2-4 Trees

Goal

Keep sorted lists subject to the following operations:

find(x,L)

insert(x,L)

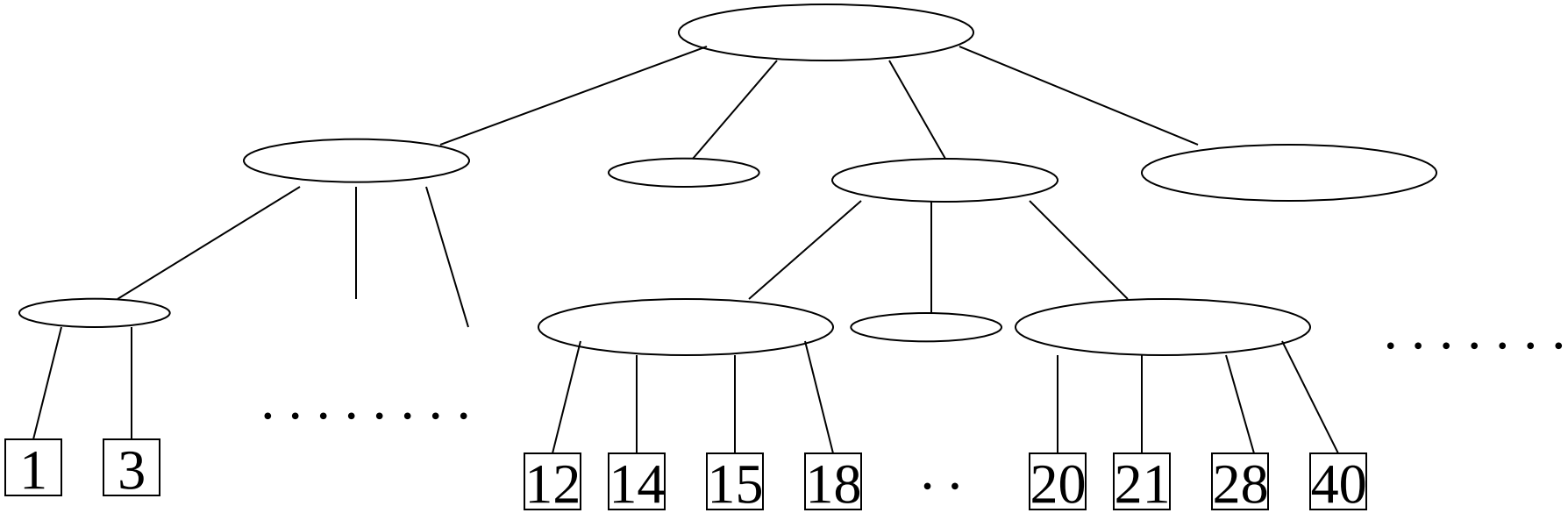
delete(x,L)

catenate(L1,L2) : Assumes that all items in L2 are greater than all items in L1.

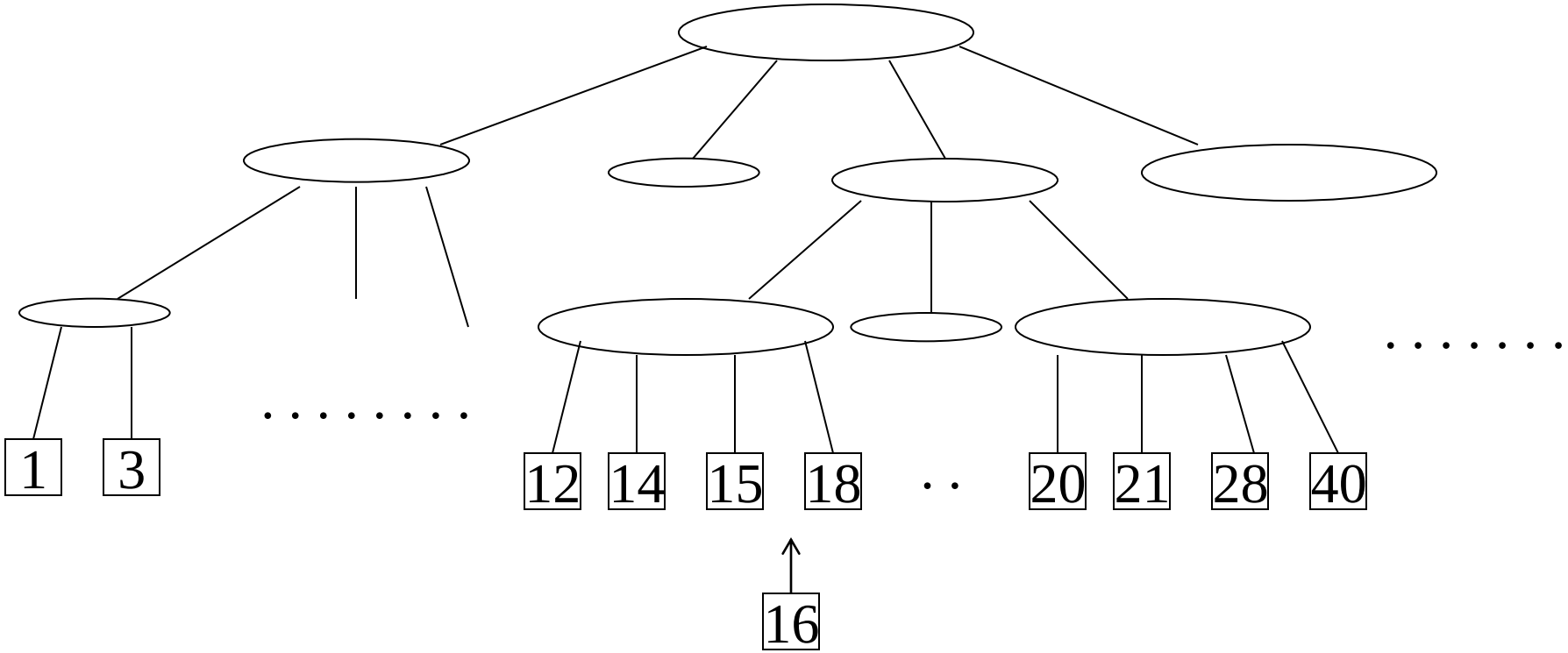
split(x,L) : returns two lists one with all item less than or equal to x and the other with all items greater than x.

