Lecture 16

Tries & Skip Lists

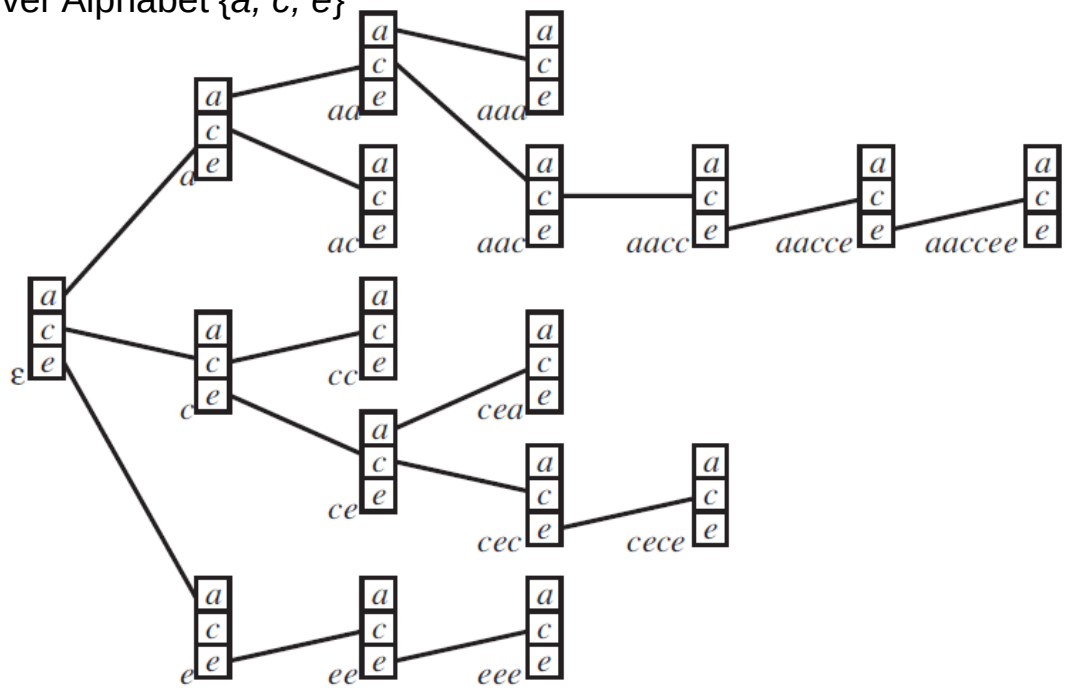
Dr. Yusuf H. Sahin
Istanbul Technical University

sahinyu@itu.edu.tr

Trie (Prefix Tree)

- Similar to the balanced binary search tree, a trie is an effective data structure to store strings which is deriven from “reTRIEval.”
  - This structure was invented by de la Briandais (1959)

A Trie over Alphabet {*a*, *c*, *e*}



- The degree of each node is at most the size of the alphabet.
- In the tree structure, the node that corresponds to the empty prefix is the root.

