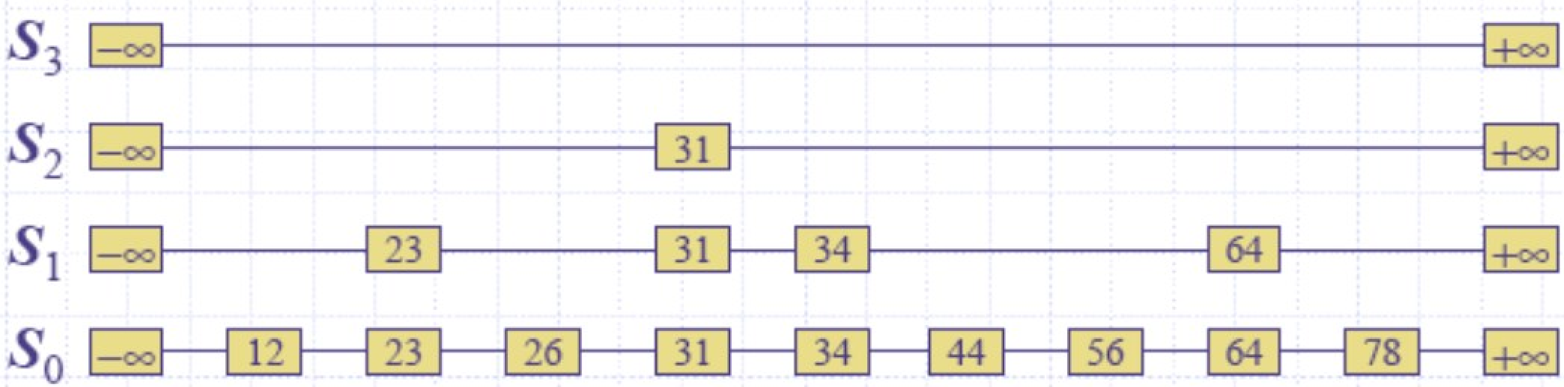
Skip Lists



Sorted Arrays & Linked Lists

What is a Skip List

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



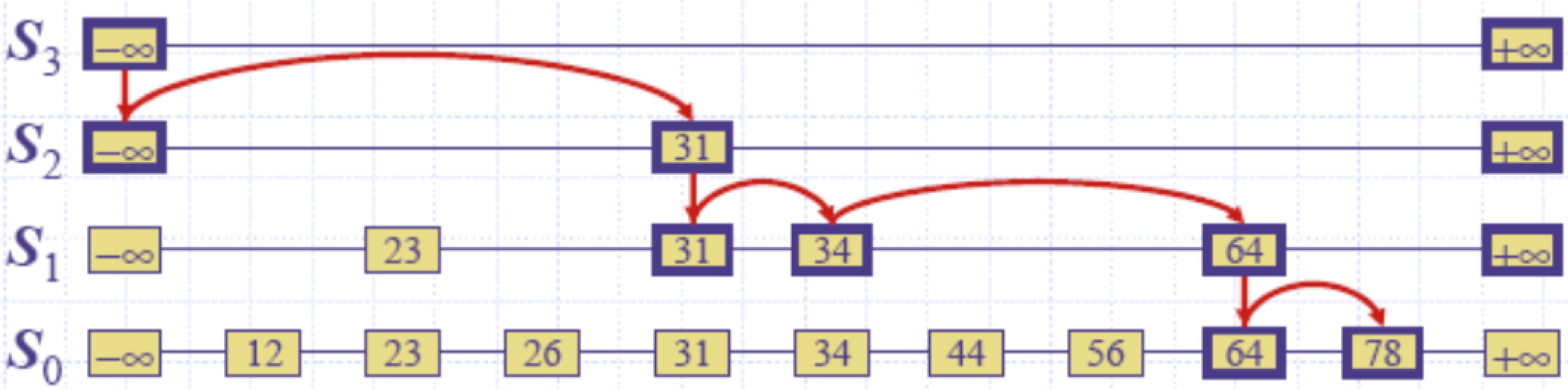
Search

- We search for a key x in a a skip list as follows:
  - We start at the first position of the top list
  - At the current position p, we compare x with $y \leftarrow key(after(p))$

