AVL Trees

Operations

get(k)
smallest()
largest()
successor(k)
predecessor(k)

O(height of tree) = O(log n)

put(k,d)
remove(k)

O(height of tree) = O(log n)

AVL Trees

Operations

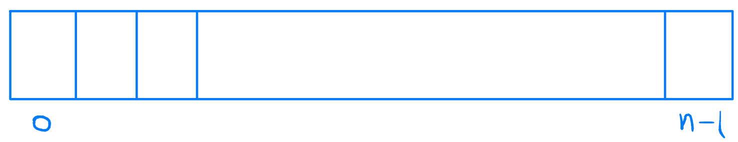
get(k)
smallest()
largest()
successor(k)
predecessor(k)

O(height of tree) = O(log n)

put(k,d)
remove(k)

O(height of tree) = O(log n)

Arrays



Operations

get(k)
smallest()
largest()
successor(k)
predecessor(k)
put(k,d)
remove(k)

O(n)

AVL Trees

Operations

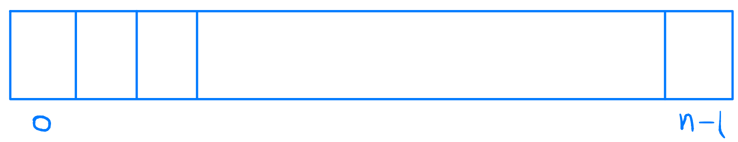
get(k)
smallest()
largest()
successor(k)
predecessor(k)

O(height of tree) = O(log n)

put(k,d)
remove(k)

O(height of tree) = O(log n)

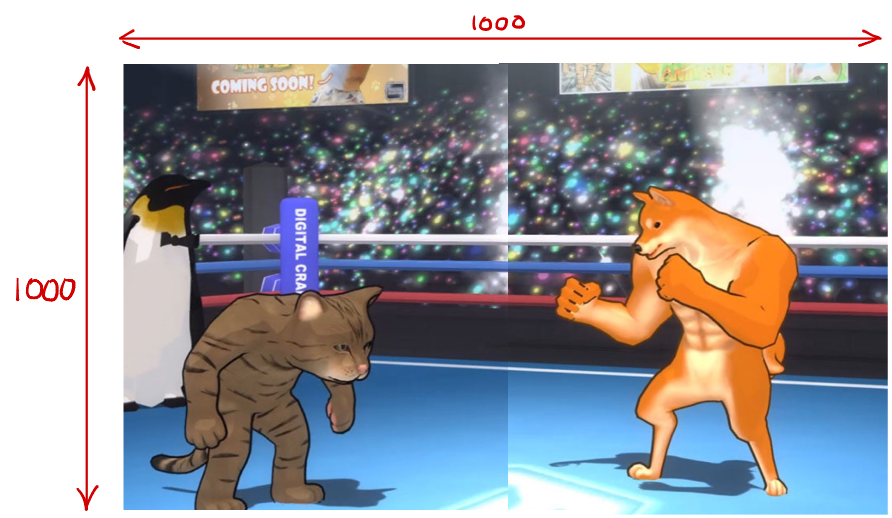
Arrays



Operations

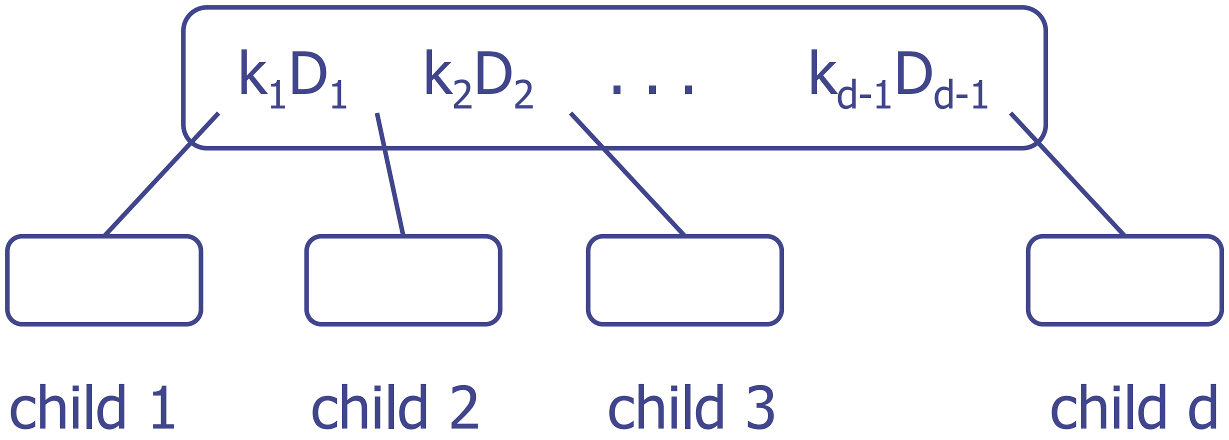
get(k)
smallest()
largest()
successor(k)
predecessor(k)
put(k,d)
remove(k)

O(n)



Multi-Way Search Tree

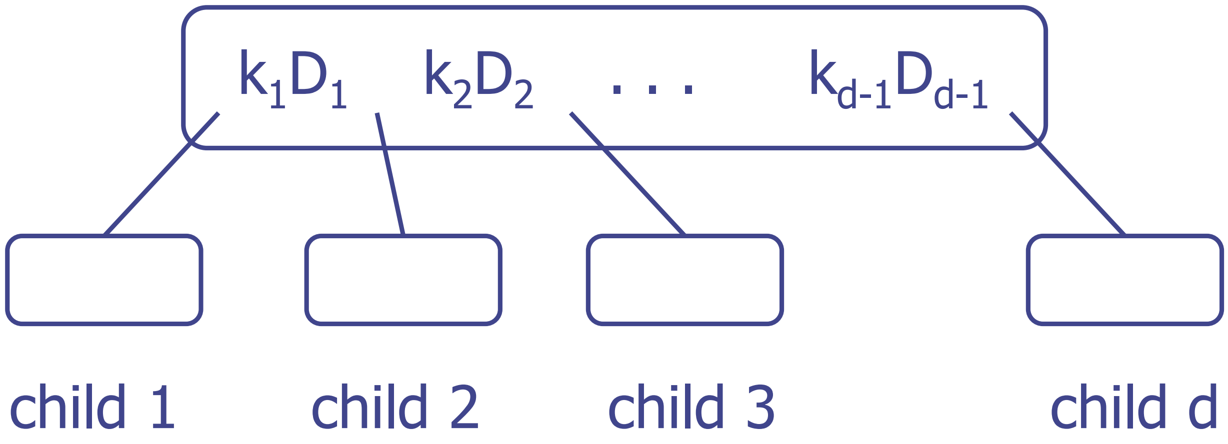
A multi-way search tree is an ordered tree such that



Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

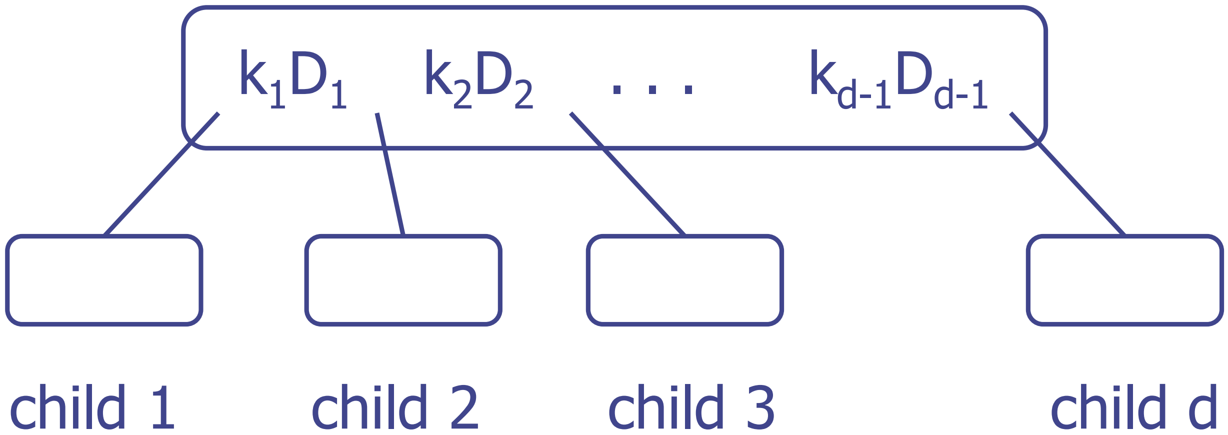
- Each internal node has at least two and at most d children and stores $\leq d-1$ data items (k_i, D_i)



Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

- Each internal node has at least two and at most d children and stores $\leq d-1$ data items (k_i, D_i)



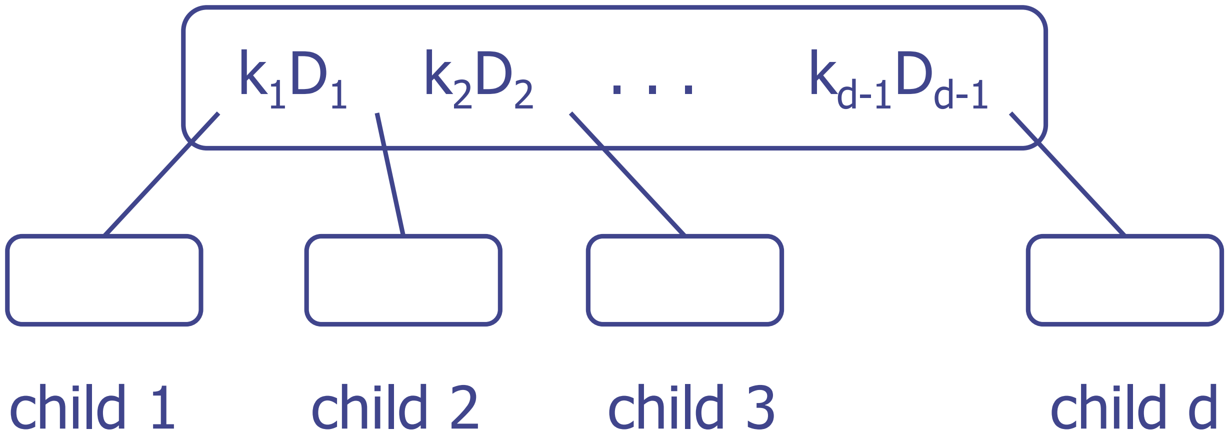
d is the degree or order of the tree

Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

- Each internal node has at least two and at most d children and stores $\leq d - 1$ data items (k_i, D_i)

