Zip-zip Trees

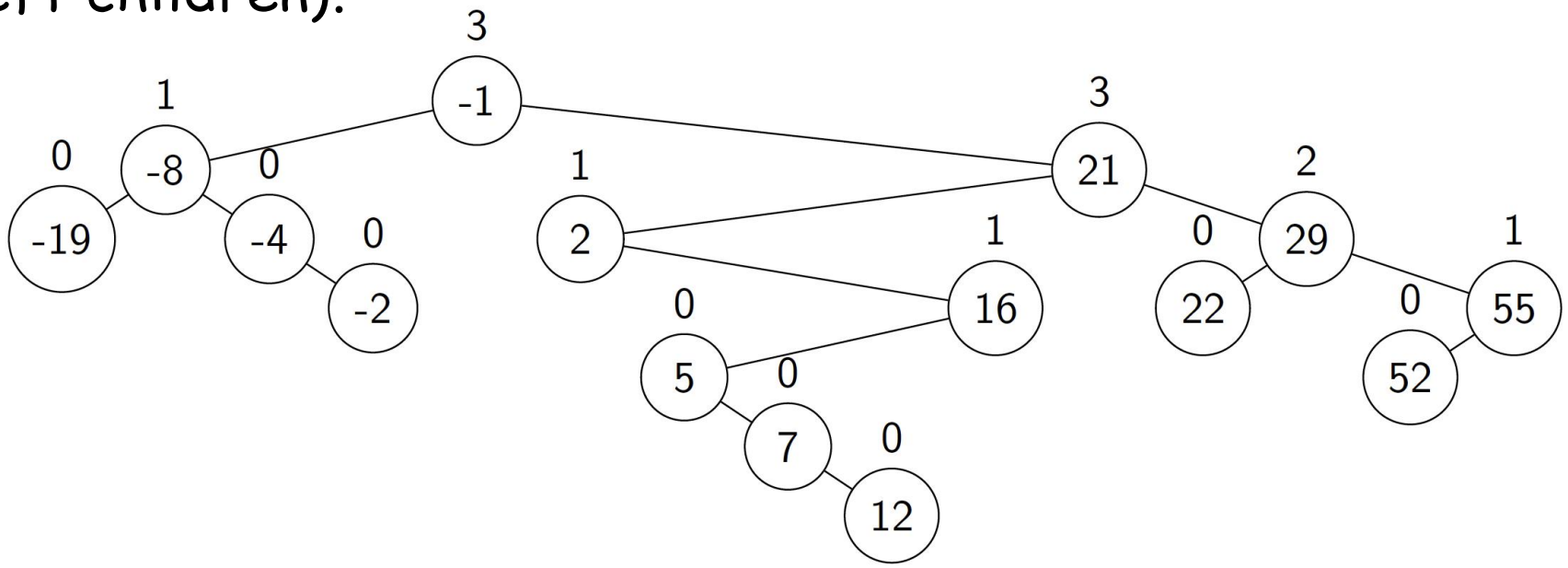
Michael Goodrich

CS 165

Some slides adapted from Ofek Gila, Robert Tarjan, Caleb Levy, Stephen Timmel.

