CSSE 230 Day 25

Skip Lists

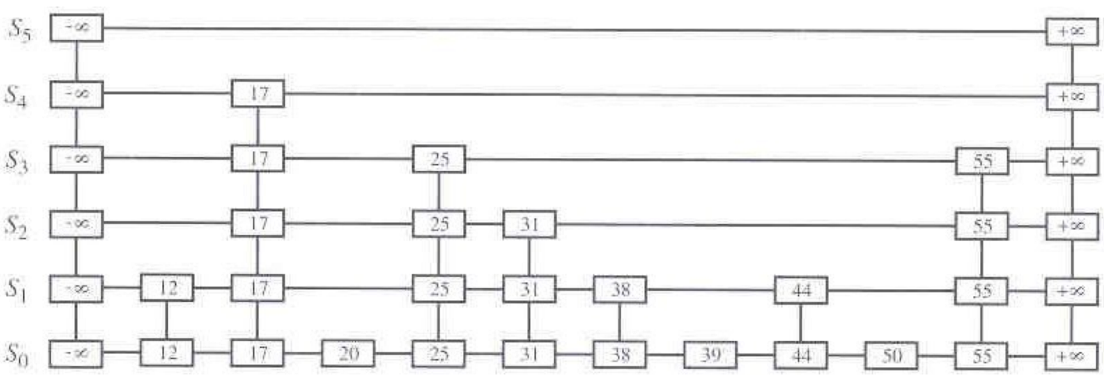
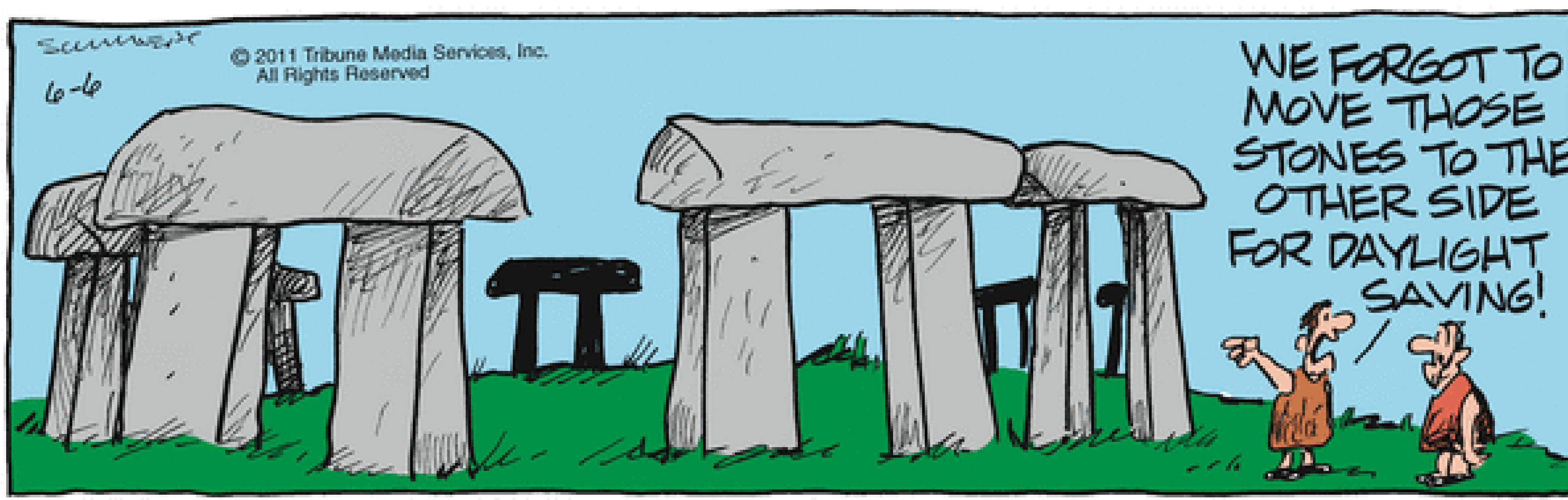


Figure 8.9: Example of a skip list.

Reminders/Announcements

- Complete the **EditorTrees partner evaluation** by Wednesday night



Skip Lists

An alternative to balanced trees

Sorted data.

Random.

Expected times are O(log n).

An alternative to AVL trees

- Indexed lists.
  - One-level index.
  - 2nd-level index.
  - 3rd-level index
  - log-n-level index.
- Problem: insertion and deletion.

Remember the problem with keeping trees *completely* balanced"?

- Solution: Randomized node height: Skip lists.
  - Pugh, 1990 CACM.
- http://iamwww.unibe.ch/~wenger/DA/SkipList/

Notice that skip lists do not share with binary trees the problem that threads are designed to solve.

A slightly different skip list representation

- Uses a bit more space, makes the code simpler.
- Michael Goodrich and Roberto Tamassia.