2-4 trees

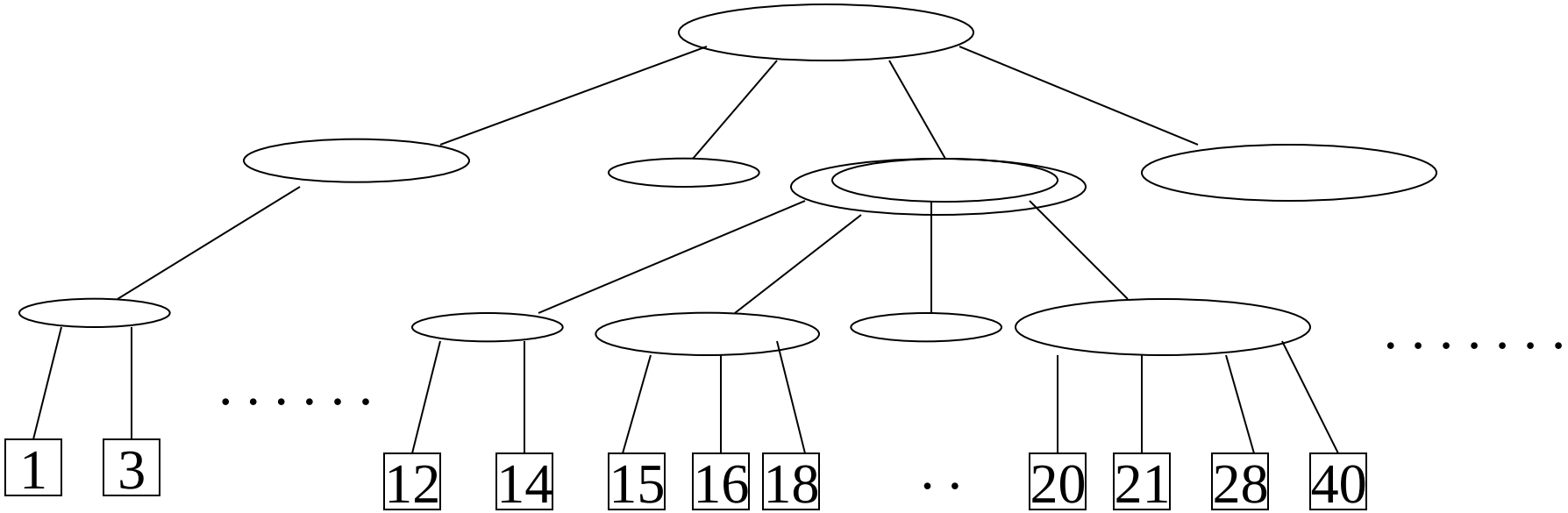
- Items are at the leaves.
- All leaves are at the same distance from the root.
- Every internal node has 2, 3, or 4 children.



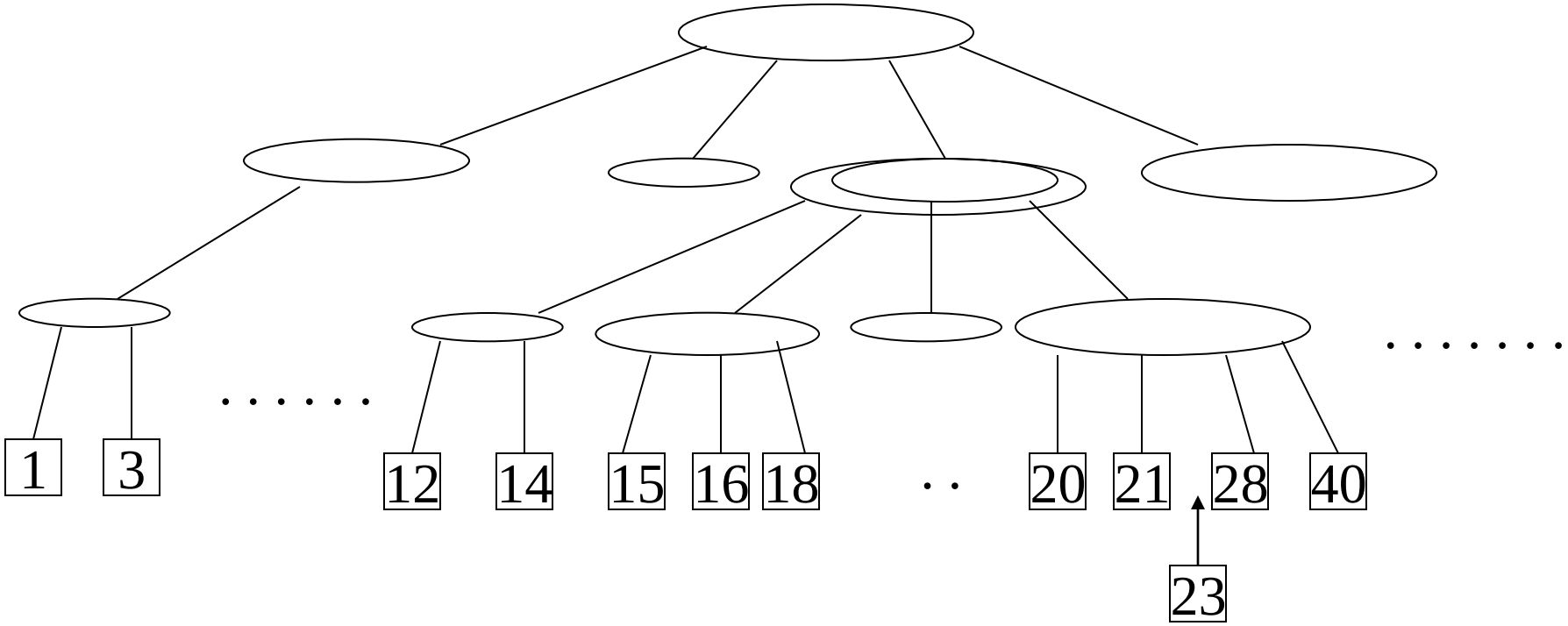
Insert



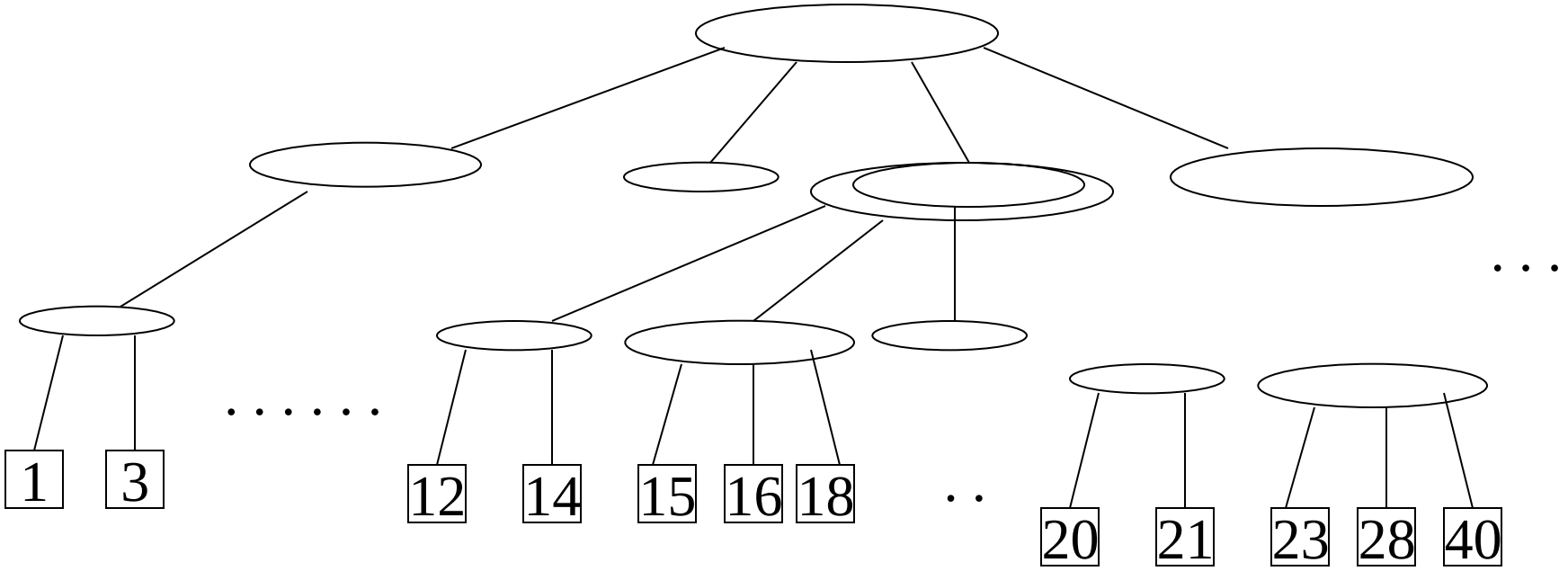
Insert (cont)