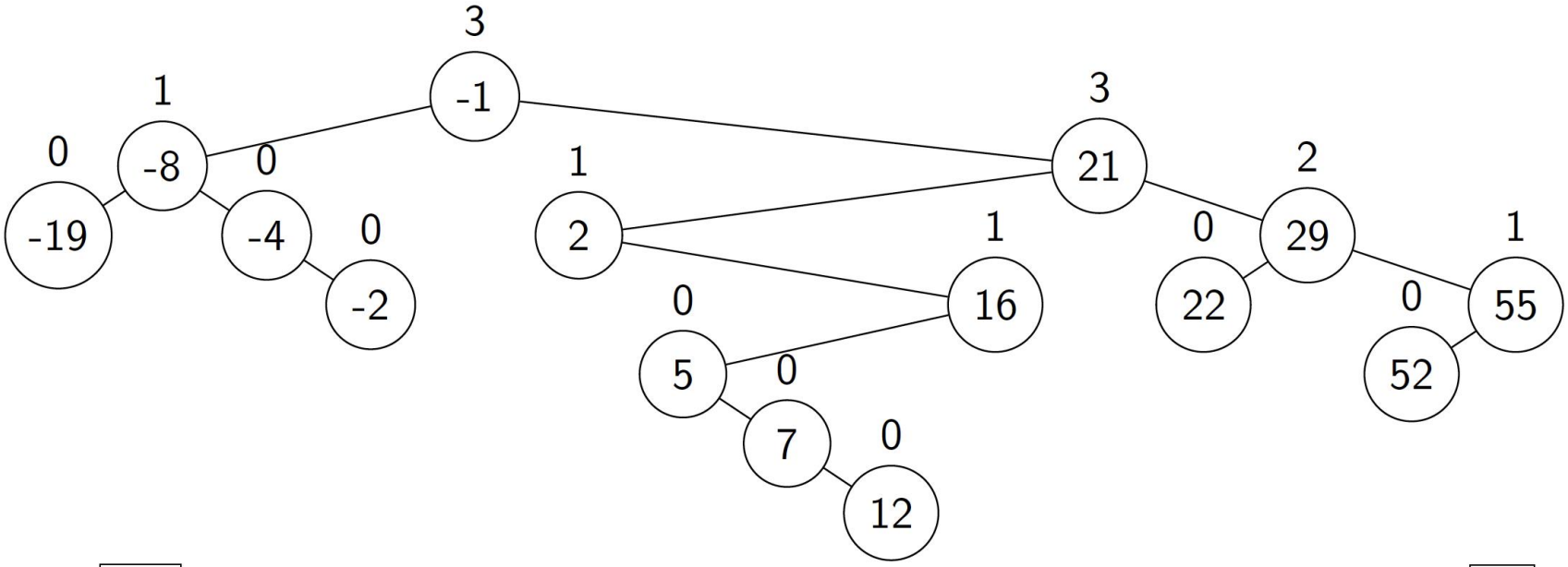
Zip Tree

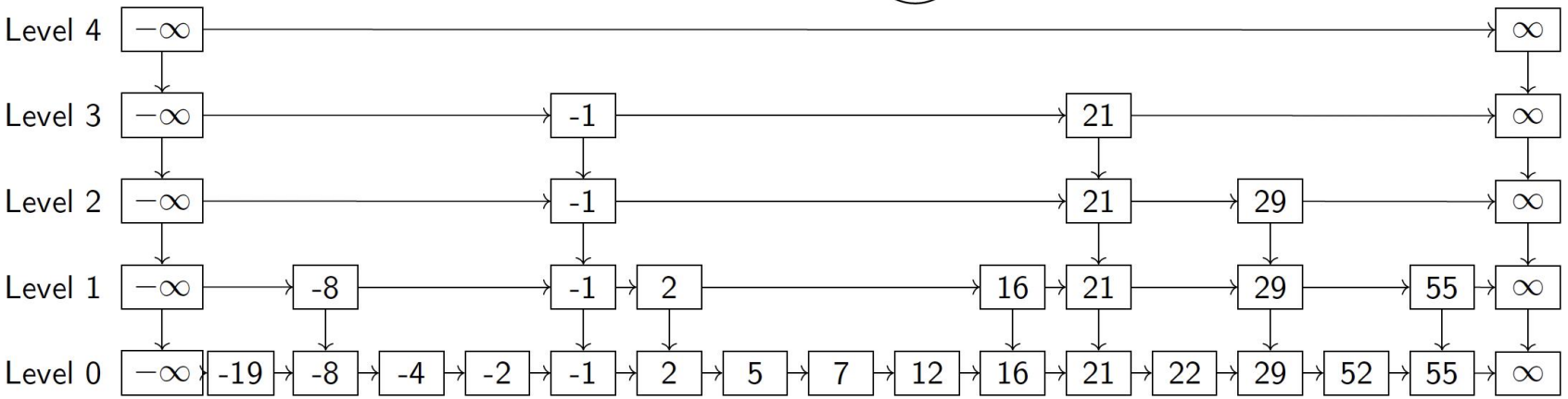
A binary search tree where each node is assigned a random **rank**, which is a geometric random variable with probability parameter p=1/2 (e.g., number of times you need to flip a fair coin until getting a "heads").
Nodes are organized so that they have the max-heap property with respect to these ranks, breaking ties by the key value (so nodes with same rank form chains of left children).