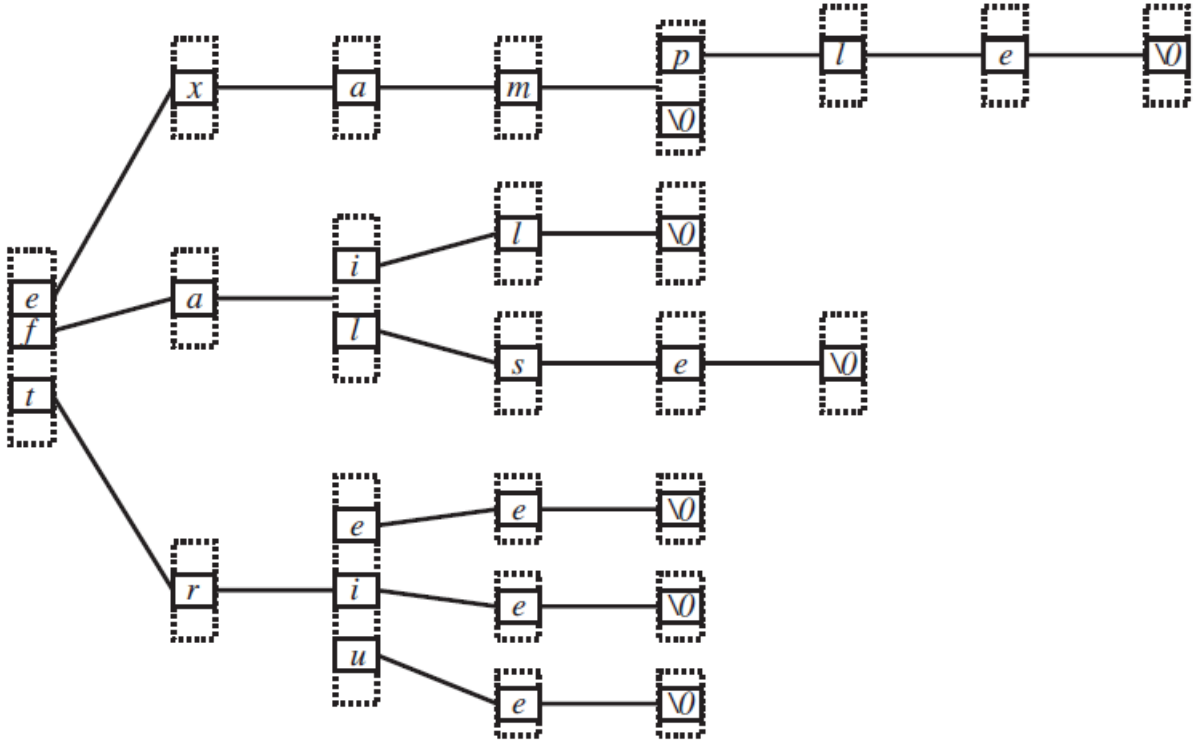
- Time complexity of searching for a word is O(L) where L indicates the length of the word.

Advanced Data Structures, Peter Brass

Trie (Prefix Tree)

- The space requirement is huge! However, even with unlimited space, the size of the alphabet impacts the insert and delete operations as new nodes need to be initialized with NULL pointers.
- **Theorem:** While the find operation is $O(L)$, insert and delete operations will cost $O(|A|L)$. The space complexity to store the strings $w_1, w_2, \ldots, w_n$ is $O(|A| \sum_i length(w_i))$.

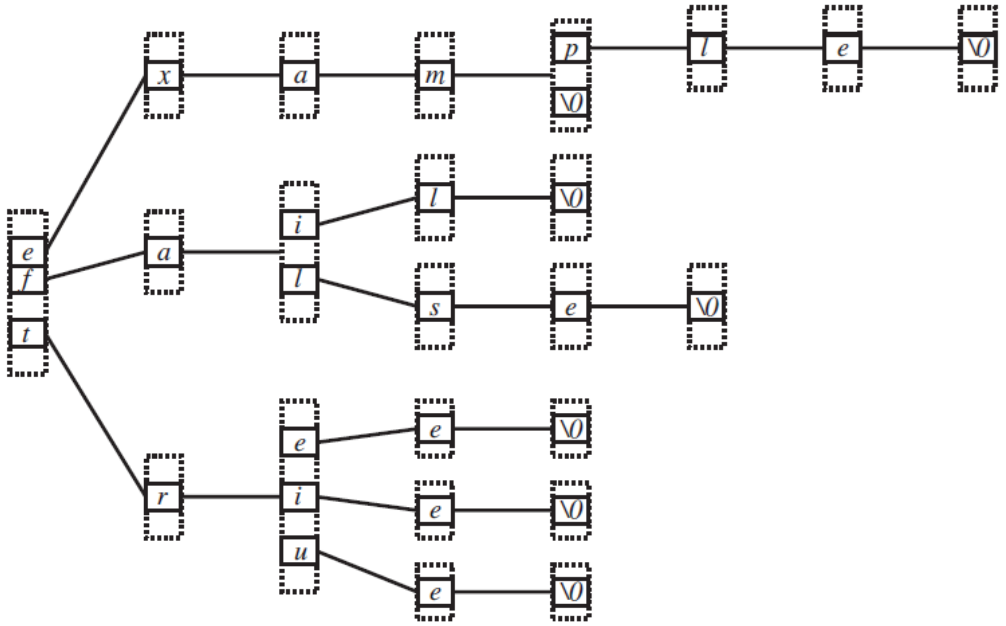
Trie for the Strings *exam, example, fail, false, tree, trie, true*

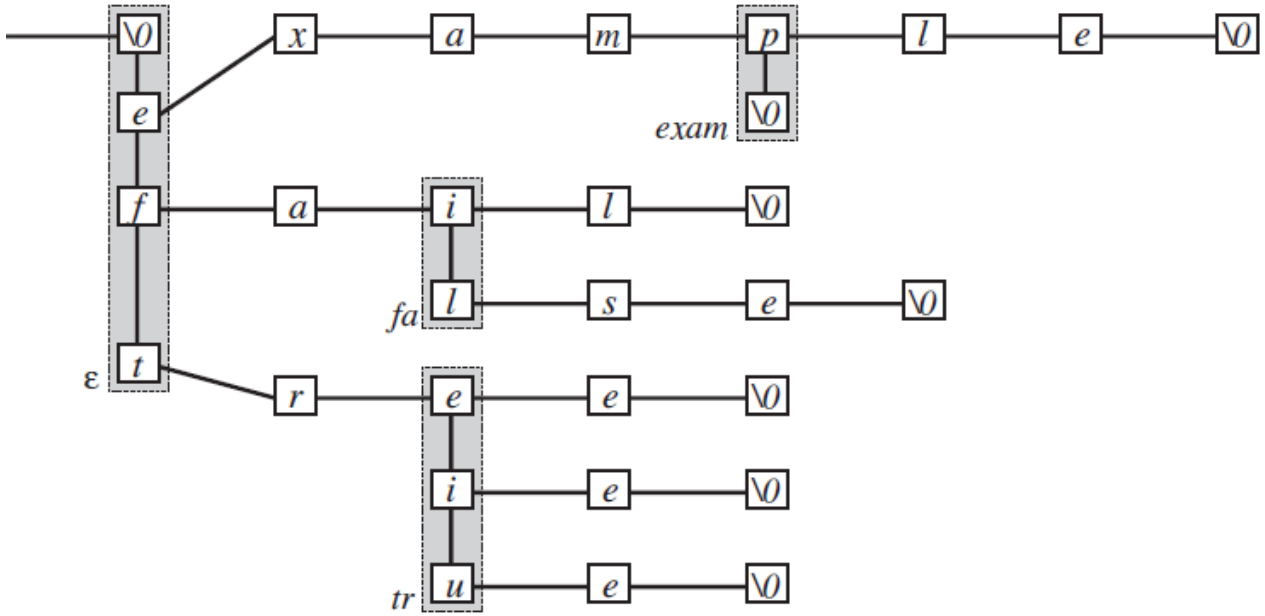


- Various methods exist to eliminate the issues caused by large |A|and empty nodes.
- Each approach involves a trade-off,
  - slight increase in query time vs space efficiency

  and faster update times.

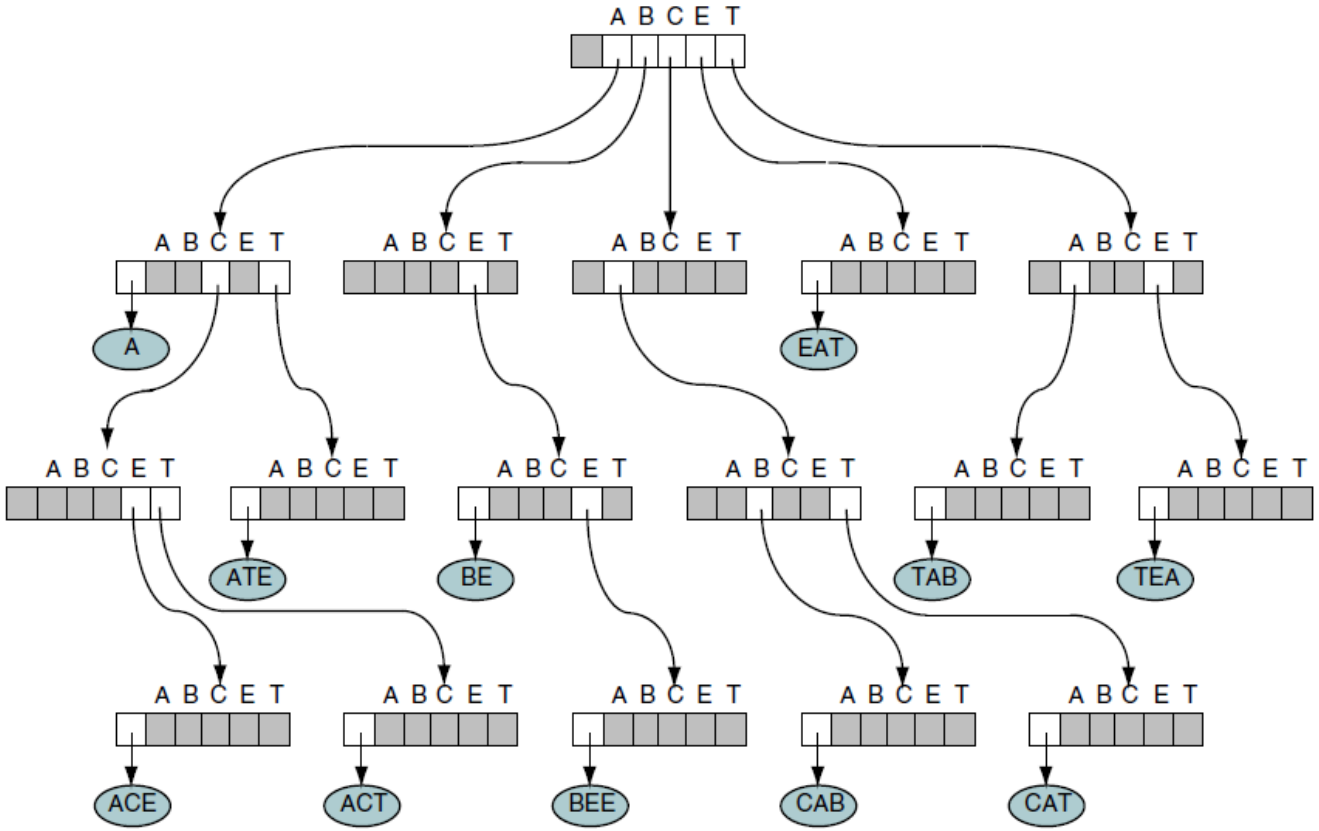
List Node Approach



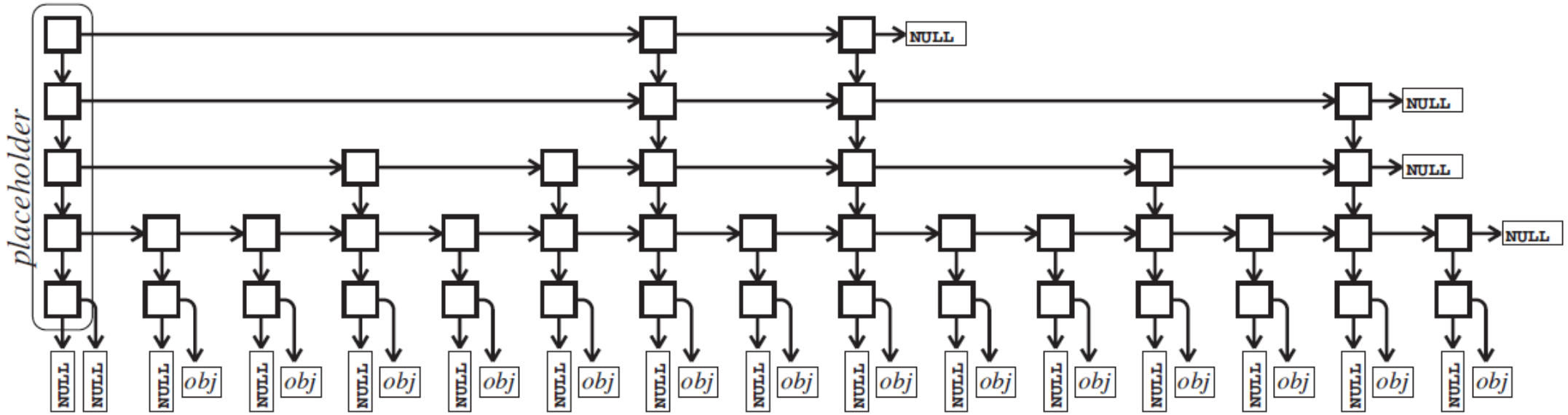


By list-node approach, now the find operation costs $O(|A|\ L)$, The space complexity to store the the strings $w_1, w_2, \ldots, w_n$ is $O(\sum_i length(w_i))$.

Example: A trie for 11 words & 5 letters



Skip Lists



From the lecture notes of Prof. Scott Schaefer
https://people.engr.tamu.edu/schaefer/teaching/index.html

Skip Lists

- Randomized Data Structures
- Enhances a sorted linked list with forward pointers that skip intermediate elements for faster access.

- On a singly linked list
  - Add a second list containing every second element of the original.
  - Reduces comparisons to:n/2 on the second list +1 additional comparison on the first list.

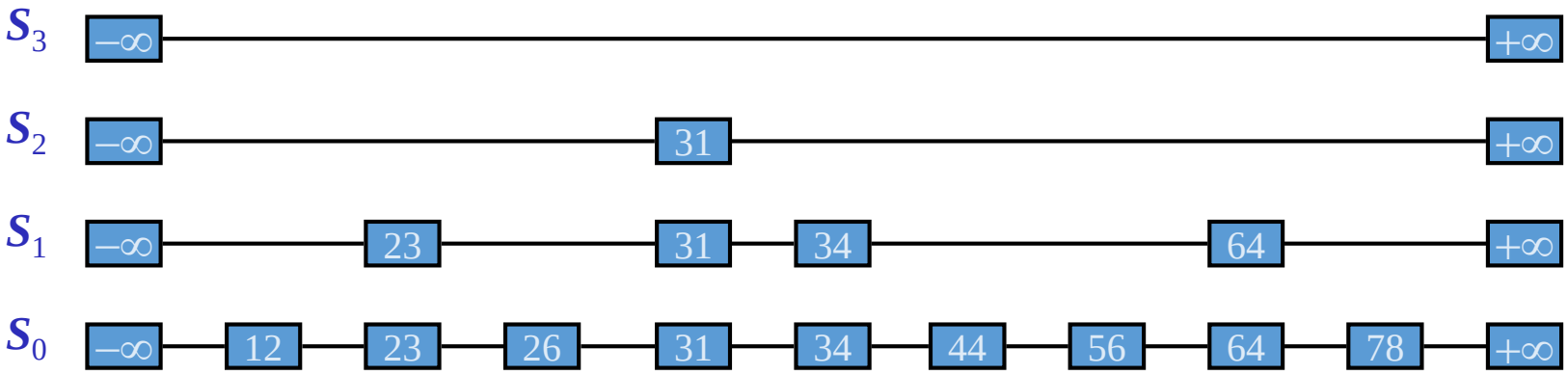
- Iterative Construction:
  - Add lists for every $2^i th$ element (i=1,…,k).

- Comparisons needed:
  - $n/2k$ on the kth list.
  - +1 for each lower-level list.

- k=log$n$ results in O(log$n$) for find operations.

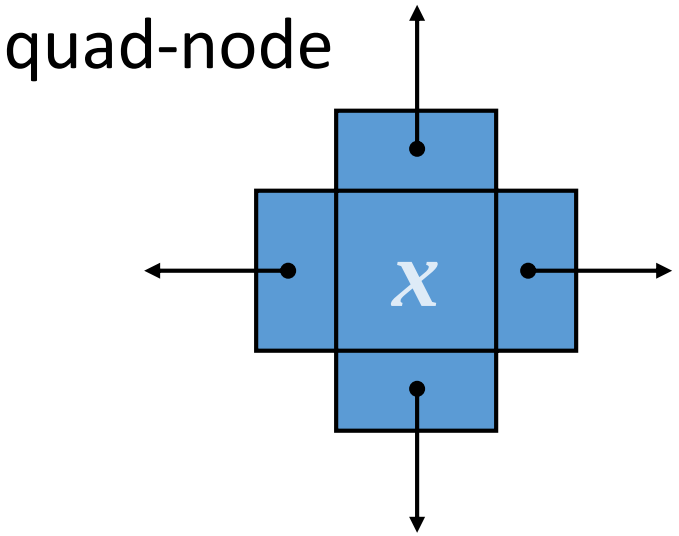
Skip Lists

- 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 non-decreasing 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