Rule: Number of children = 1 + number of data items in a node

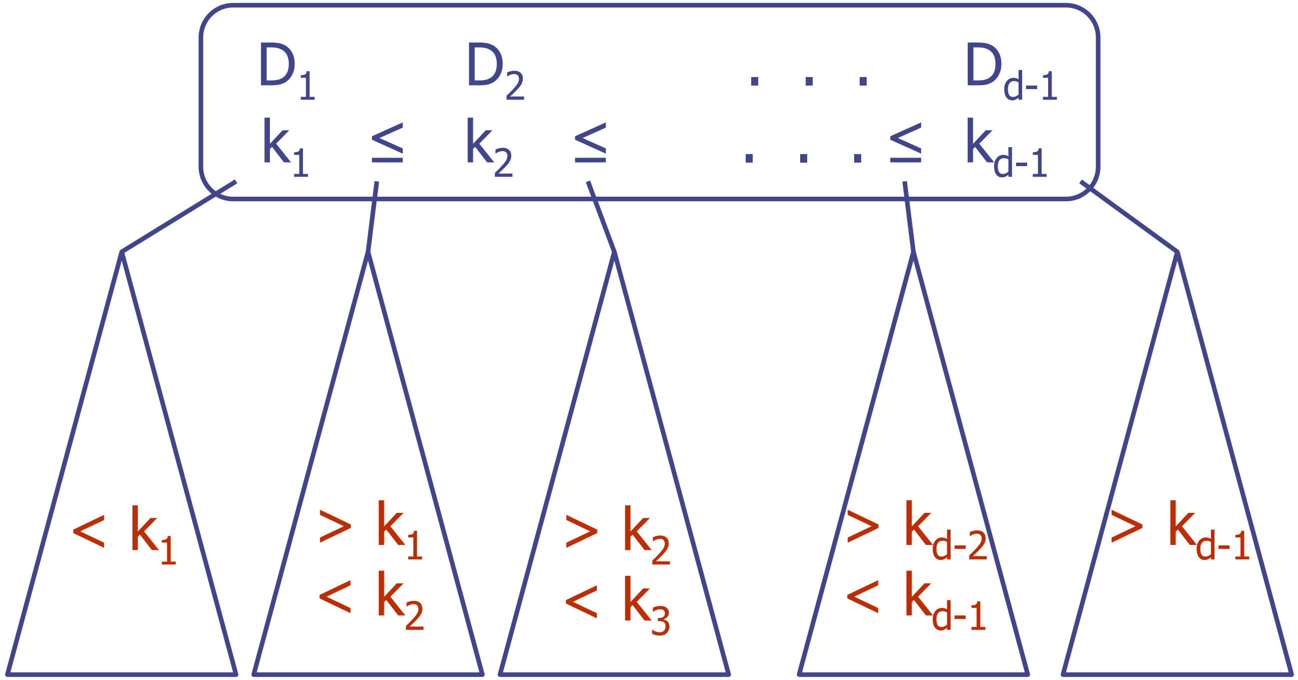


d is the degree or order of the tree

Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

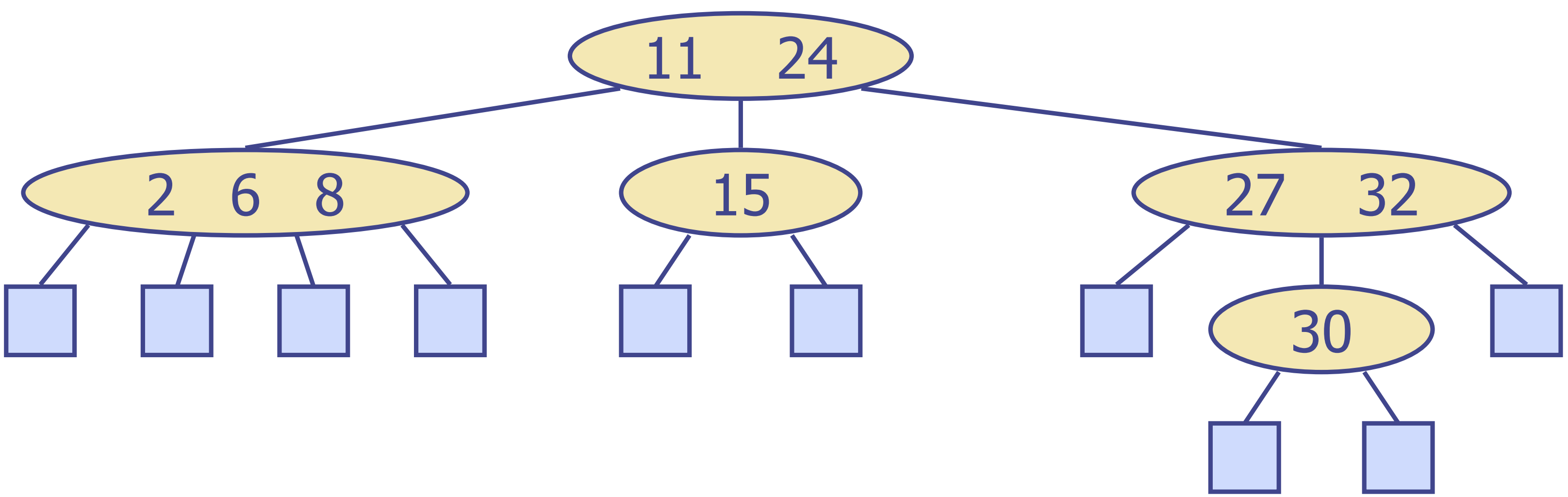
- Each internal node has at least two and at most d children and stores $d-1$ data items (k_i, D_i)
- An internal node storing keys $k_1 \le k_2 \le \ldots \le k_{d-1}$ has d children $v_1\, v_2 \ldots v_d$ such that



Multi-Way Search Tree

A multi-way search tree is an ordered tree such that

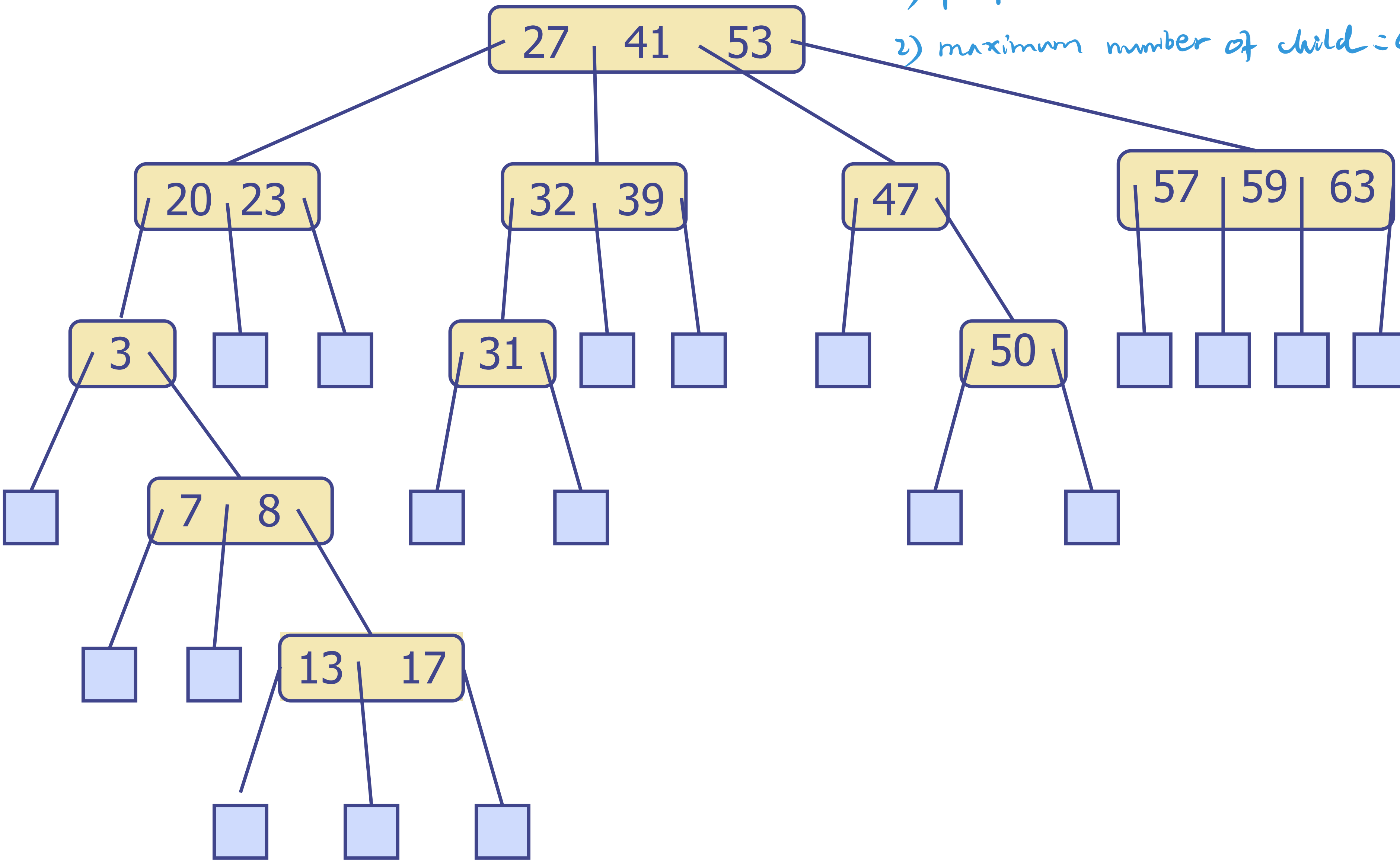
- Each internal node has at least two and at most d children and stores $d-1$ data items (k_i, D_i)
- An internal node storing keys $k_1 \le k_2 \le \ldots \le k_{d-1}$ has d children $v_1\, v_2\, \ldots\, v_d$ such that
- The leaves store no items and serve as placeholders