Duality between Zip Trees and Skip Lists

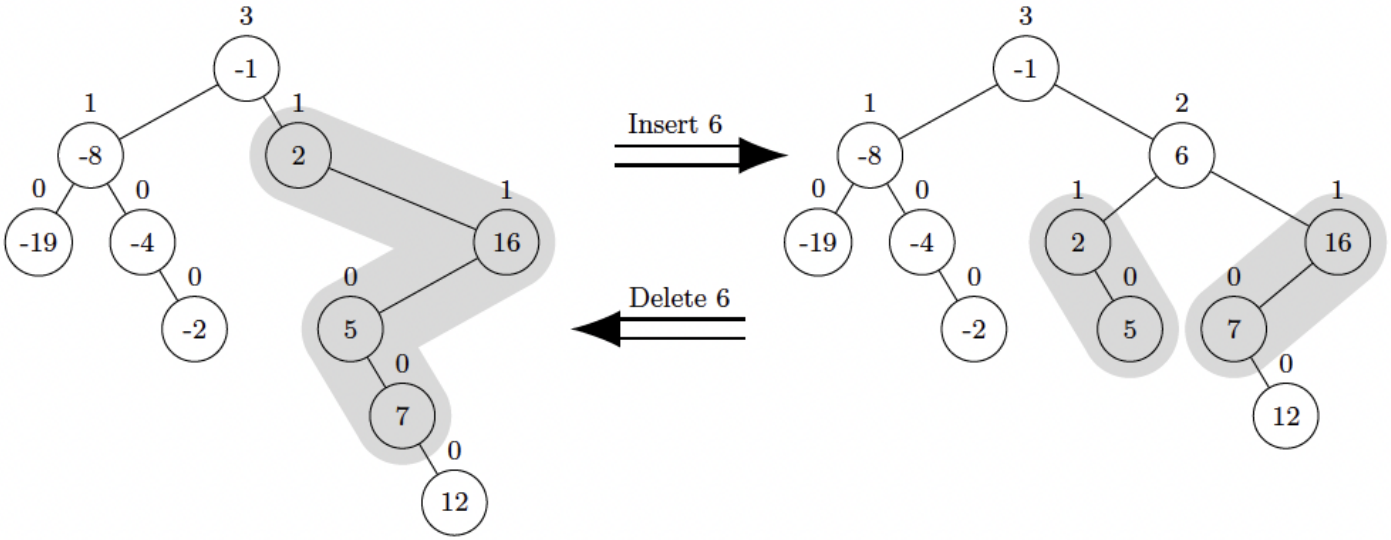
Each node is labeled with corresponding skip-list height, which is chosen randomly, i.e., its rank.





Insert Operation

- To insert a node x into a zip tree, we search for x in the tree until reaching the node y that x wil replace, namely the node y such that y.rank ≤ x.rank, with strict inequality if y.key < x.key.
- From y, we follow the rest of the search path for x, unzipping it by splitting it into a path, P, containing each node with key less than x.key and a path, Q, containing each node with key greater than x.key (recall that we assume keys are distinct).