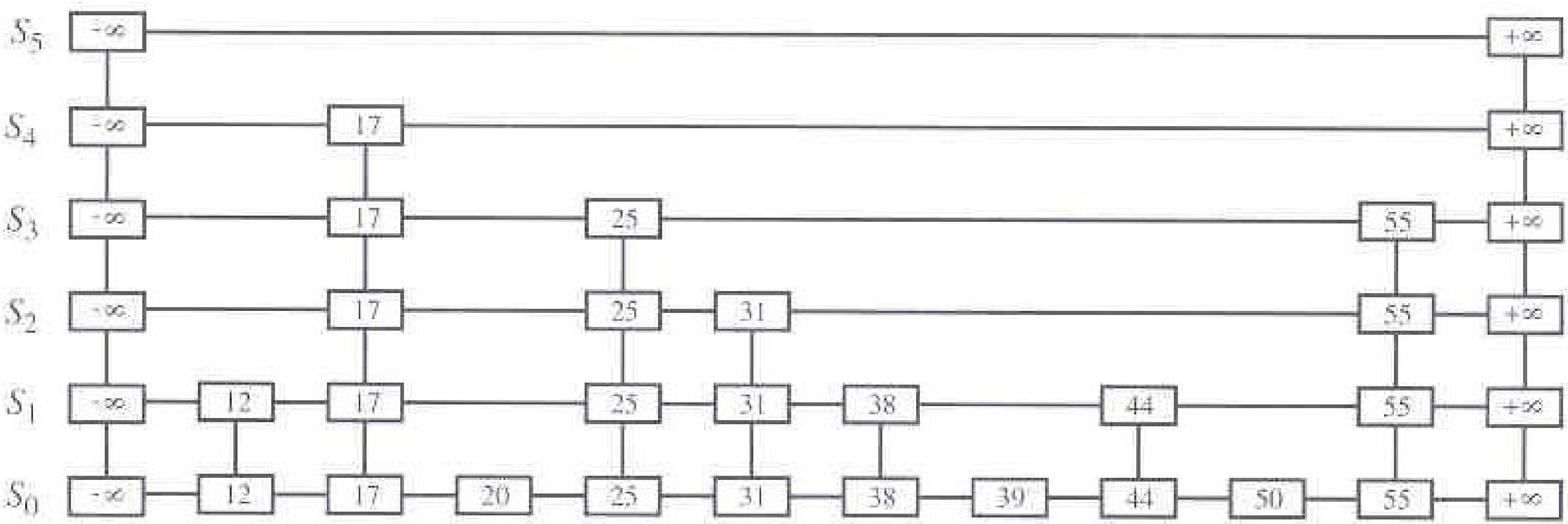


Figure 8.9: Example of a skip list.

Methods in SkipListNode class

after(p): Return the position following p on the same level.

before(p): Return the position preceding p on the same level.

below(p): Return the position below p in the same tower.

above(p): Return the position above p in the same tower.

Search algorithm

1. If S.below(p) is null, then the search terminates—we are ***at the bottom*** and have located the largest item in S with key less than or equal to the search key k. Otherwise, we ***drop down*** to the next lower level in the present tower by setting $p \leftarrow S$.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) $\leq k$. We call this the ***scan forward*** step. Note that such a position always exists, since each level contains the special keys $+\infty$ and $-\infty$. In fact, after we perform the scan forward for this level, p may remain where it started. In any case, we then repeat the previous step.