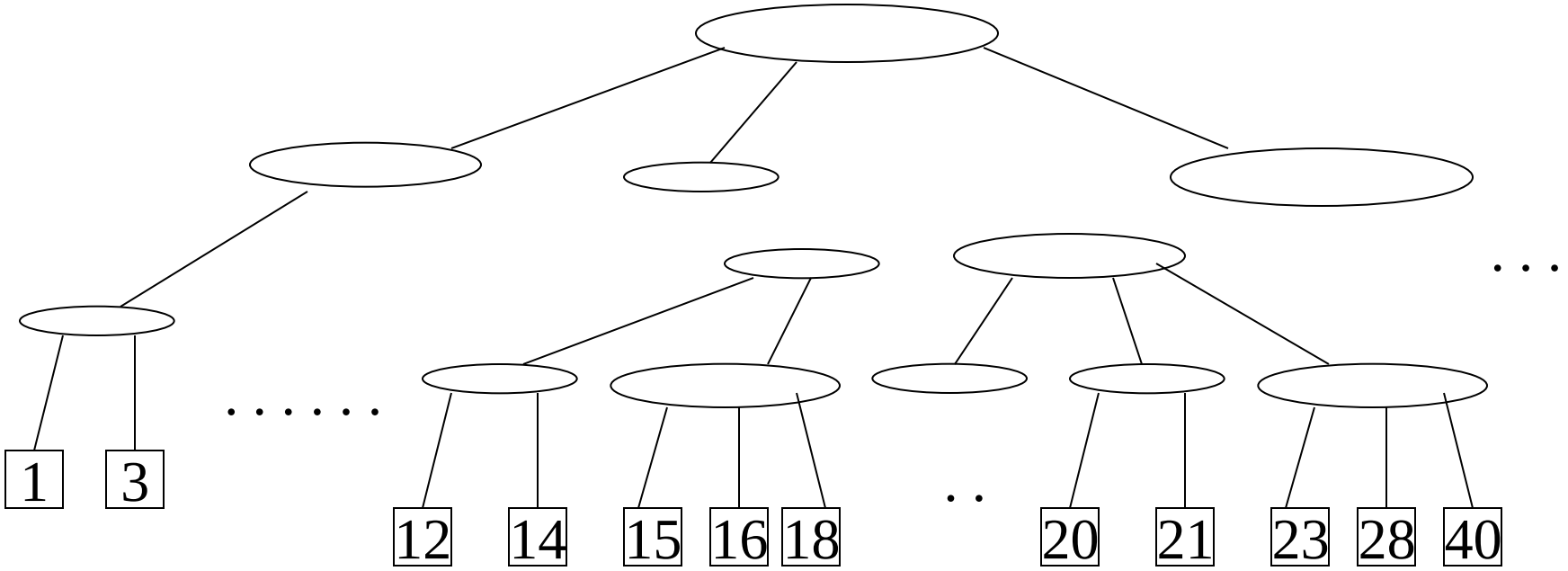
Insert (cont)



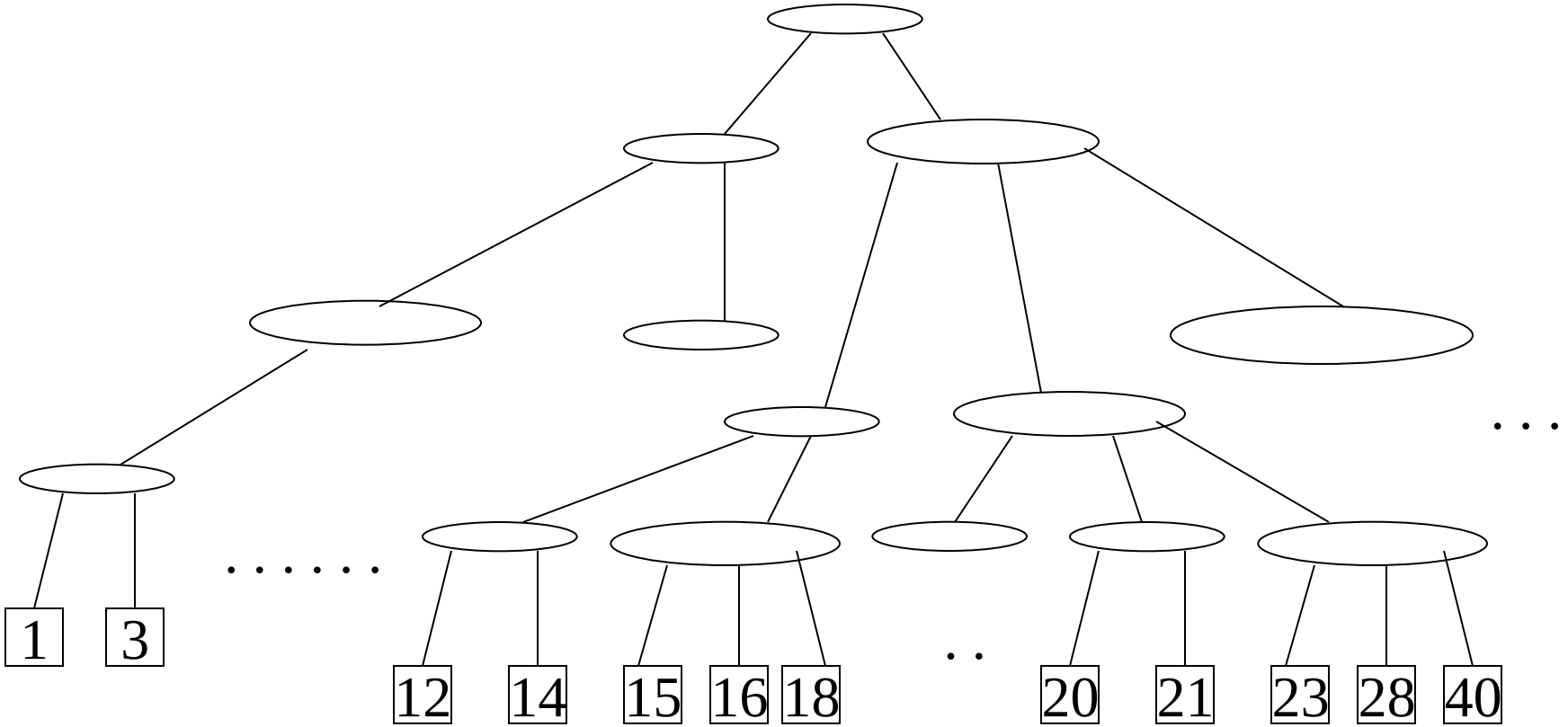
Insert (cont)



Insert (cont)



Insert (cont)



Insert -- definition

Add the new leaf in its position. Say under a node v.

