CMPT 225: Data Structures & Programming – Unit 21 – Skip Lists

Dr. Jack Thomas
Simon Fraser University
Spring 2021

Today's Topics

- Binary Search and Data Structures
- The Skip List
- Skip List ADT
- Skip Lists in Java

Implementing Ordered Maps

- Ordered Maps, recall, let us retrieve a subset of entries from a Map whose keys fall between some criteria.
- An Ordered Search Table based on an Arraylist was one example implementation, but its efficiency left something to be desired (put and remove were both O(n)).
- Can we do better?

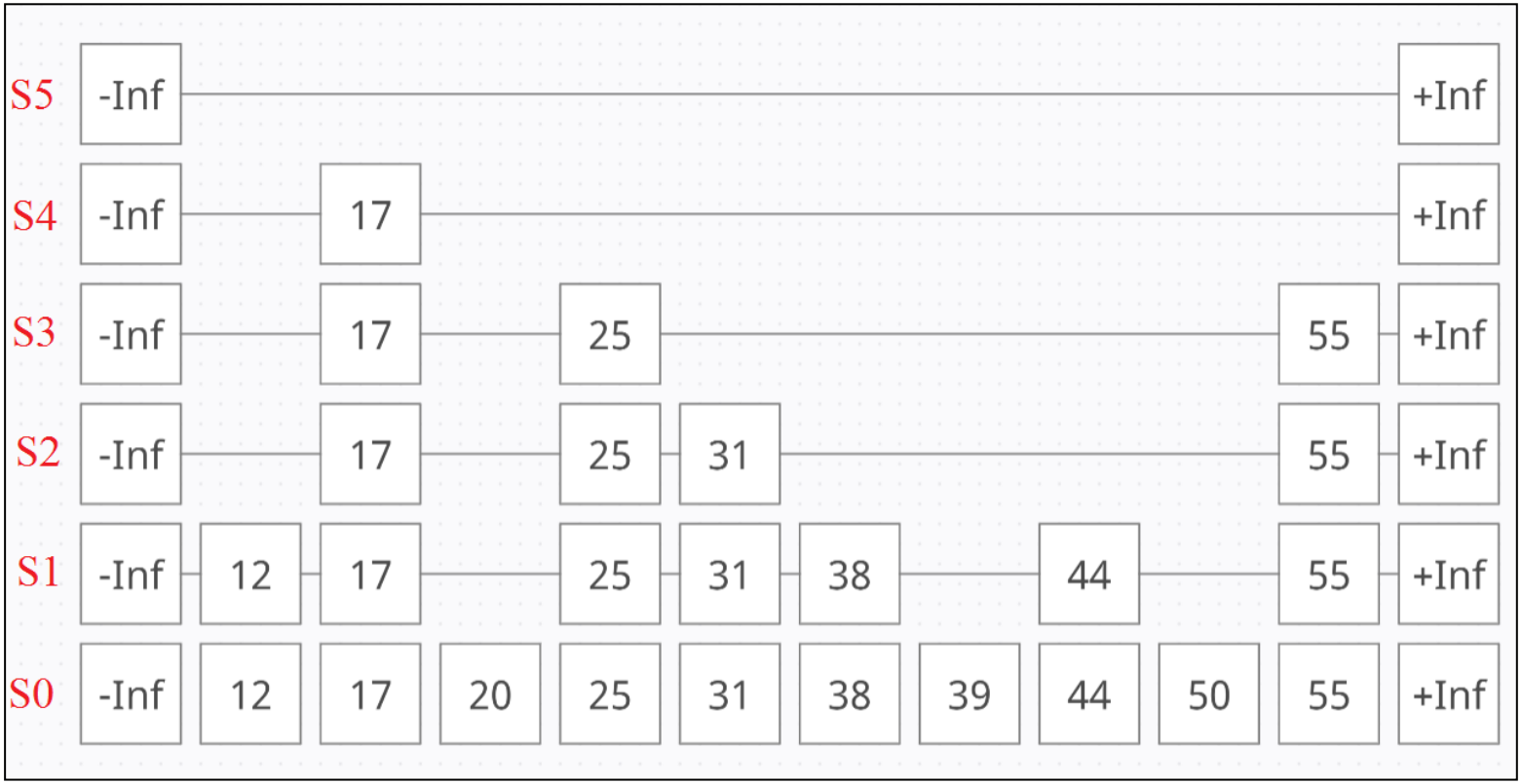
Introducing the Skip List

- A weird data structure we can build our Ordered Map on top of to improve our insertion and removal times to O(log n) on average.
- Why do I say weird? How is something's worst-case run-time "on average"?

- Let's flip some coins.

Image credit: http://clipart-library.com/clipart/339002.htm

Behold, A Skip List



- We started with ten entries, arranged smallest to largest by integer keys, and this is what we ended up with. How?

Defining a Skip List

- A Skip List S for a Map M consists of a series of lists $\{S_0, S_1, \ldots S_h\}$, with h as the height of the Skip List.
- Each List stores a subset of the entries of M sorted by increasing keys, plus header and trailer entries with two special keys (plus and minus infinity) to guarantee they bracket all other entries.