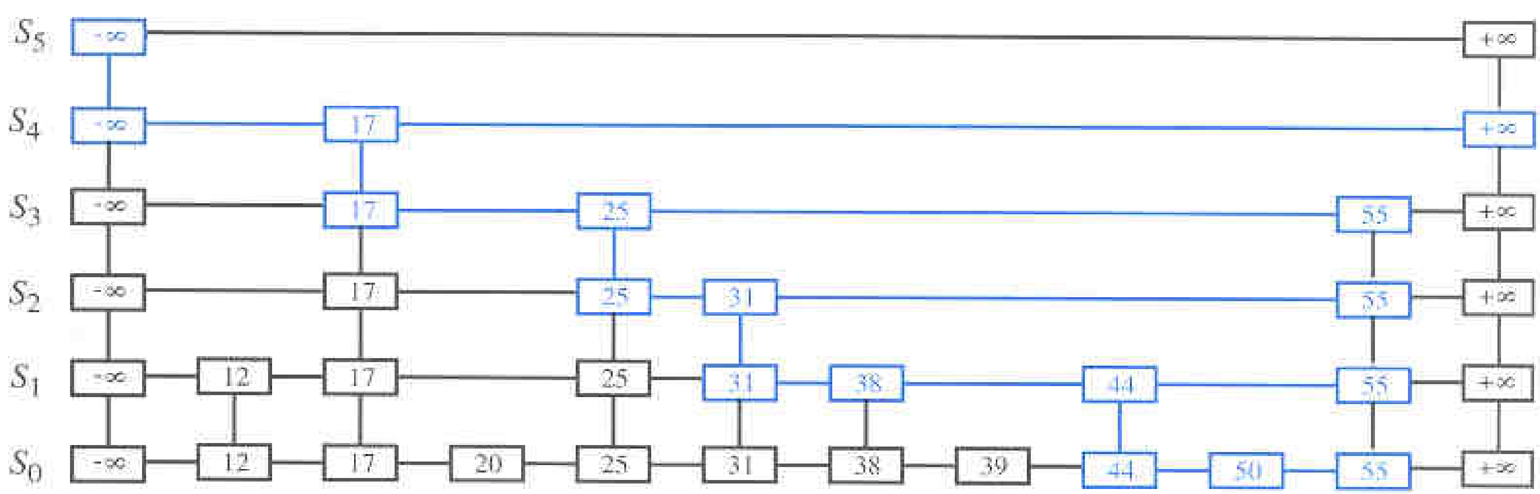
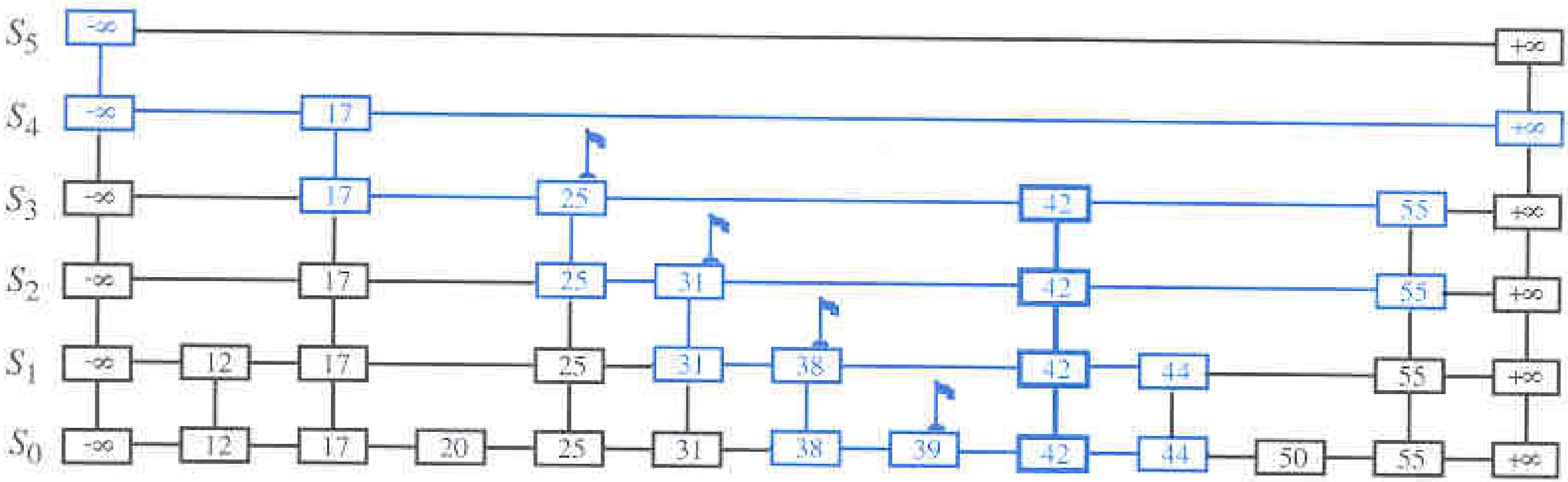
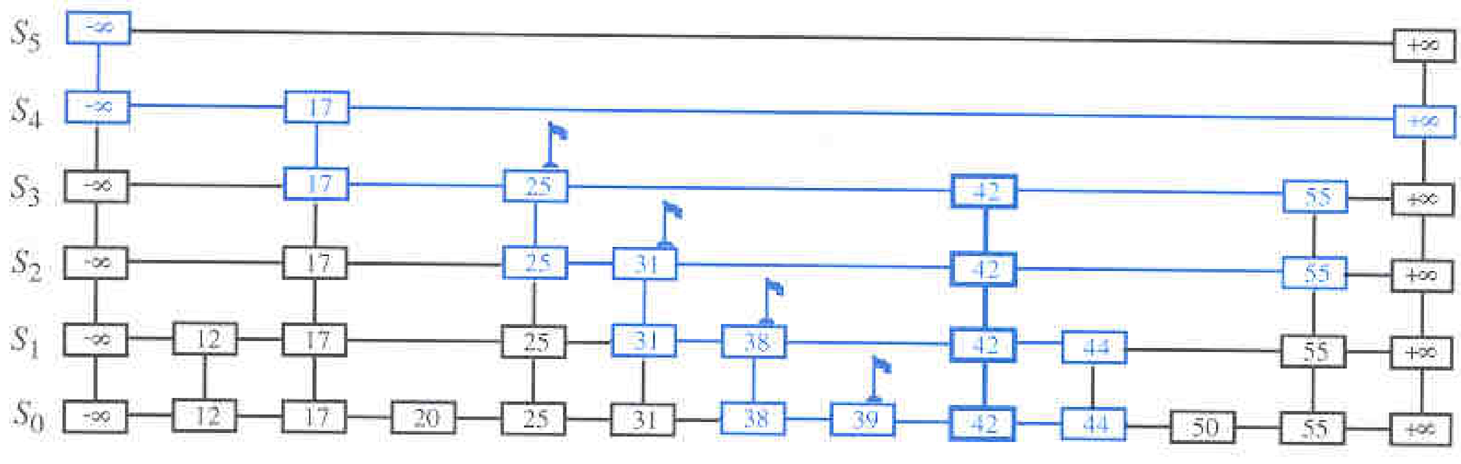