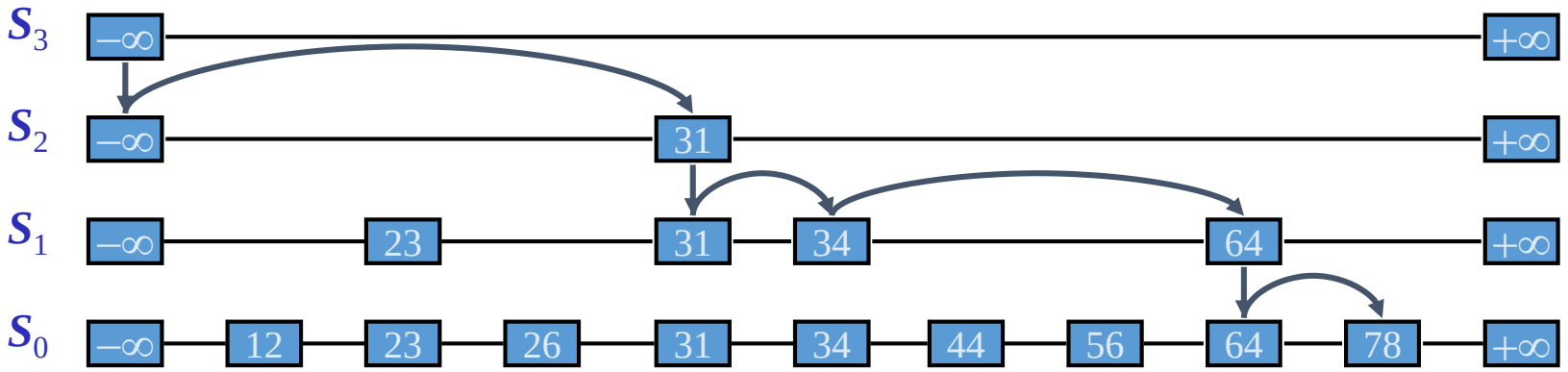
Skip List Implementation

- 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
- Also, we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them



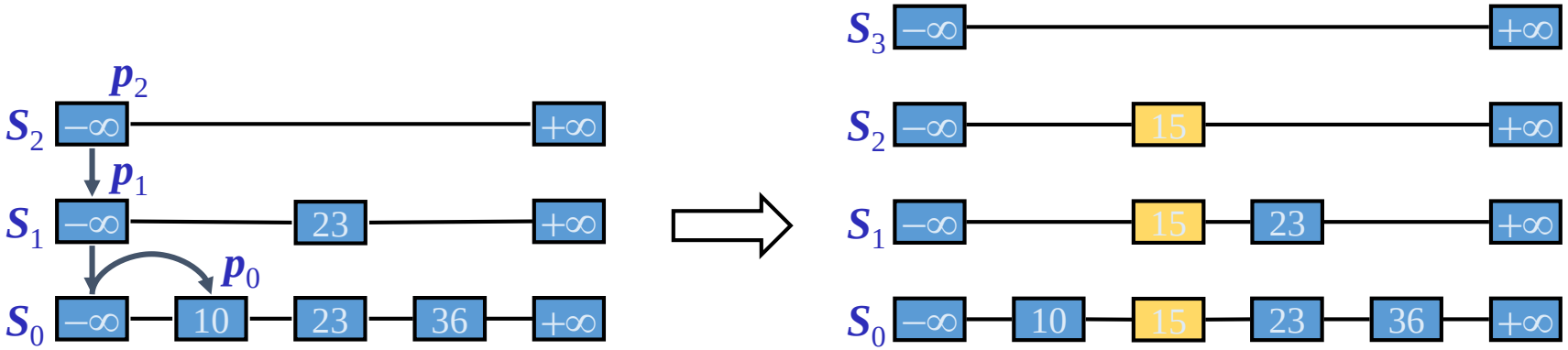
Skip List Search

- 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



Skip List Insert

- 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 $i = 2$



Skip List Delete

- 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