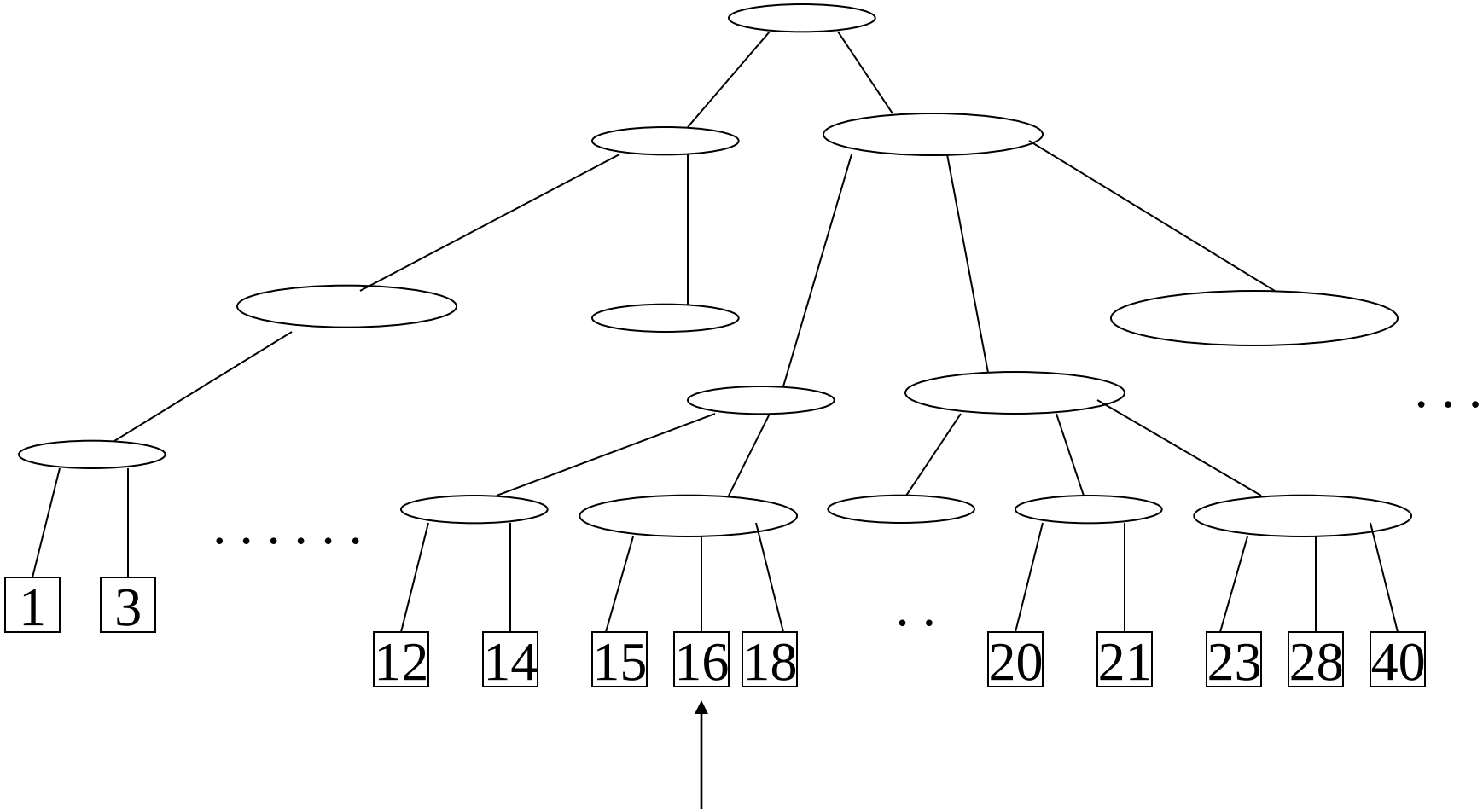
(*) If the degree of v is 5 split v into a 2-node u and a 3-node w.

If v was the root then create a new root r parent of u and w and stop.

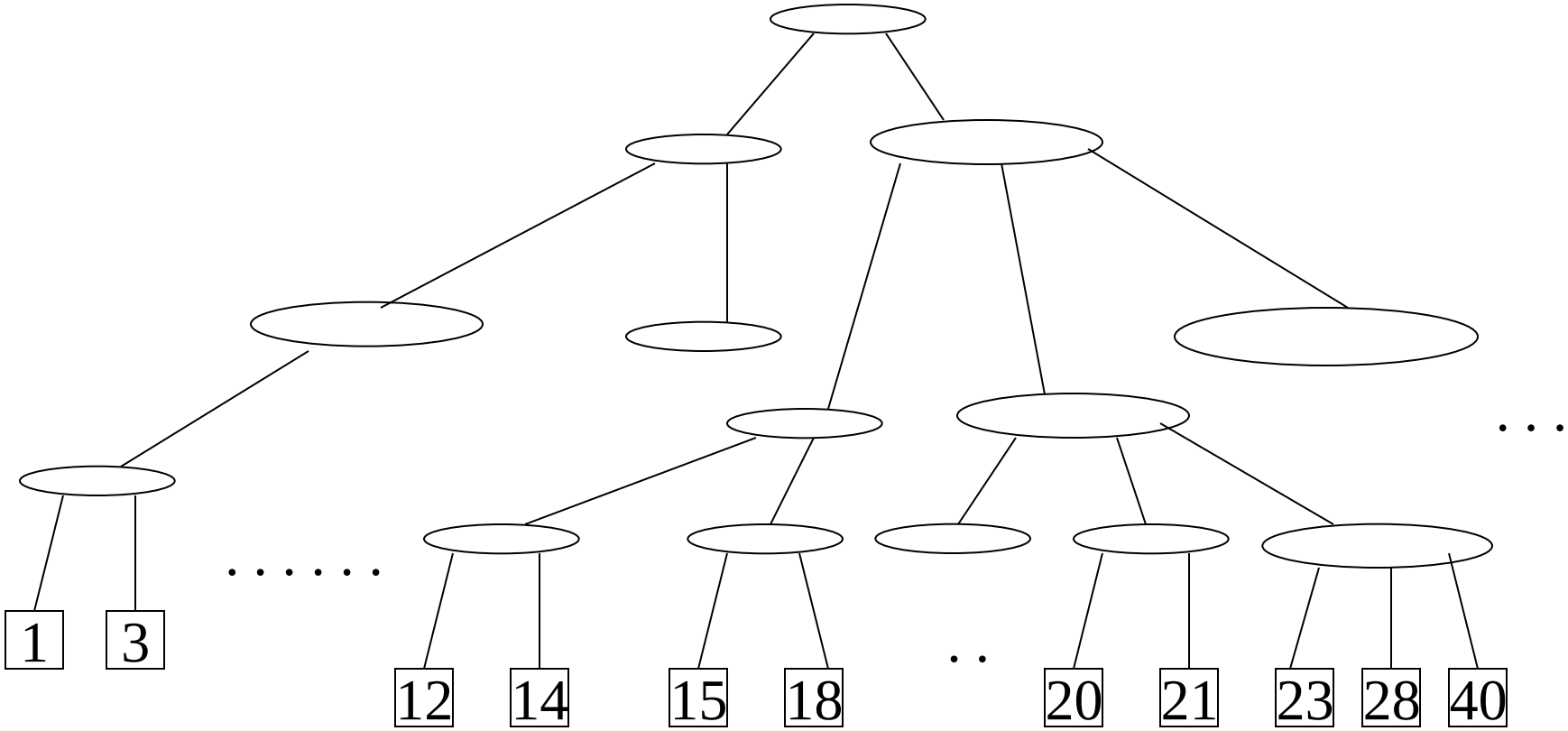
Replace v by u and w as children of p(v).

Repeat (*) for v := p(v).