Figure 8.10: Example of a search in a skip list. The positions visited when searching for key 50 are highlighted in blue.

Insertion diagram



Insertion algorithm



Algorithm SkipInsert(k, e):

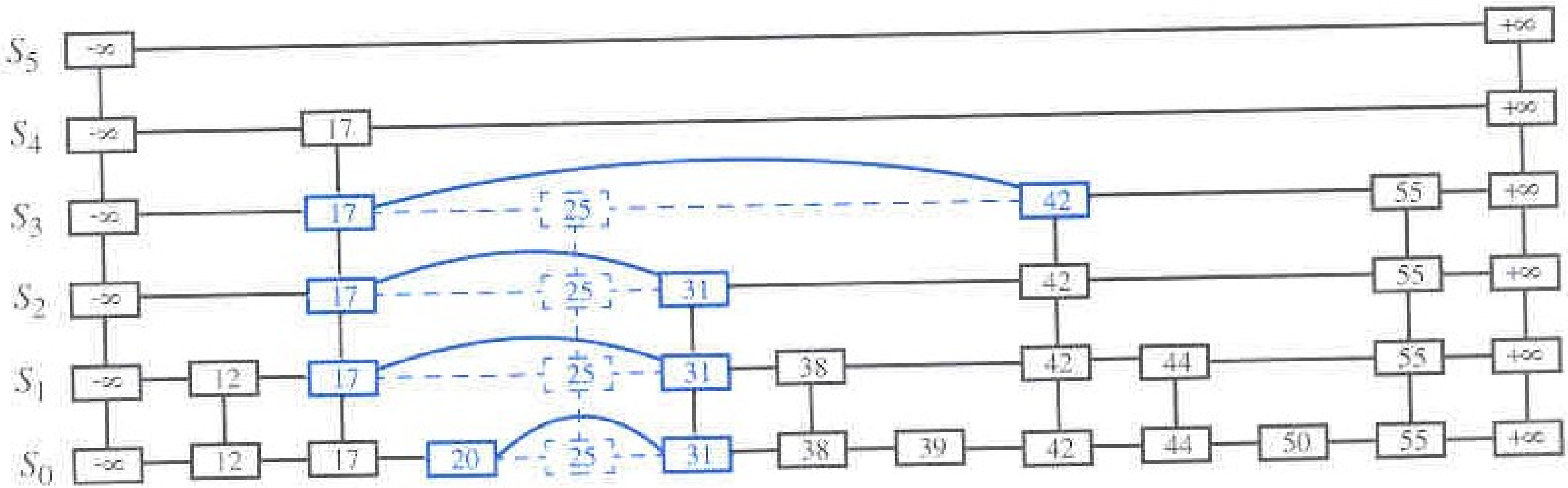
Input: Item (k, e)

Output: None

$p \leftarrow$ SkipSearch(k)
$q \leftarrow$ insertAfterAbove$(p, \textbf{null}, (k, e))$ {we are at the bottom level}
while random$() < 1/2$ **do**
  while above$(p) =$ **null do**
    $p \leftarrow$ before(p) {scan backward}
  $p \leftarrow$ above(p) {jump up to higher level}
  $q \leftarrow$ insertAfterAbove$(p, q, (k, e))$ {insert new item}

Code Fragment 8.5: Insertion in a skip list, assuming random() returns a random number between 0 and 1, and we never insert past the top level.

Remove algorithm



(sort of) Analysis of Skip Lists

- No guarantees that we won't get O(N) behavior.
  - The interaction of the random number generator and the order in which things are inserted/deleted *could* lead to a long chain of nodes with the same height.
  - But this is **very** unlikely.
  - ***Expected*** time for search, insert, and remove are O(log n).