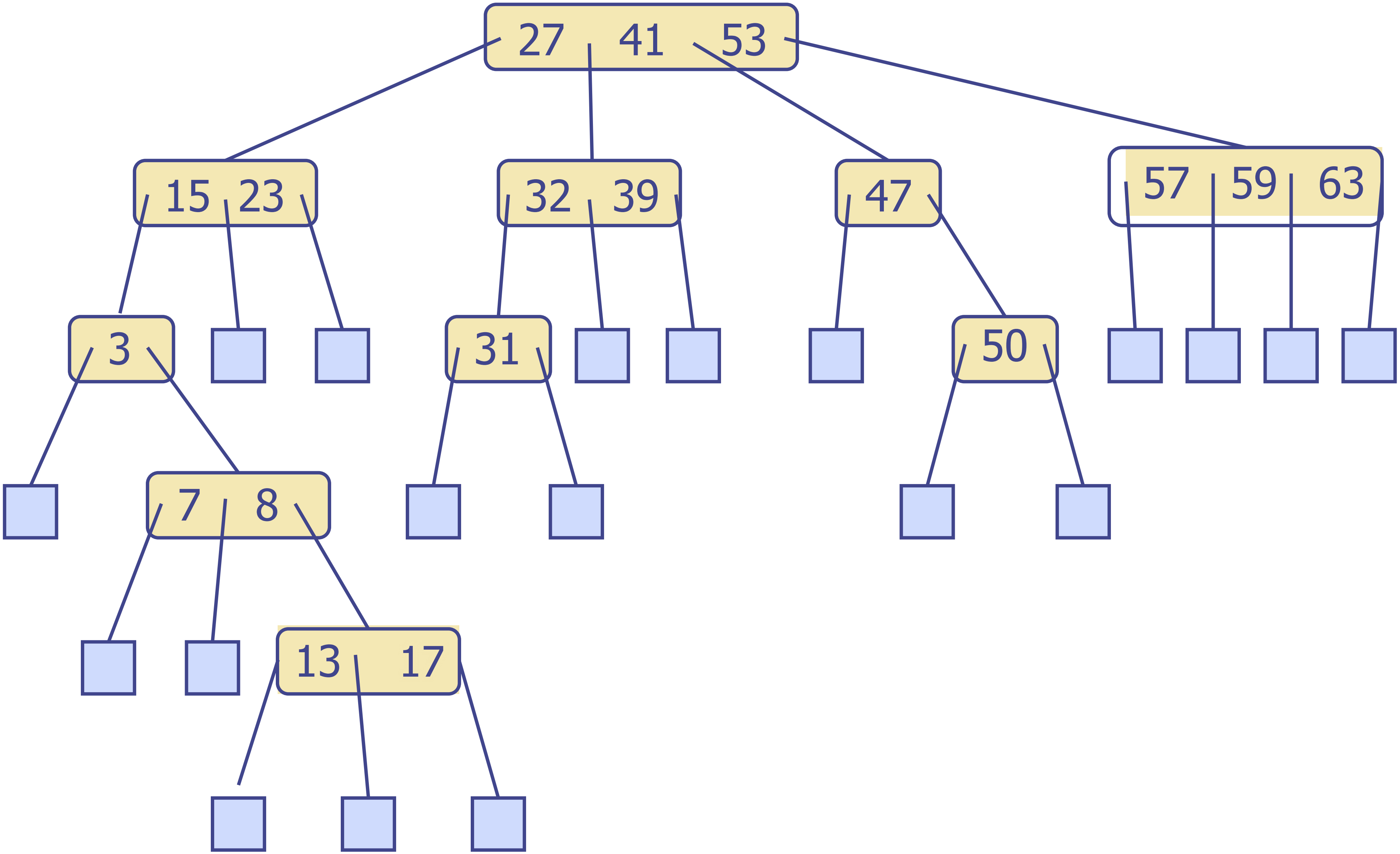
Multi-Way Search Tree of Degree 4?

1) proper value

2) maximum number of child = 4.



Multi-Way Search Tree of Degree 4?



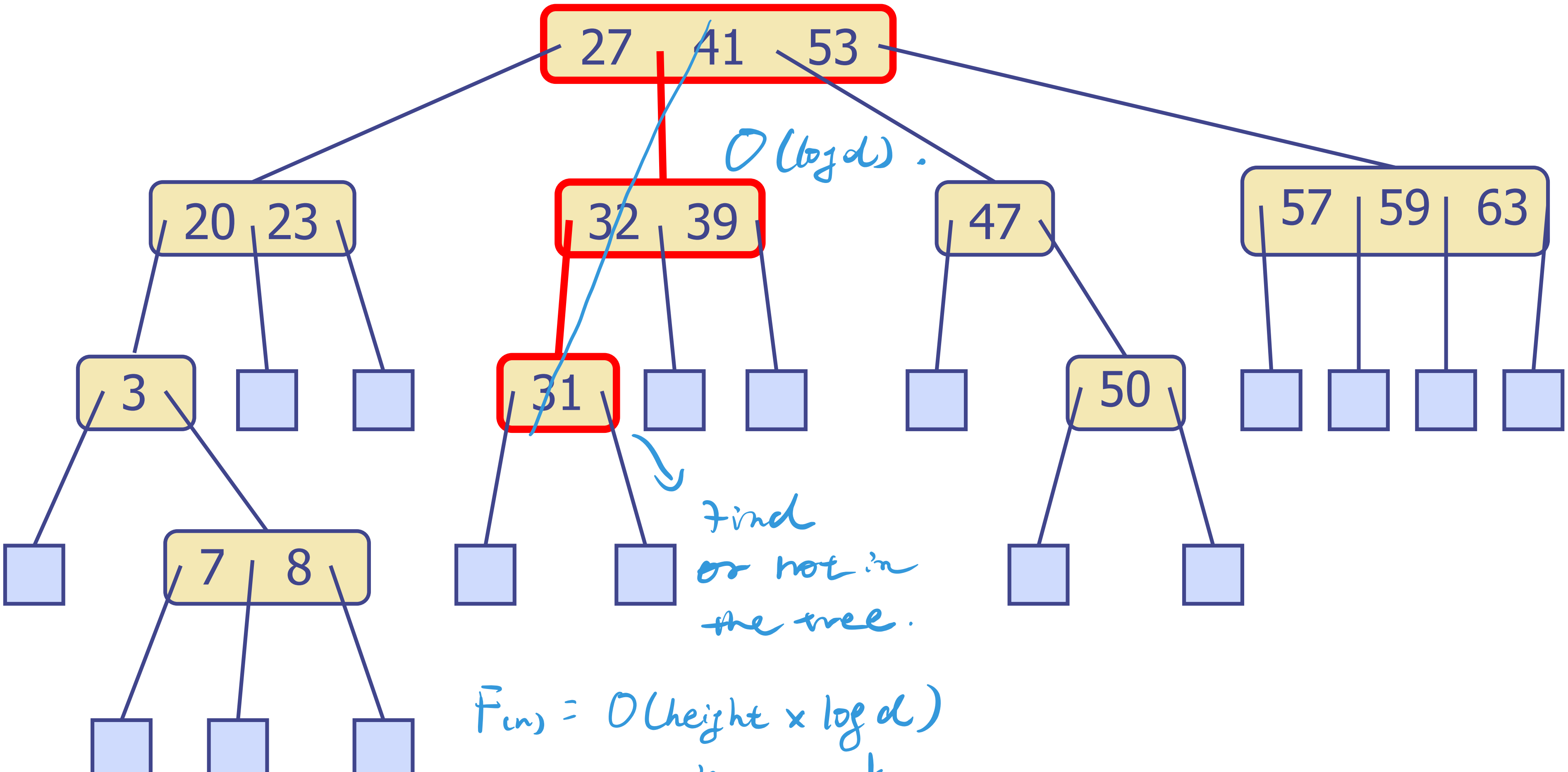
Ordered Dictionary (Map) ADT

- get (k): record with key k
- put (k,data): add record (k,data)
- remove (k): delete record with key k
- smallest(): record with smallest key
- largest(): record with largest key
- predecessor(k): record with largest key less than k
- successor(k): record with smallest key greater than k

Get Operation

- Similar to search in a binary search tree
- Example: search for 31

binary search $O(\log d)$



$O(\log d)$.

find or not in the tree.