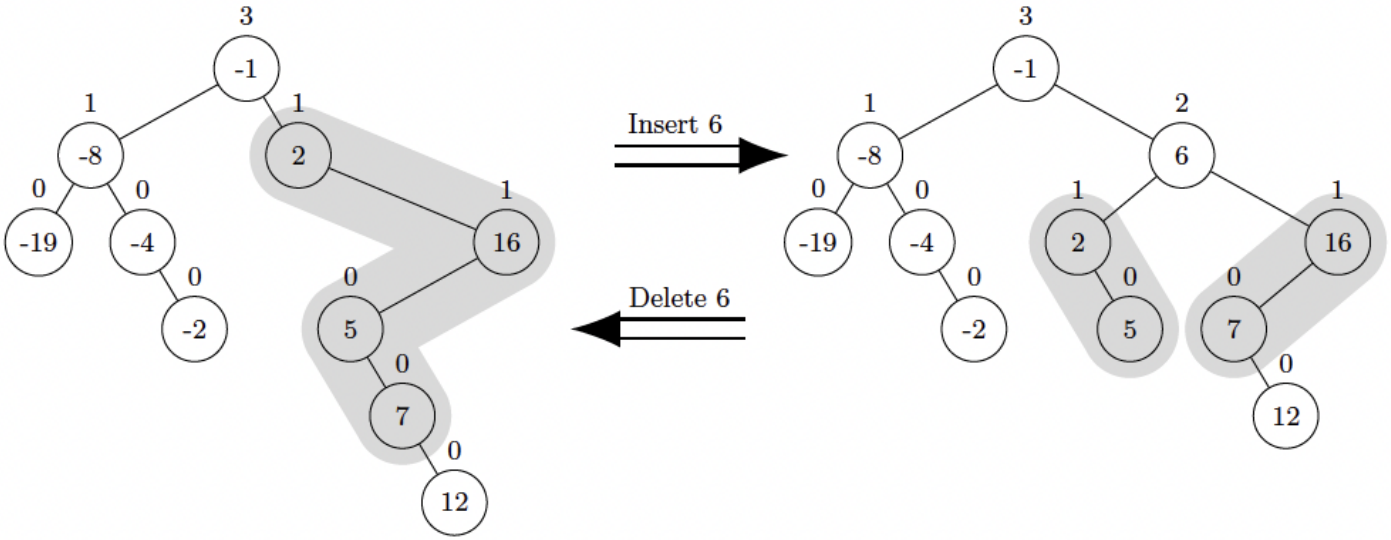


Insert Pseudo-code

```
insert(x)
    rank ← x.rank ← RandomRank
    key ← x.key
    cur ← root
    while cur ≠ null and (rank < cur.rank or (rank = cur.rank and key > cur.key)) do
        prev ← cur
        cur ← if key < cur.key then cur.left else cur.right
    if cur = root then root ← x
    else if key < prev.key then prev.left ← x
    else prev.right ← x
    if cur = null then {x.left ← x.right ← null; return}
    if key < cur.key then x.right ← cur else x.left ← cur
    prev ← x
    while cur ≠ null do
        fix ← prev
        if cur.key < key then
            repeat {prev ← cur; cur ← cur.right}
            until cur = null or cur.key > key
        else
            repeat {prev ← cur; cur ← cur.left}
            until cur = null or cur.key < key
        if fix.key > key or (fix = x and prev.key > key) then
            fix.left ← cur
        else
            fix.right ← cur
```

Delete Operation

- To delete a node x into a zip tree, we perform the reverse zip operation, where we do a search to find x and let P and Q be the right spine of the left subtree of x and the left spine of the right subtree of x, respectively.
- Then we zip P and Q to form a single path R by merging them from top to bottom in non-increasing rank order, breaking a tie in favor of the smaller key.



Delete Pseudo-code

```
delete(x)
    key ← x.key
    cur ← root
    while key ≠ cur.key do
        prev ← cur
        cur ← if key < cur.key then cur.left else cur.right
    left ← cur.left; right ← cur.right

    if left = null then cur ← right
    else if right = null then cur ← left
    else if left.rank ≥ right.rank then cur ← left
    else cur ← right

    if root = x then root ← cur
    else if key < prev.key then prev.left ← cur
    else prev.right ← cur

    while left ≠ null and right ≠ null do
        if left.rank ≥ right.rank then
            repeat {prev ← left; left ← left.right}
            until left = null or left.rank < right.rank
            prev.right ← right
        else
            repeat {prev ← right; right ← right.left}
            until right = null or left.rank ≥ right.rank
            prev.left ← left
```

Zip-zip Trees

- What if rank was a tuple, (r_1, r_2)?
  - Let r_1 be geometrically distributed
  - Let r_2 be uniformly distributed from $[1, \log^c n]$
- Compare ranks lexicographically
- ☐ Metadata size $O(\log \log n)$? - $O(\log \log n) + O(c \log \log n)$
- Hope: fewer collisions, better depth?