Defining a Skip List

- List S_0 contains every entry of the map M (plus the header and trailer).
- For i = 1,…,h-1, list S_i contains a randomly generated subset of the entries in list S_{i-1} (plus the header and trailer).
- We randomly generate this subset for S_i by flipping a coin for each entry in S_{i-1}, and adding it on a heads.
- List S_h contains only the header and trailer.

Probable Properties

- If S0 has n entries, we expect S1 to have around n/2 entries, S2 around n/4, and so on.
- Any level Si of the Skip List probably has n/2i entries.
- The height h of the Skip List will probably be about log n.
- All of this is “probably” because the coin flip is truly random, to avoid patterns.

Towers and Levels

- We think of each separate list in S as being a horizontal level.
- Meanwhile, the entries that reappear in each list are thought of as arranged vertically into a tower.
- We implement this through two different sets of doubly-linked lists, one for levels (the S lists) and one for each tower.
- This also means we can move vertically or horizontally through the Skip List in O(1).

Skip List: The ADT

- A data structure that implements and extends the Ordered Map ADT.
- Improves the average time of search and update operations to O(log n) through random arrangements.
- Standard methods include those of the Ordered Map ADT, along with:
  - Next: The position following a given position on a level.
  - Prev: The position preceding a given position on a level.
  - Below: The position below a given position in a tower.
  - Above: The position above a given position in a tower.

Skip Lists in Java

- A Skip List can be found in Java as part of the ConcurrentSkipListMap class.

```
ConcurrentSkipListMap<Integer, String> exampleSkipList = new ConcurrentSkipListMap<Integer, String>();
exampleSkipList.put(3, "First");
exampleSkipList.put(2, "Second");
exampleSkipList.put(4, "Third");
System.out.println(exampleSkipList.remove( key: 3));
System.out.println(exampleSkipList.remove( key: 2));
System.out.println(exampleSkipList.remove( key: 4));
System.out.println();
```

```
First
Second
Third
```

- Nothing you haven't seen before, what's really changed is under the hood at the implementation level.

Okay Cool Why Did We Do All That Though

- In a way, our Skip List is similar to a structural version of the Binary Search algorithm.

- It allows us to SkipSearch for a given key, navigate the positions in our grid of doubly linked lists, and turn up a matching entry.

Skip Search

- The Start Position of S is the top-left position, one of our special header nodes. Starting with that node, and treating p as the entry at our current position, we perform the following:
  1. If S.below(p) is null, the search terminates. We are at the bottom and have located the largest entry in S with a key less than or equal to the search key k. Otherwise, we drop down to the next lower level in the current tower by setting p to S.below(p).
  2. Starting at position p, we move p forward until it is at the right-most position on the present level such that key(p) <= k. We call this the scan forward step. This can lead to us moving all the way to the trailer node, or not moving at all.

Insertion

- When inserting a new entry, start with a skip search to find the position in S0 with the largest key less than or equal to our new key.
- We then insert our new entry immediately after that position.
- Then, we build the tower by backtracking up each list and flipping a coin – heads, add a new entry to the tower and keep going. Tails, you're done building.

```
Algorithm SkipInsert(k,v):
        Input: Key k and value v
        Output: Topmost position of the entry inserted in the skip list.
        p <- SkipSearch(k)
        q <- null
        e <- (k,v)
        i <- -1
        repeat
                i <- i + 1
                if i >= h then
                        h <- h+1
                        t <- next(s)
                        s <- insertAfterAbove(null,s,(-inf,null))
                        insertAfterAbove(s,t,(+inf, null))
                while above(p) = null do
                        p <- prev(p)
                p <-above(p)
                q<-insertAfterAbove(p,q,e)
        until coinFlip() = tails
        n <- n+1
        return q
```

Building Beyond h

- If a new entry flips heads enough time to take it beyond the current height, what should you do?
- You can choose to either restrict h to some maximum, perhaps the original height on creation, or allow new layers to be built.
- While these make some difference in terms of constant time operations added, they don’t change the O() analysis.

Removal

- Perform a skip search. If the key of the entry you find isn’t exactly the same as the key you were looking for, return null.
- Otherwise, remove P, then back-track through its tower and remove it on each subsequent level as well.

Analysis

- The Skip List allows for insertions and removals to take O(log n) thanks to Skip Search and the height being roughly log n.
- Some Skip List methods involve multiple O(log n) operations one after the other, like working down to S0 then back up while building or erasing a tower, but these are not nested, so they avoid being squared.
- The only odd point is that, due to the coin-flip nature, different Skip Lists will end up being a little bit more or less efficient, and there's little way of knowing how well they turned out before they're used.

Analysis

- One advantage of Skip Lists is that the coin-flip mechanic detaches our notion of “average case” from whatever the keys actually are – even if the keys are clustered in a way that frustrates a Hash Table, they have no impact on the Skip Lists we generate.
- Any strict rule, like building Si out of every other entry from Si-1, would be prone to patterns on some data-sets and end up being less efficient on average.

Recap – Skip To The End

- Skip Lists are a data structure we can use to implement an Ordered Map with improved run-time efficiency, on average.
- They accomplish this through randomly constructing a special grid of entries.
- Skip search allows us to navigate this grid in O(logn), on average.
- Java offers a Map based on a Skip List called ConcurrentSkipListMap