$F(n) = O(\underbrace{\text{height}}_{\text{number of nodes}} \times \underbrace{\log d}_{\text{time per node}})$

Multi-Way Searching

Algorithm get(r,k)

In: Root r of a multiway search tree, key k
Out: data for key k or null if k not in tree

if r is a leaf **then return** null

else {

Use binary search to find the index i such that either

- r.keys[i] = k, or
- r.keys[i] < k < r.keys[i+1]

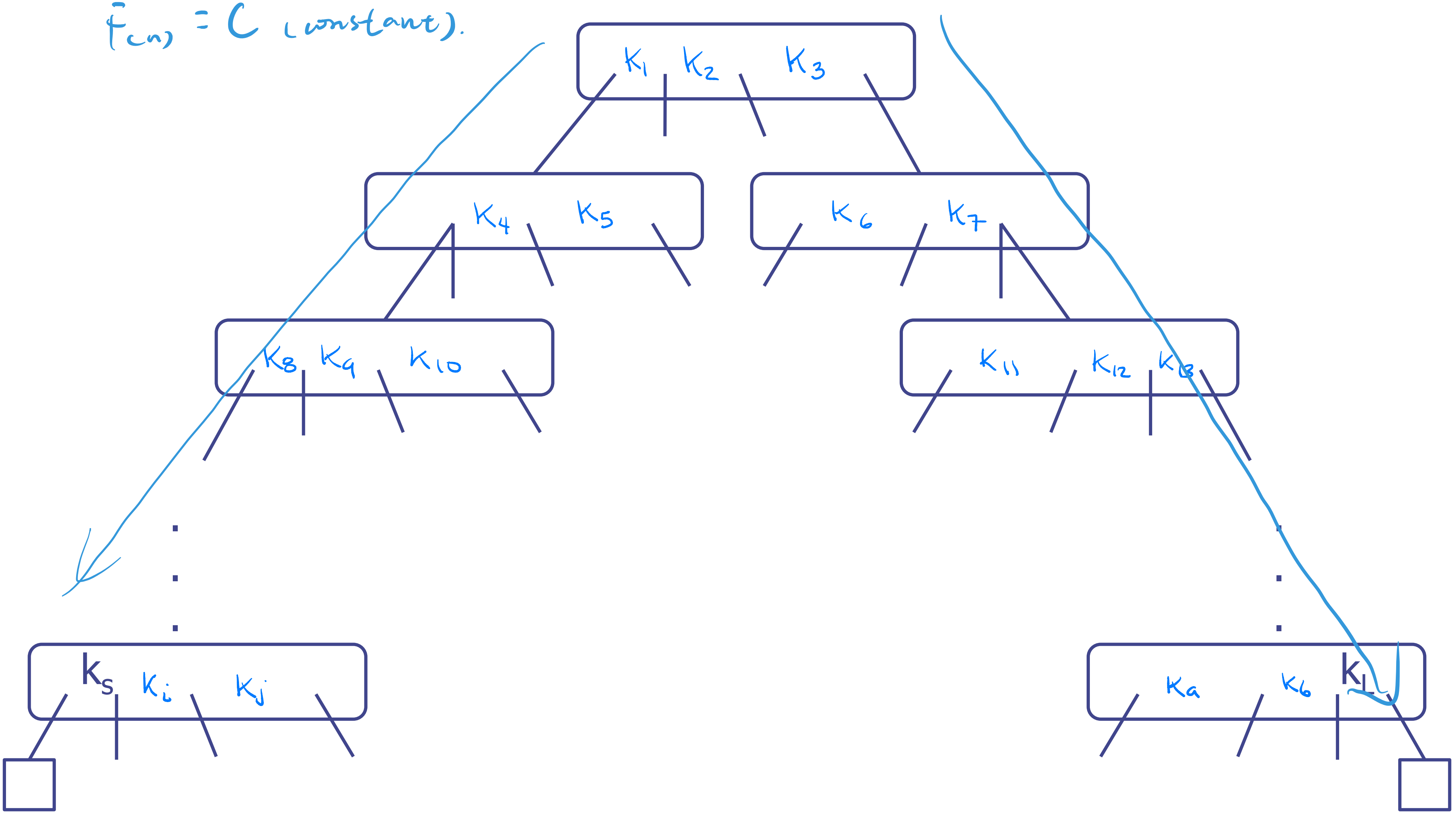
if k = r.keys[i] **then return** r.data[i]

else return get(r.child[i],k)

}

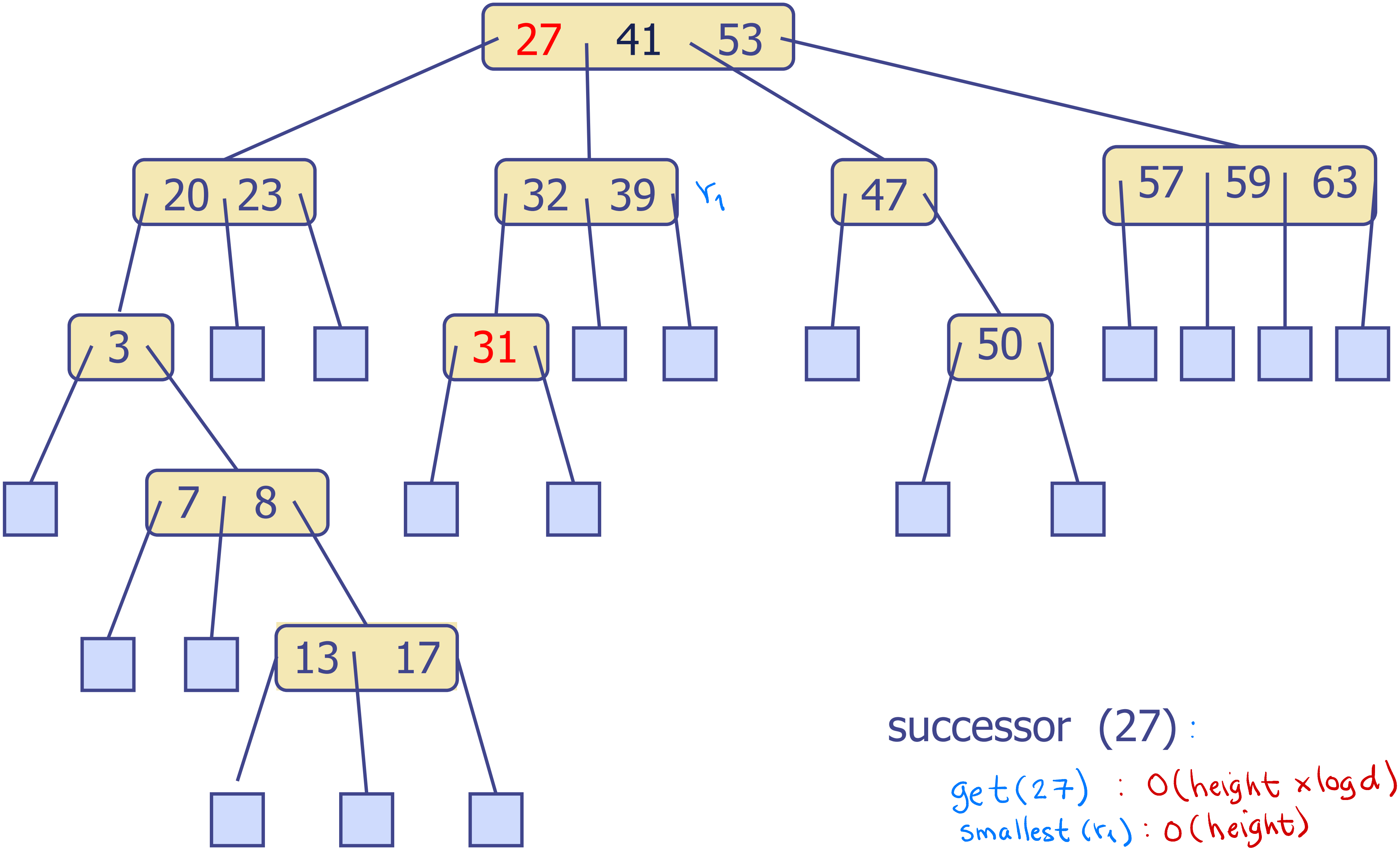
Smallest and Largest Operations

$f(n) = C$ (constant).



Successor Operation

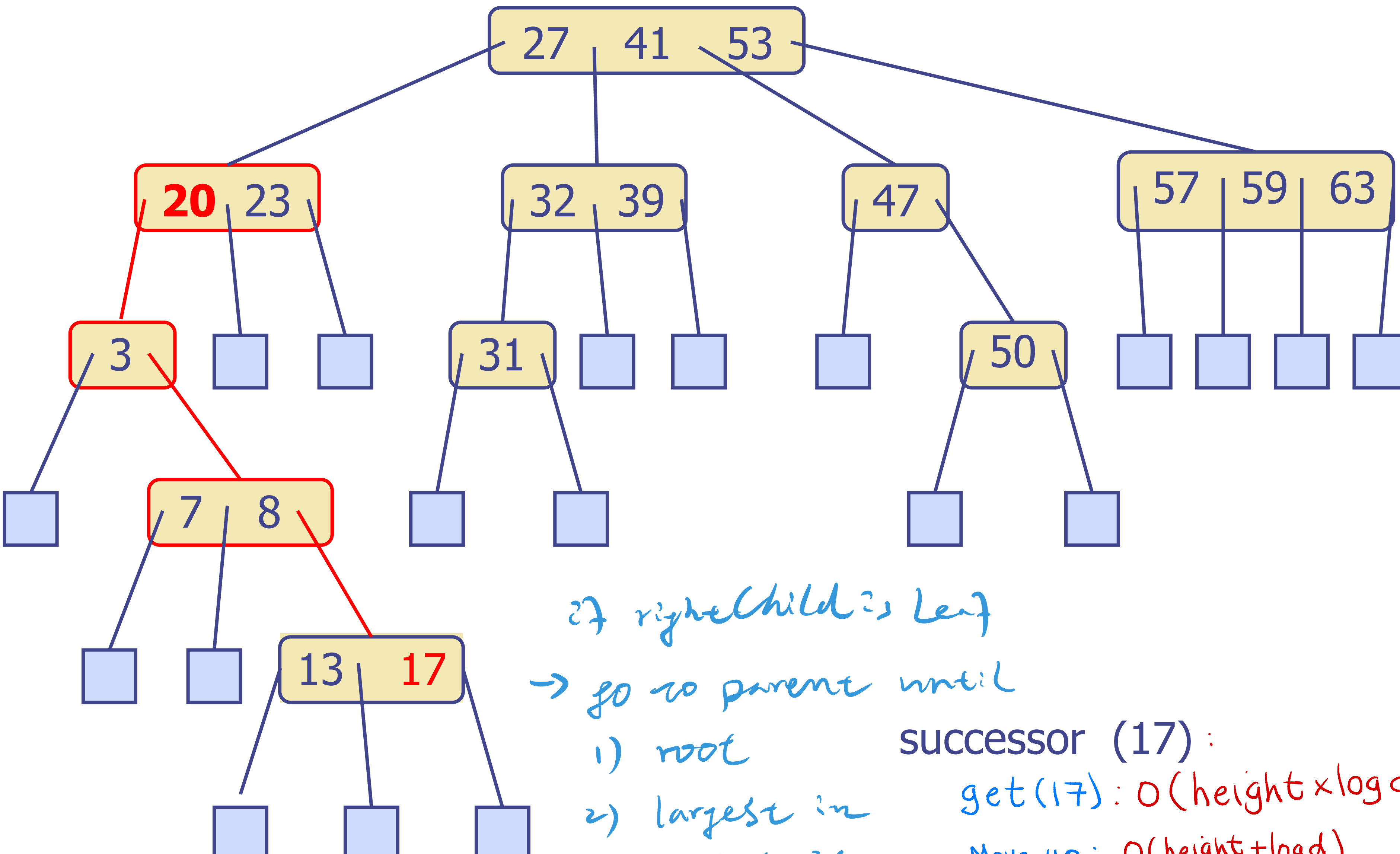
}
get(r, key)
smallest in the subtree.



successor (27) :

get(27) : $O(\text{height} \times \log d)$
smallest (r_1) : $O(\text{height})$
$= O(\text{height} \times \log d)$.

Successor Operation



if rightChild is leaf

→ go to parent until

1) root

2) largest in parent is larger than value

⇒ .....

successor (17):