    $x = y$: we return $element(after(p))$

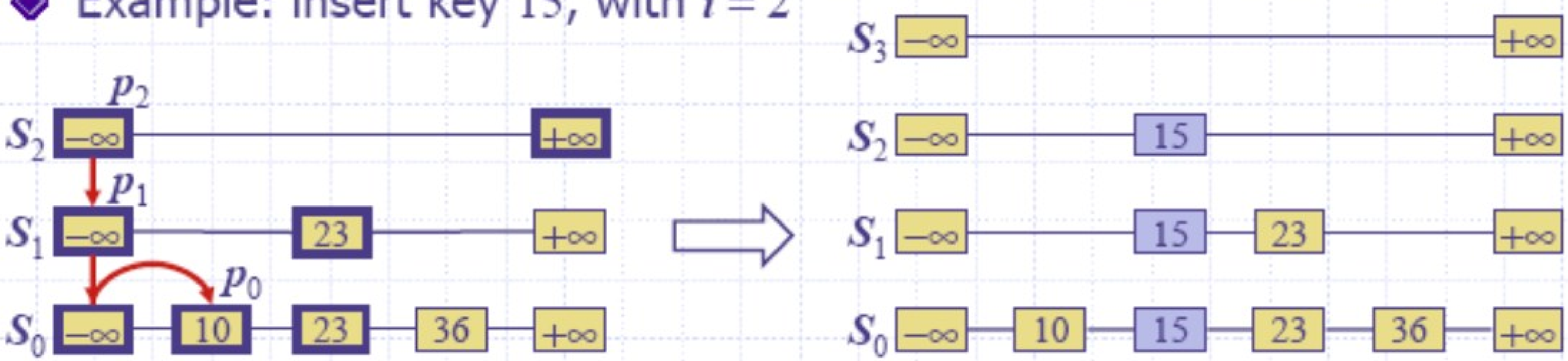
    $x > y$: we "scan forward"

    $x < y$: we "drop down"
  - If we try to drop down past the bottom list, we return *NO_SUCH_KEY*
- Example: search for 78

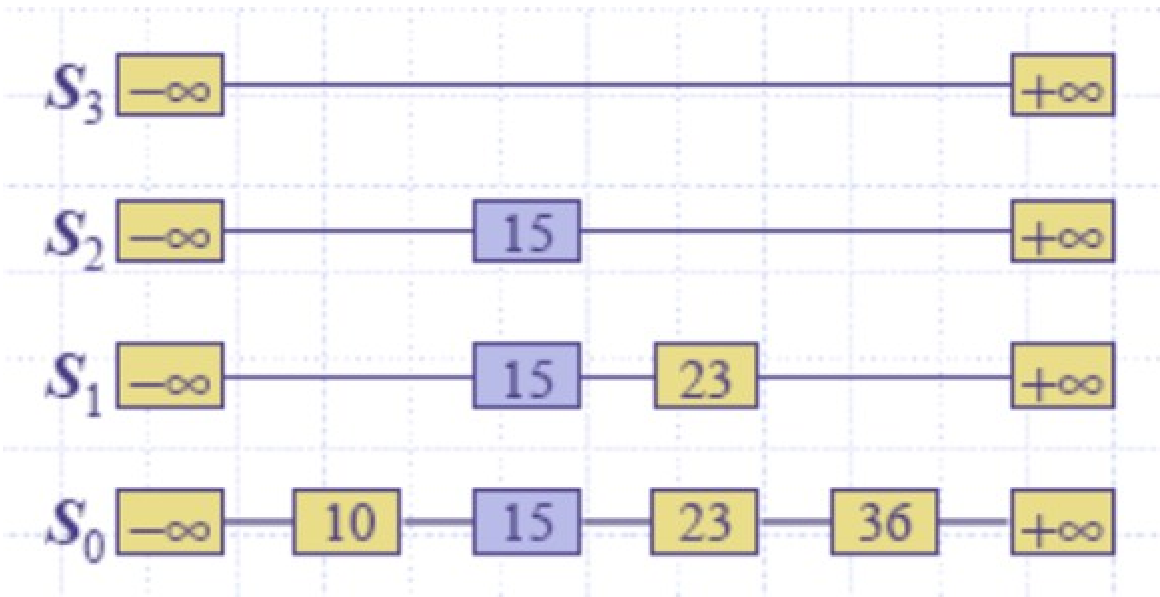


Insertion

- 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 $i = 2$

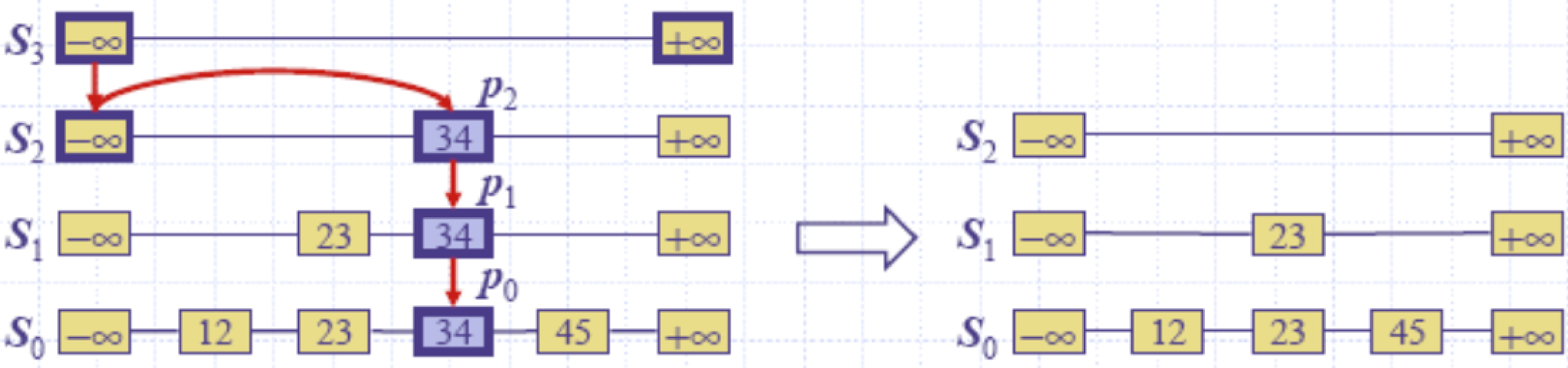


Insert 12



Deletion

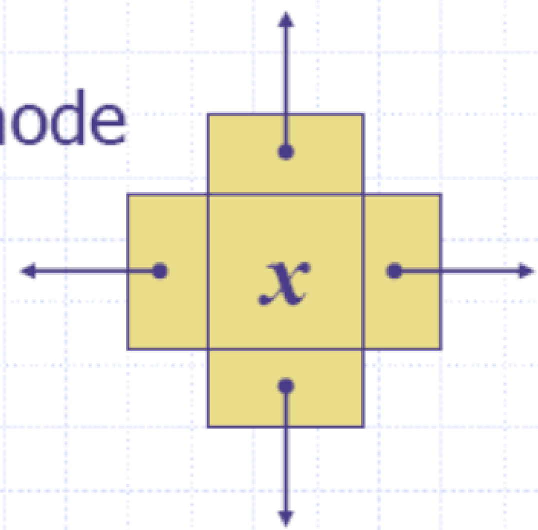
- To remove an item with key x from a skip list, we proceed as follows:
  - We search for x in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key x, where position p_j is in list S_j
  - We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, \ldots, S_i$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

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 $n/2^i$

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

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

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

Height

- The running time of the search an insertion algorithms 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

Height

- The running time of the search an insertion algorithms 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 $n/2^i$

Height

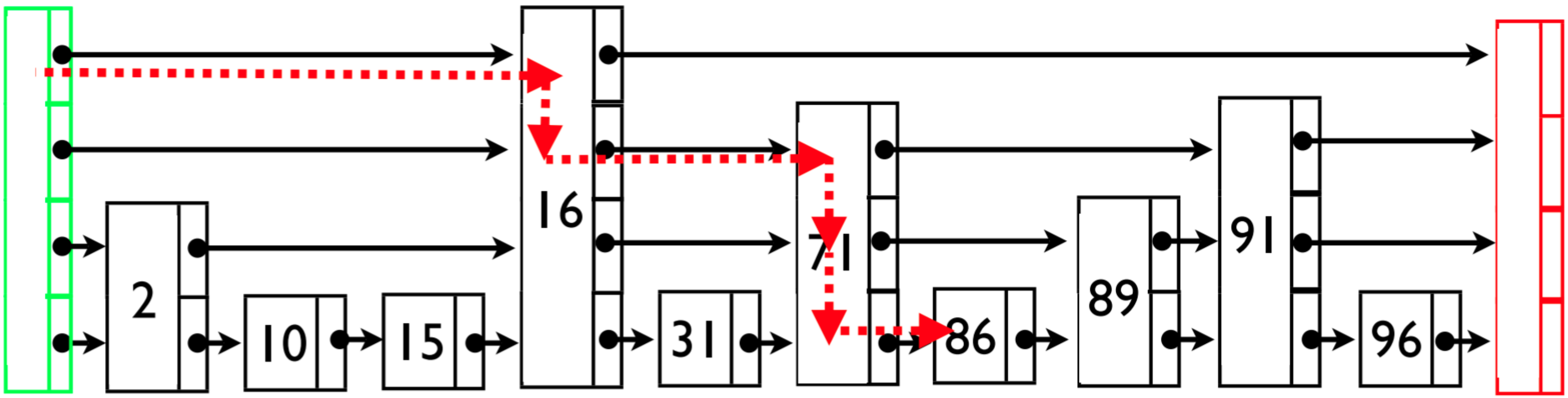
- The running time of the search and insertion algorithms is affected by the height h of the skip list
- We show that with high probability, a skip list with n items has height $O(\log n)$
- We use the following additional probabilistic fact:

  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 that 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 Time: Backward Analysis

Consider the *reverse* of the path you took to find k:



Note that you *always* move up if you can.
(because you always enter a node from its topmost level when doing a find)

Search Time: Backward Analysis

- What's the probability that you can move up at a give step of the reverse walk?

0.5

- Steps to go up j levels =
  Make one step, then make either
  C(j-1) steps if this step went up [Prob = 0.5]
  C(j) steps if this step went left [Prob = 0.5]

- Expected # of steps to walk up j levels is:

$$C(j) = 1 + 0.5C(j-1) + 0.5C(j)$$

Search Time: Backward Analysis

- Expected # of steps to walk up j levels is:

$$C(j) = 1 + 0.5C(j\text{-}1) + 0.5C(j)$$

So:

$$2C(j) = 2 + C(j\text{-}1) + C(j)$$

$$C(j) = 2 + C(j\text{-}1)$$

Expected # of steps at each level = 2

- Expanding $C(j)$ above gives us: $C(j) = 2j$
- Since $O(\log n)$ levels, we have $O(\log n)$ steps, expected

Summary

- Skip Lists are easy to implement
- They have expected complexity of O(log n)
- They have O(n) space