Zip Tree and Zip-zip Tree Examples

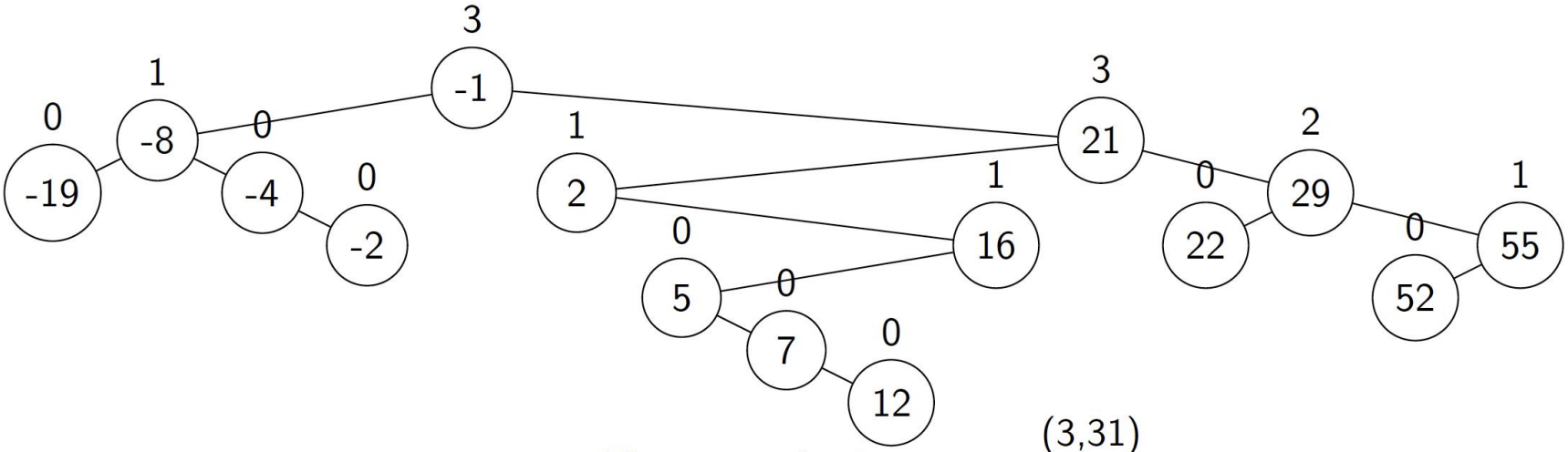


Figure 2: A zip tree