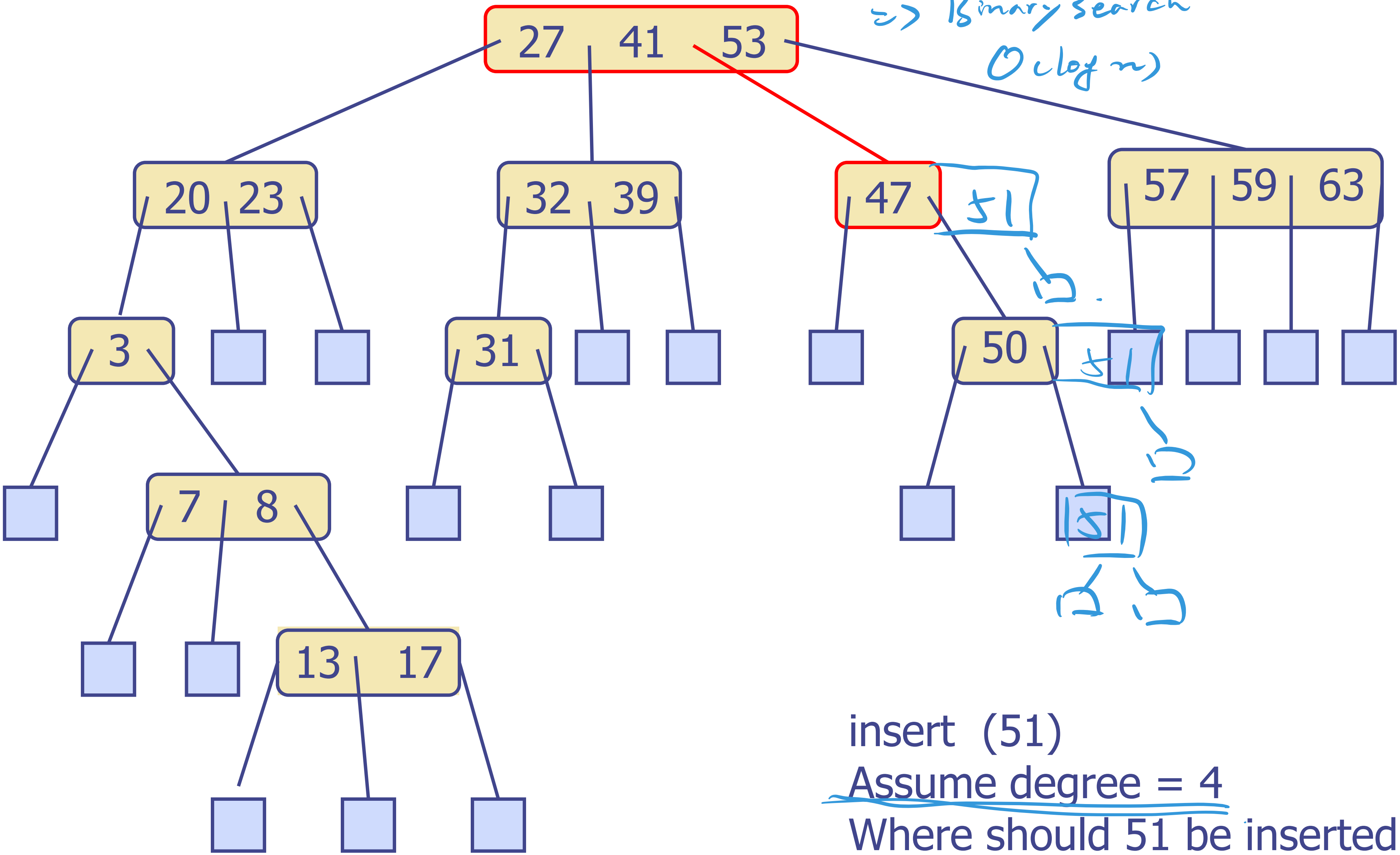
get(17): $O(\text{height} \times \log d)$

Move up: $O(\text{height} + \log d)$

$O(\text{height} \times \log d)$.

Put Operation

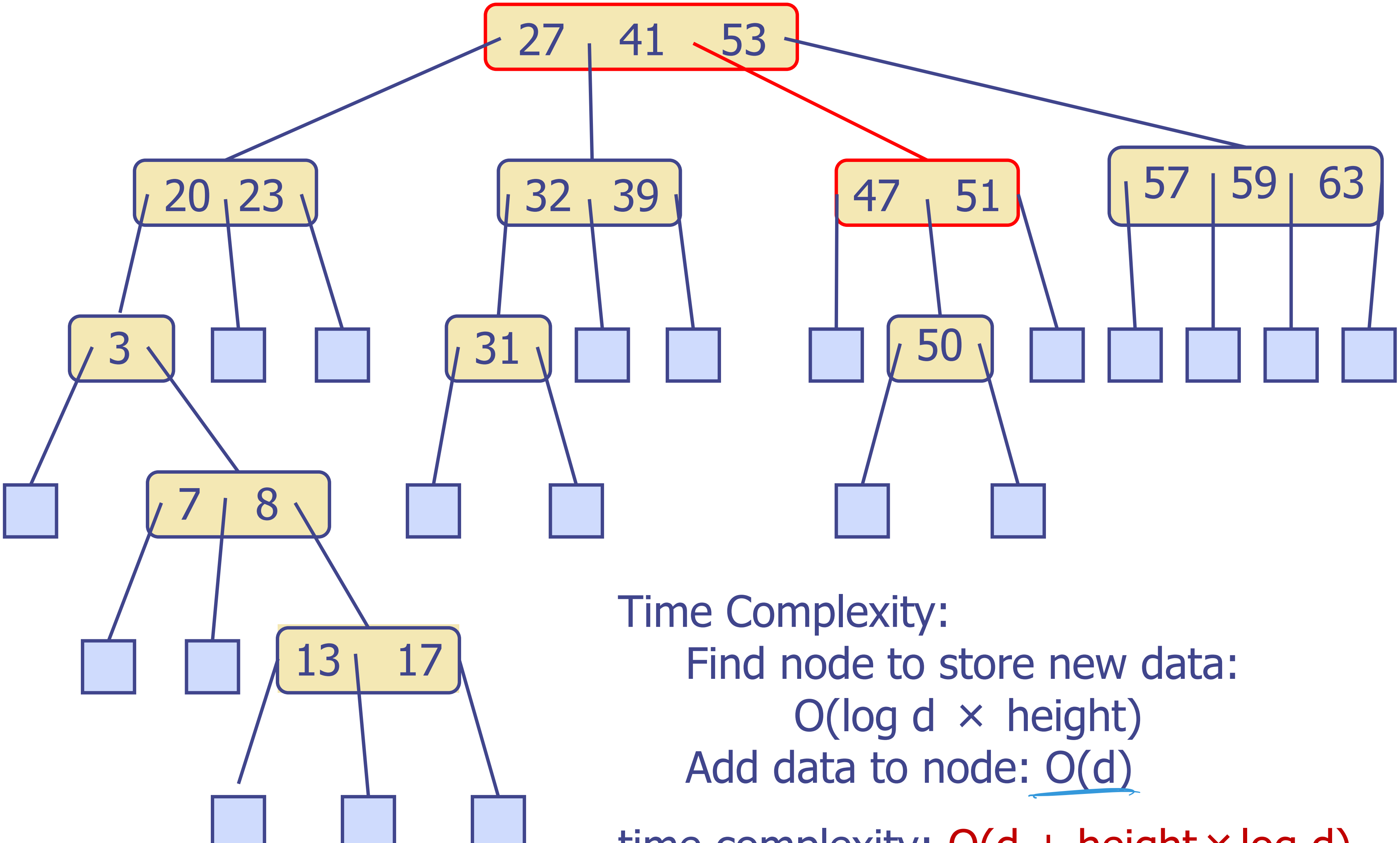
Find the place to put
=> Binary search
O(log n)



insert (51)
Assume degree = 4
Where should 51 be inserted?

it can be inserted in three positions.

Put Operation



Time Complexity:

Find node to store new data:

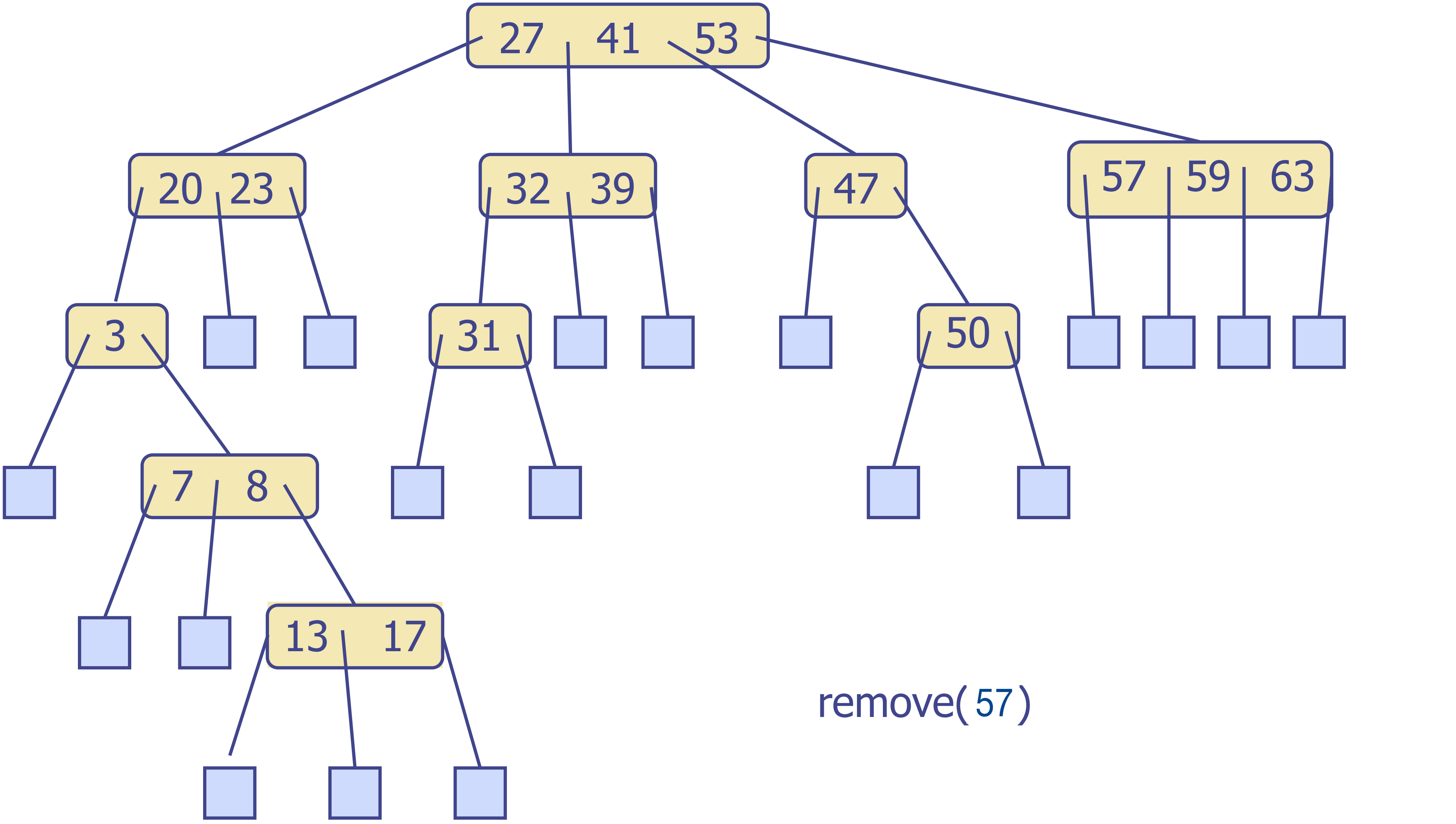
O(log d × height)

Add data to node: O(d)

time complexity: O(d + height × log d)

O(height × log d).

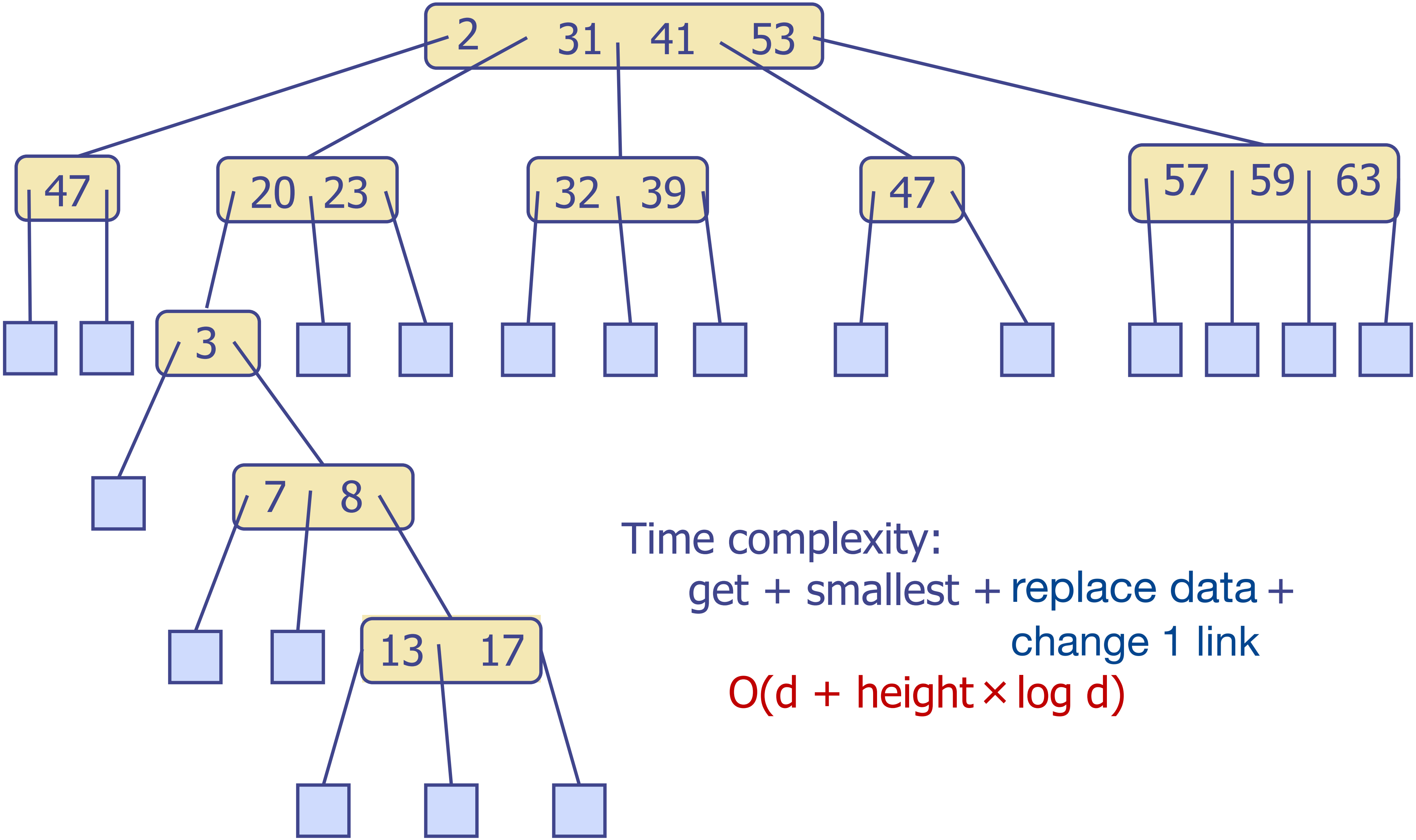
Remove Operation



remove(57)

Remove Operation

remove(z)



Time complexity:
get + smallest + replace data +
change 1 link
O(d + height × log d)

Ordered Dictionary Operations on a Multiway Search Tree of Degree d

smallest	O(height)
largest	O(height)
get	$O(\text{height} \times \log d)$
successor	$O(\text{height} \times \log d)$
predecessor	$O(\text{height} \times \log d)$
put	$O(d + \text{height} \times \log d)$
remove	$O(d + \text{height} \times \log d)$

=> build a short tree
=> a balance tree