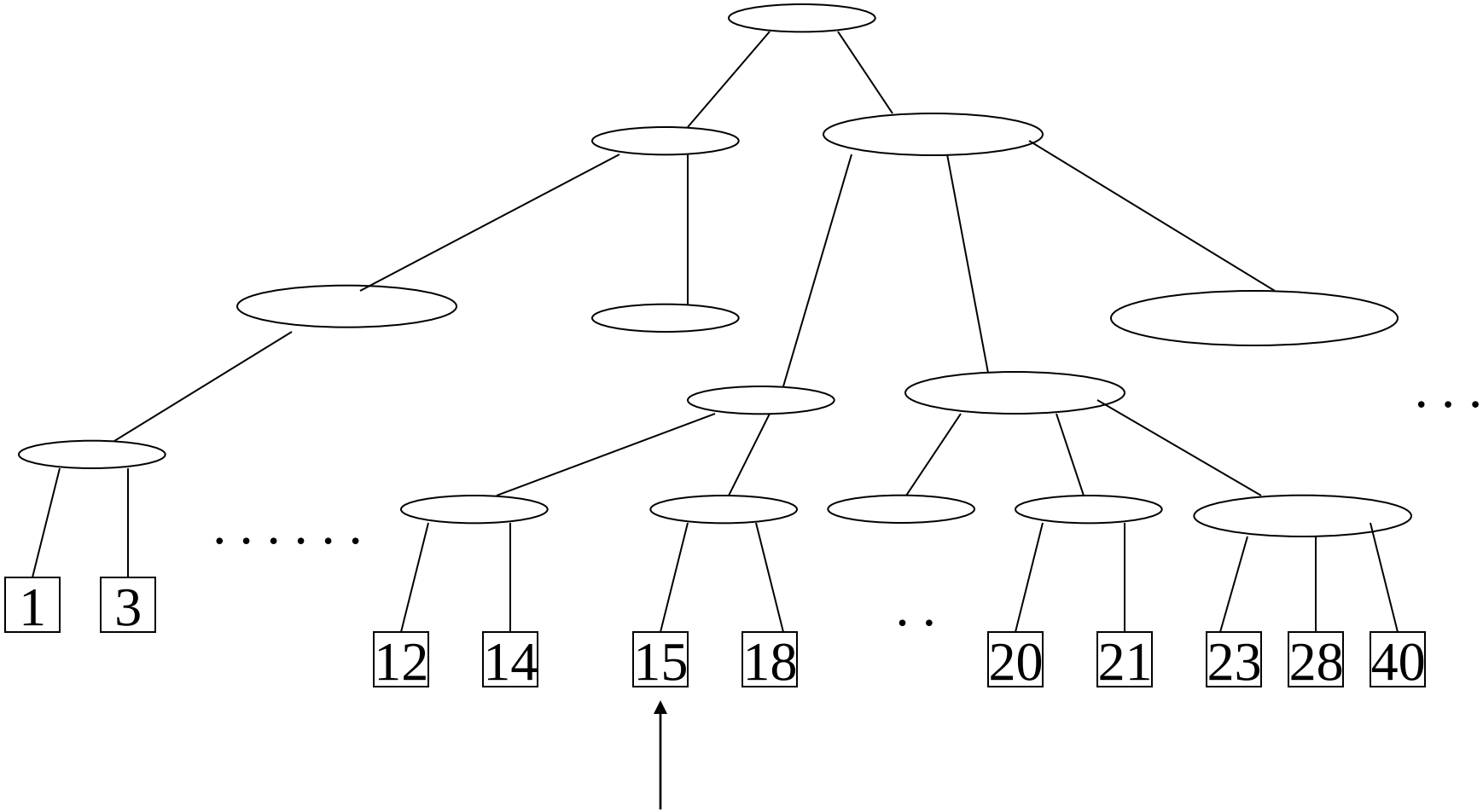
Delete



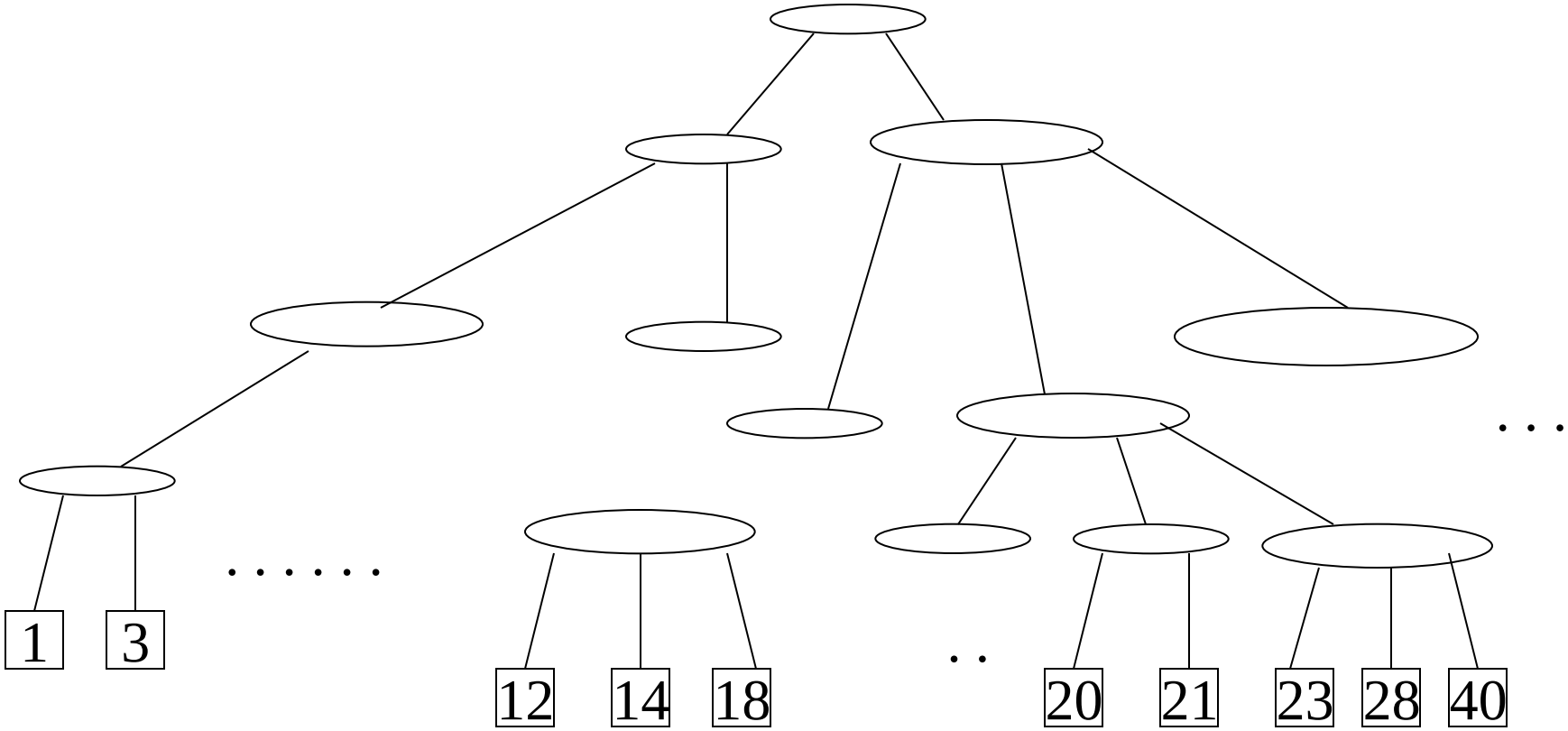
Delete



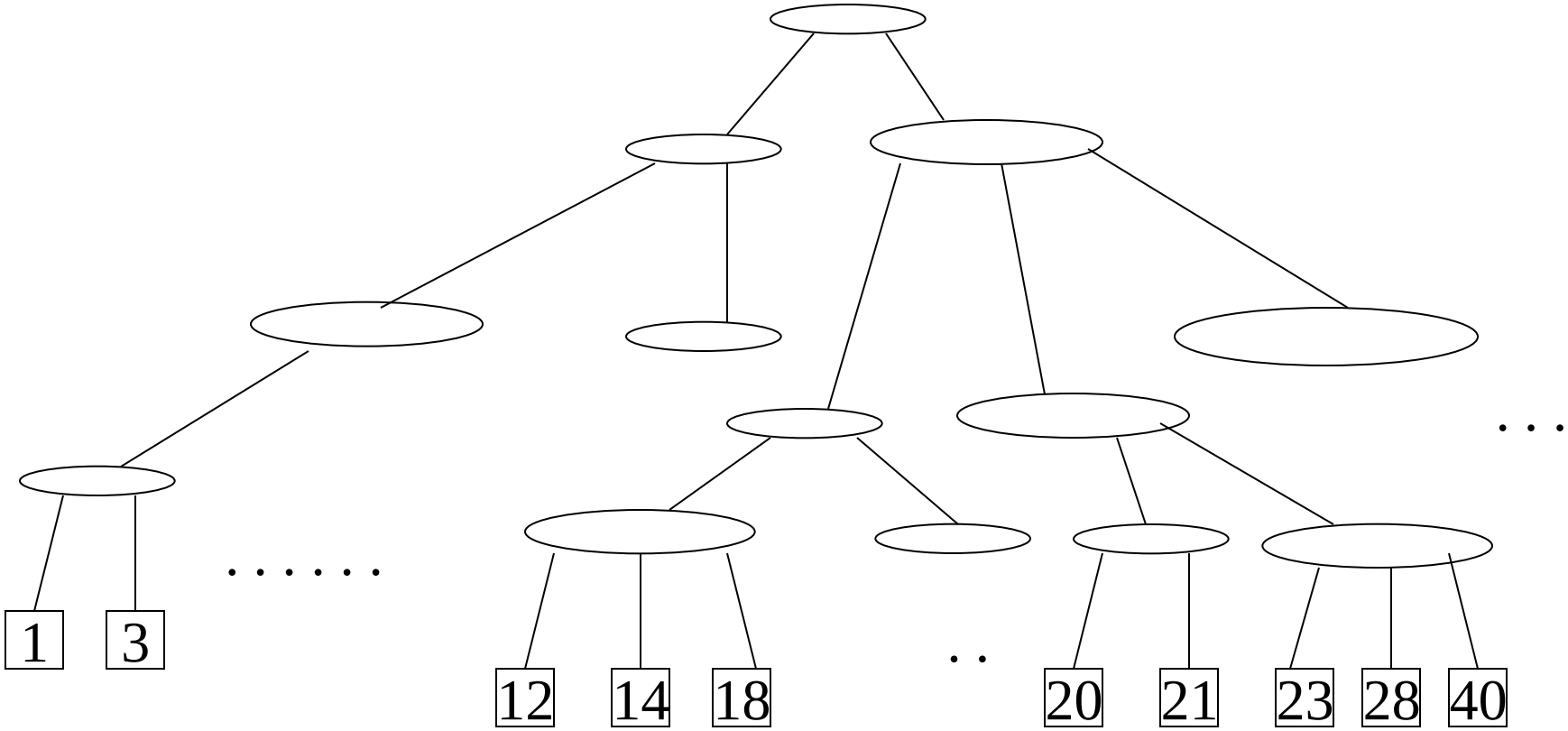
Delete



Delete



Delete



Delete -- definition

Remove the leaf. Let v be the parent of the removed leaf.

(*) If the degree of v is one, and v is the root discard v.

Otherwise (v is not a root), if v has a sibling w of degree 3 or 4, borrow a child from w to v and terminate.

Otherwise, fuse v with its sibling to a degree 3 node and repeat (*) with the parent of v.

Summary

Delete and insert take O(log n) on the worst case.

Theorem : A sequence of m delete and insert operations on an initial tree with n nodes takes O(m+n) time

Catenate similar to insert.

Implement split via catenate

Finger search trees

Same ADT

Keep sorted lists subject to the following operations:

find(x,L)

insert(x,L)

delete(x,L)

catenate(L1,L2) : Assumes that all items in L2 are greater than all items in L1.

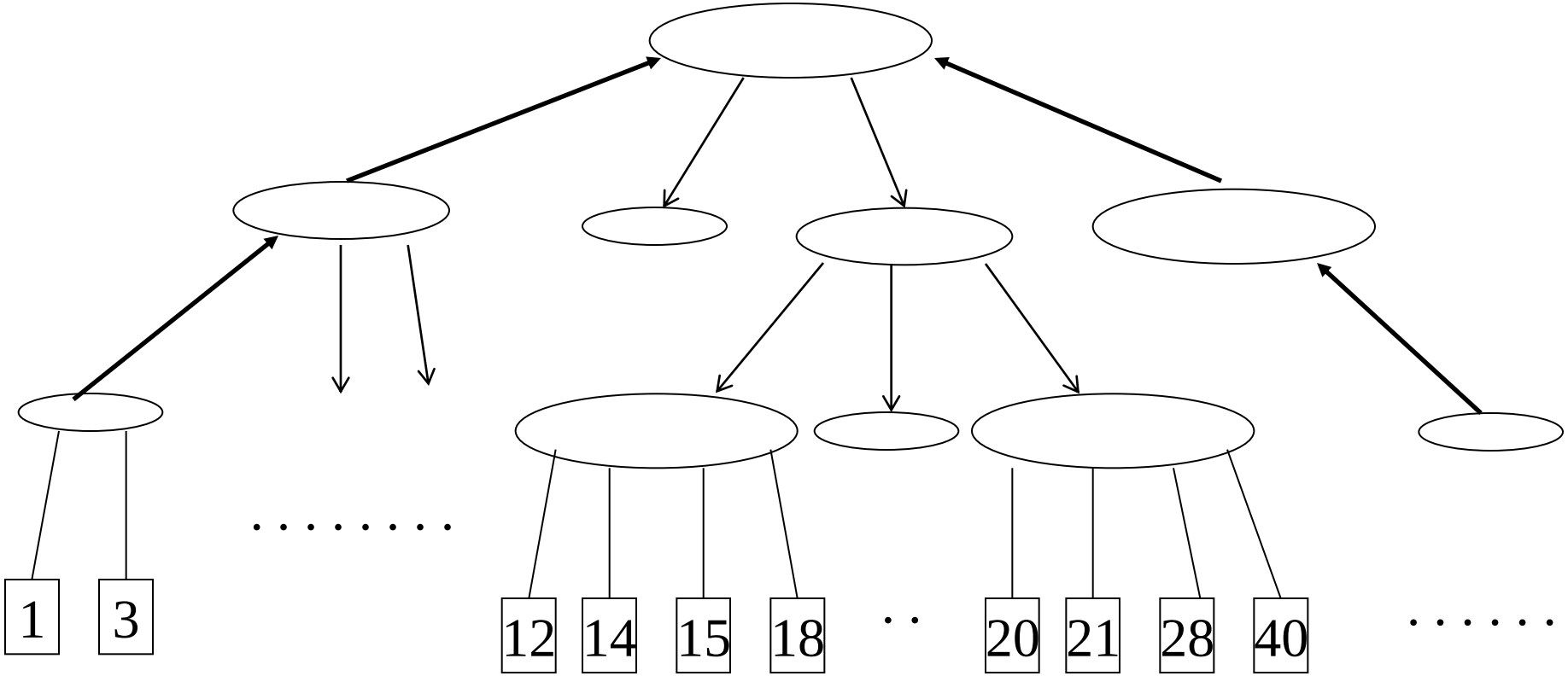
split(x,L) : returns two lists one with all item less than or equal to x and the other with all items greater than x.

Goal (cont)

In addition, we want to speed up operations near the ends of the list.

Solution: Take a regular search tree and reverse the direction of the pointers on the leftmost and the rightmost spines (paths).

Finger 2-4 trees



Finger 2-4 trees

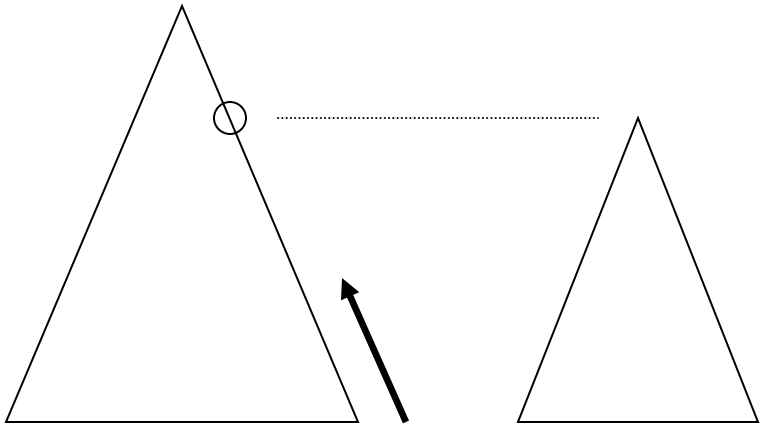
Start the search in parallel from the leftmost and rightmost nodes on the spines.

Search for an element at distance d from one of the endpoints of the list takes O(log d) time.

Insertions and deletions still take O(log n) worst case time but O(log d) amortized time (including the search time).

Finger trees catenation

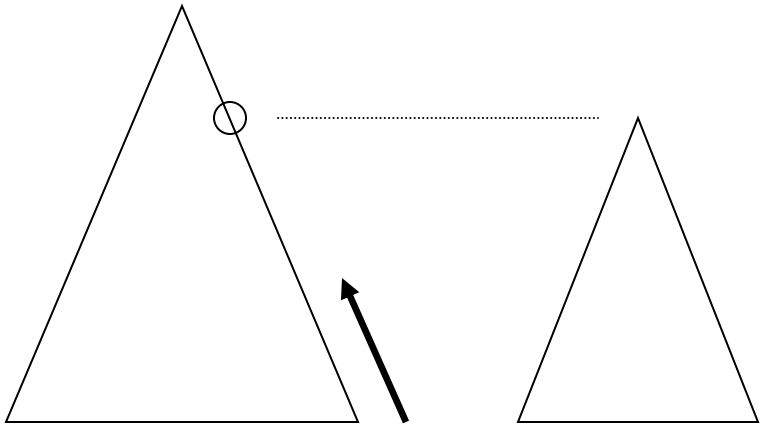
Like for regular trees but the search on the spine of the tall tree for a node of the same height as the small tree starts from the lowest node on the spine.



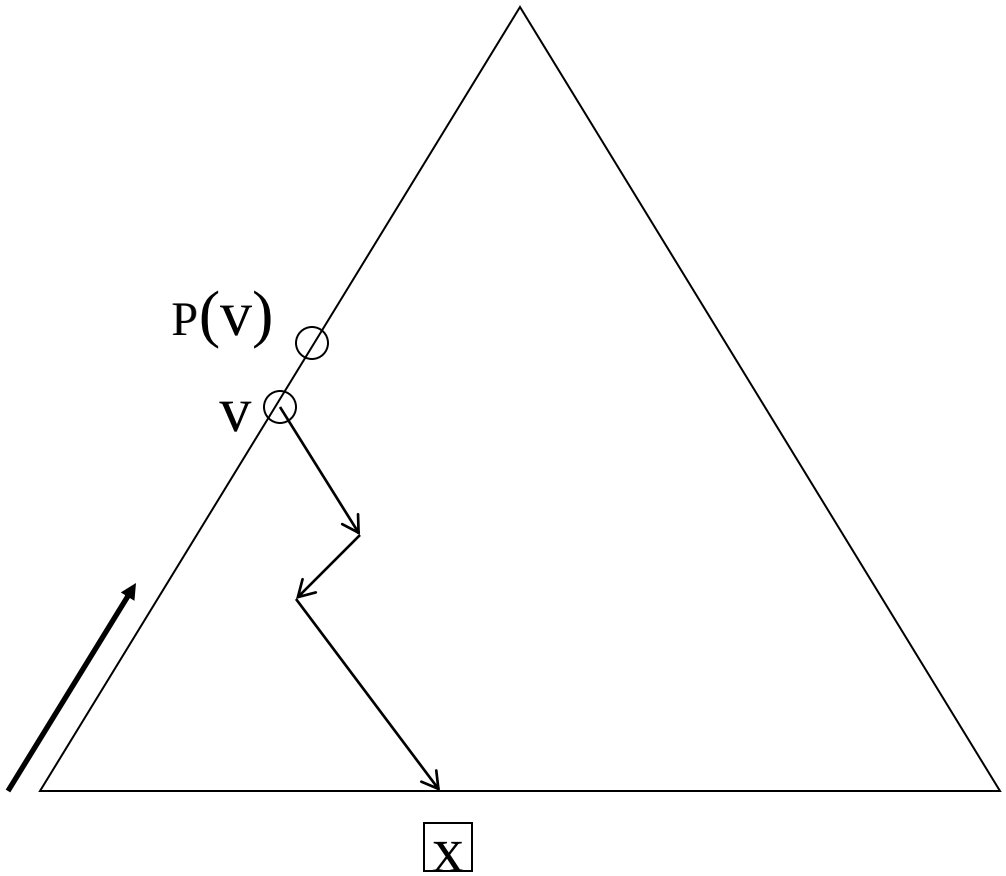
Finger trees catenation (cont)

Catenation takes O(log n) on the worst-case

But amortized O(1 + min{h1, h2}) =
O(1 + min{log (n1), log (n2))