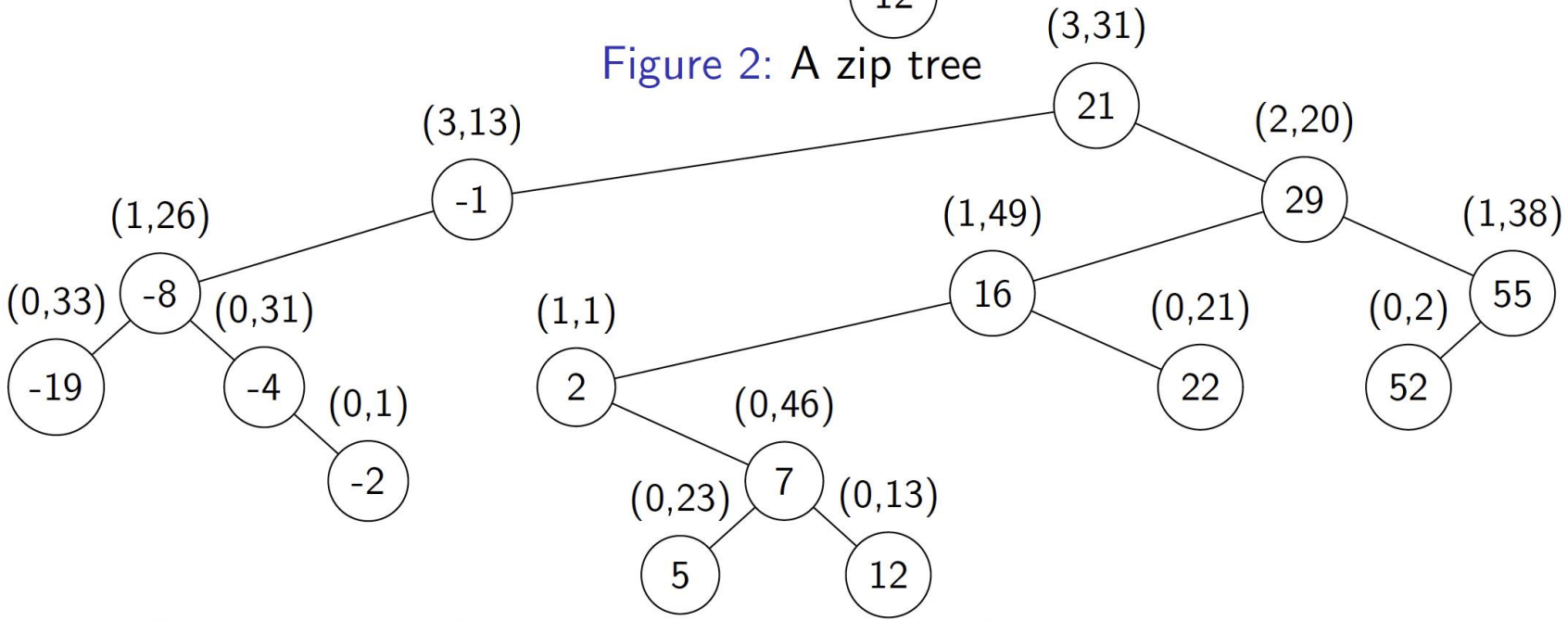
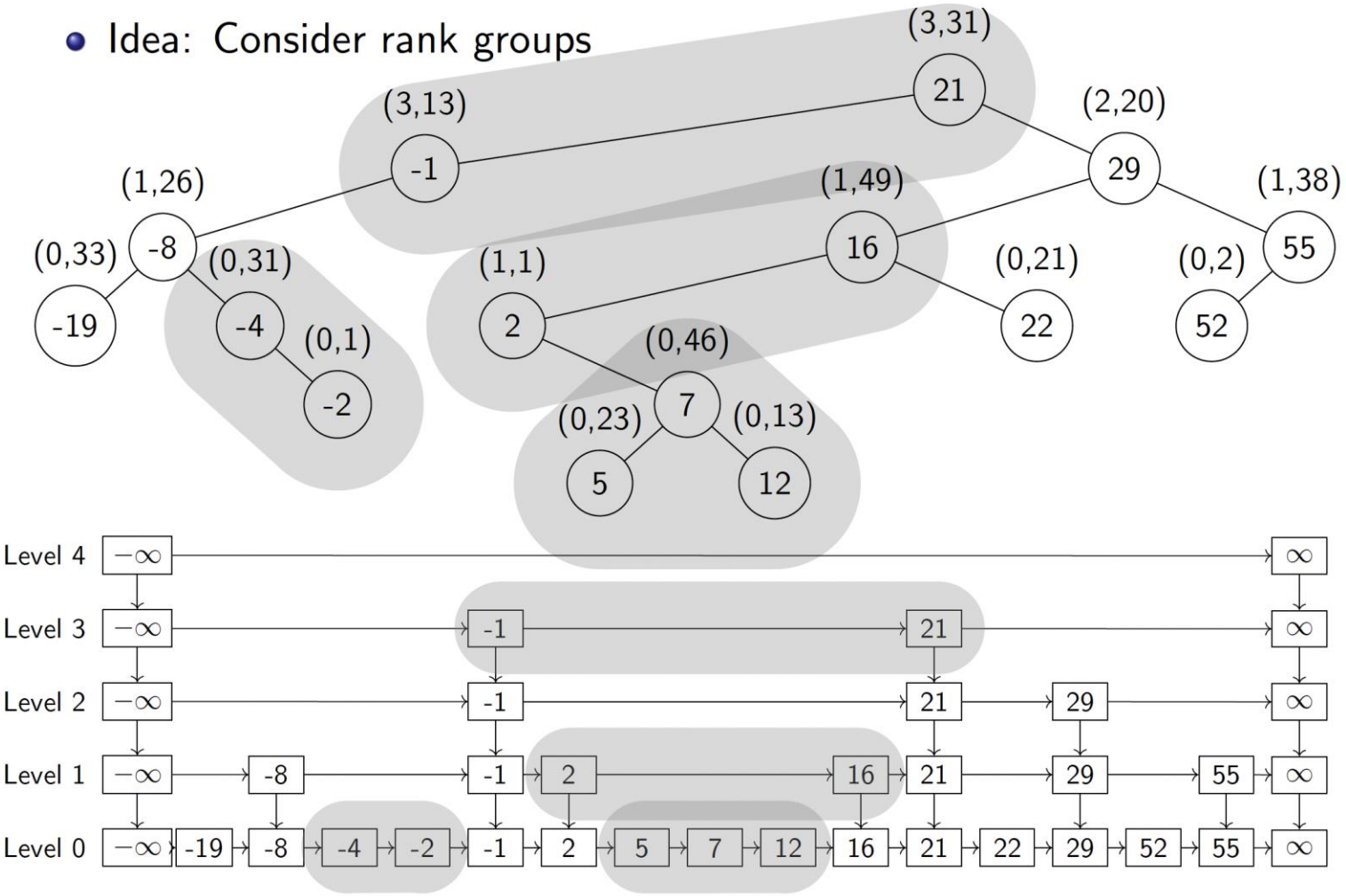