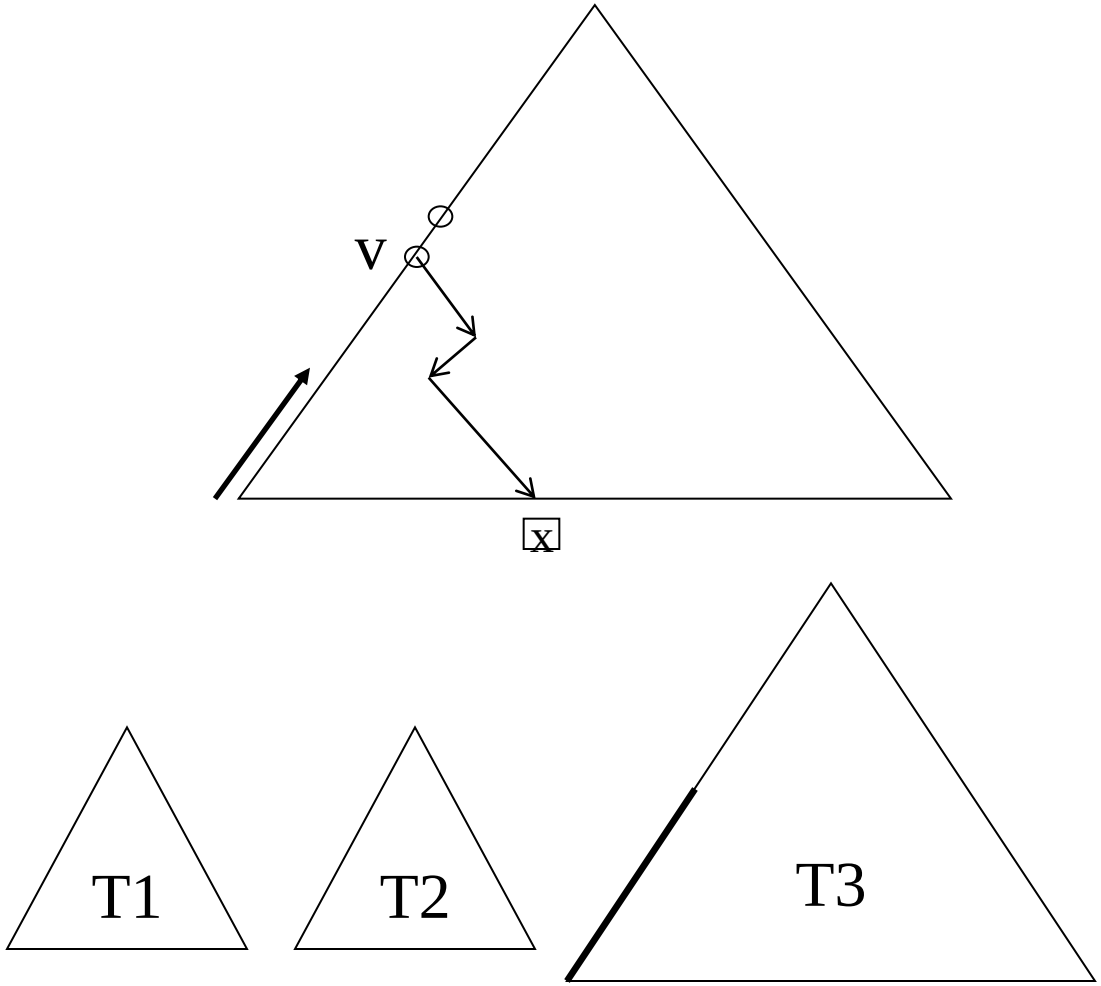
Splitting finger search trees



Catenate bottom-up trees to the left of the path from v to x and to the right of the path from v to x. Obtain T1 and T2.

Delete v as a child from p(v), rebalance as in deletion. Obtain T3.

Splitting finger search trees (cont)



Make T1 and T2 finger trees.

Fix the spine of T3.

Return T1 and the tree obtained from the catenation of T2 and T3.

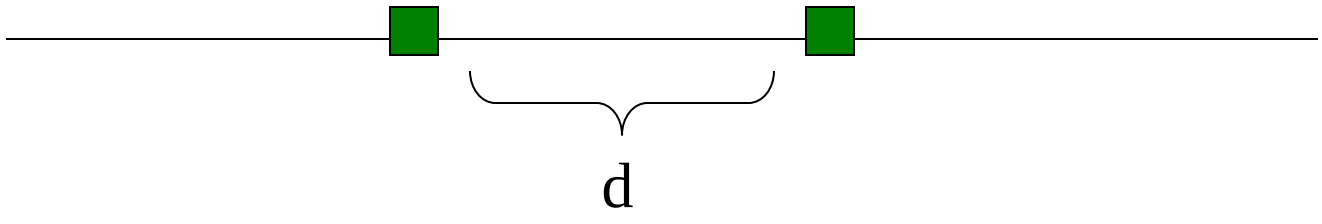
Split -- analysis

Split takes O(log n) worst-case.

But O(log d) amortized.

Homogenous finger search trees

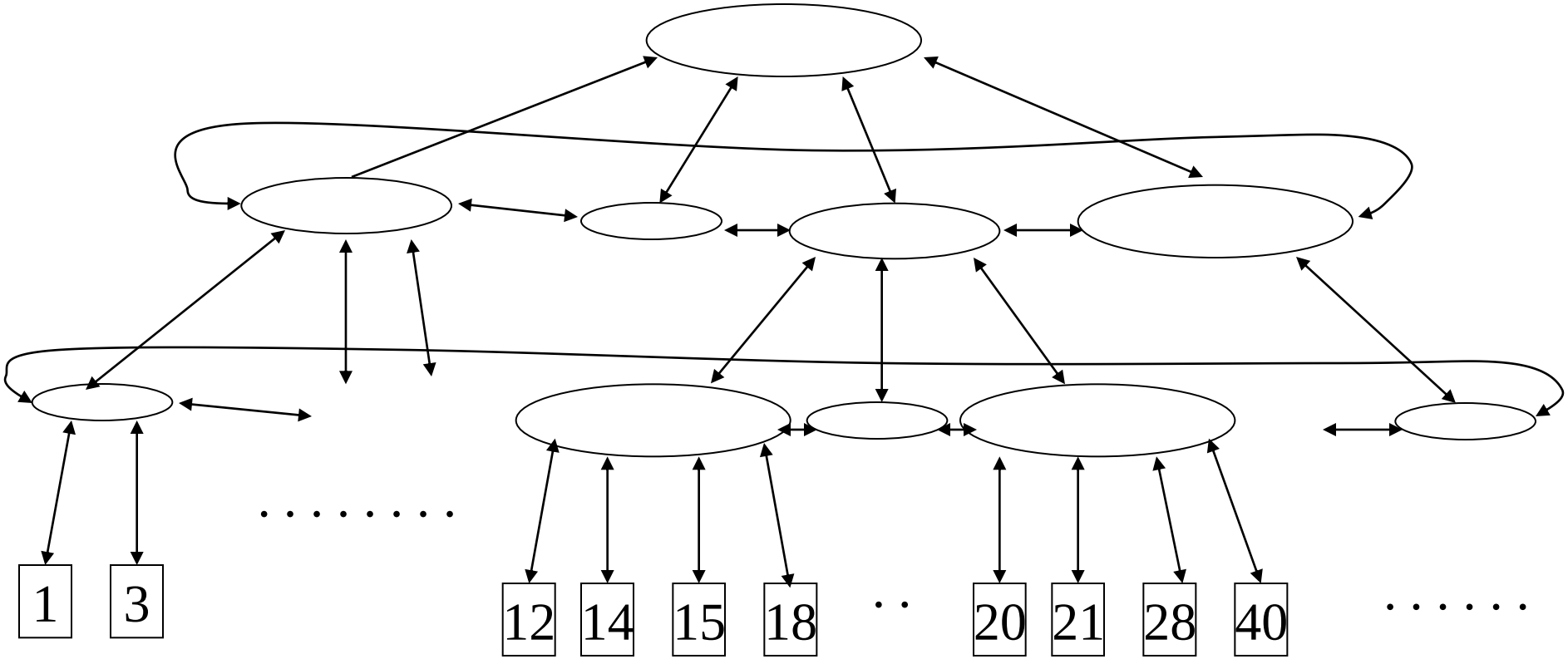
Want to be able to get from every element to every other element in time proportional to the logarithm of the distance between them.



O(log min {d, n-d})

Homogenous finger search trees (cont)

Add level links.



Search

Start from x and search for y.

Say $x < y$.

Keep going up until one of the following conditions holds:

1) You hit the right path of the tree

2) Your right neighbor has a key which is not smaller than y

3) You are on the left path and your neighbor on the right path has a key smaller than y

Search down one or two subtrees

Homework

Analyze the search

Show how to split out a sublist efficiently