Figure 3: A random zip-zip tree generated from the above zip tree

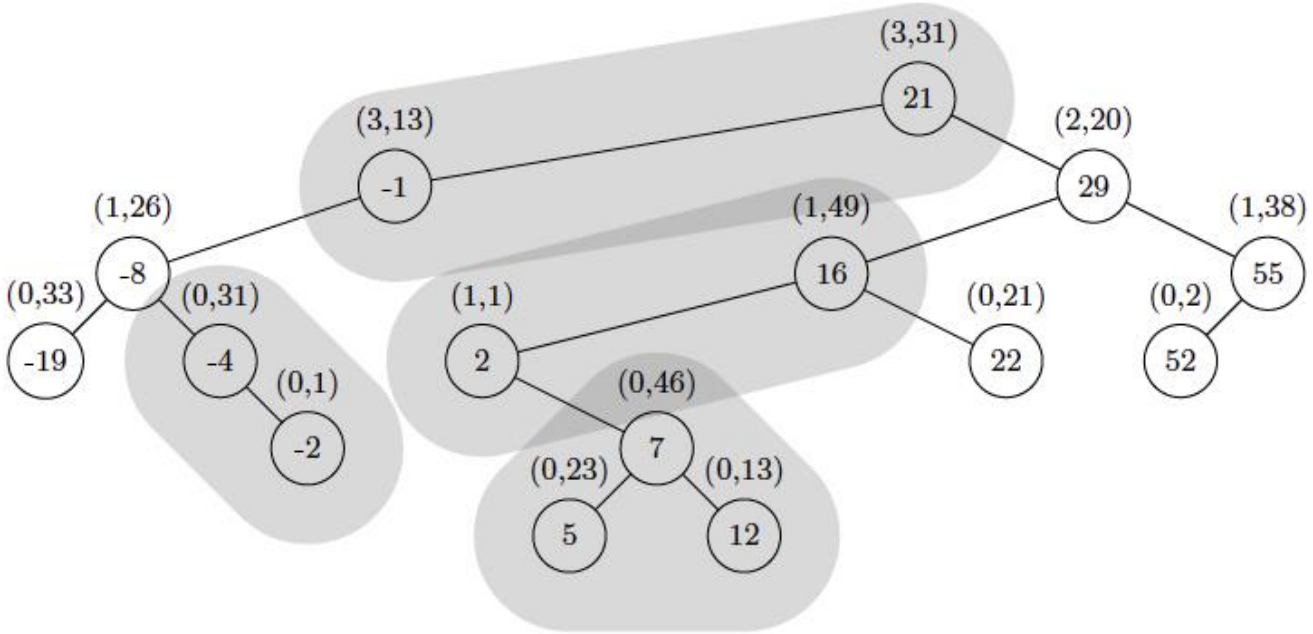
Zip-zip Trees are Dual to Skip Lists

- Each rank group (nodes with same first coordinate) correspond to nodes in a skip list group on the same level.



Insert and Delete Operations

- Insert and delete operations are the same as in a zip tree, except that instead of just using the rank random value, we use the (r_1, r_2) random pairs to determine node heights.
- That is, if there would be a tie with the r_1 value (i.e., rank in a zip tree), we break the tie using r_2.



Performance

- Insert, delete, and search run in O(log n) time with high probability.
- The expected depth of a node in an n-node zip-zip tree is at most 1.3863 log n - 1 + o(1).
- Average key depth and tree height for zip trees and zip-zip trees: