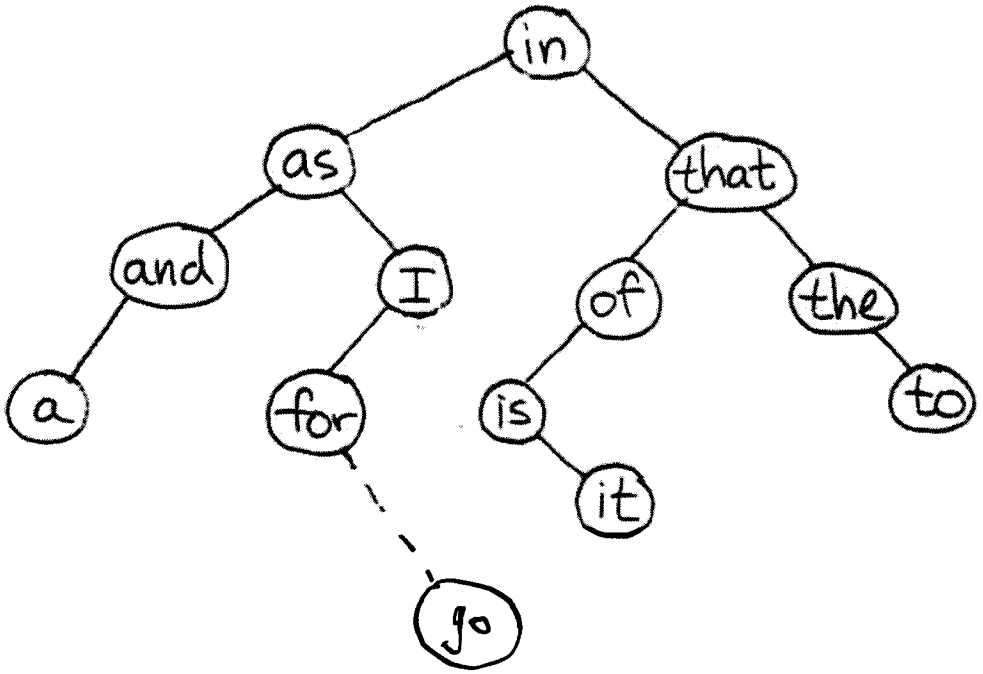
Binary Search Trees

Binary Tree: A rooted tree, each node having a left and a right child, either or both missing.

Binary Search Tree: Each node contains an item. Items are totally ordered and arranged in the tree in symmetric order: all items in left subtree are less, all items in right subtree are greater.

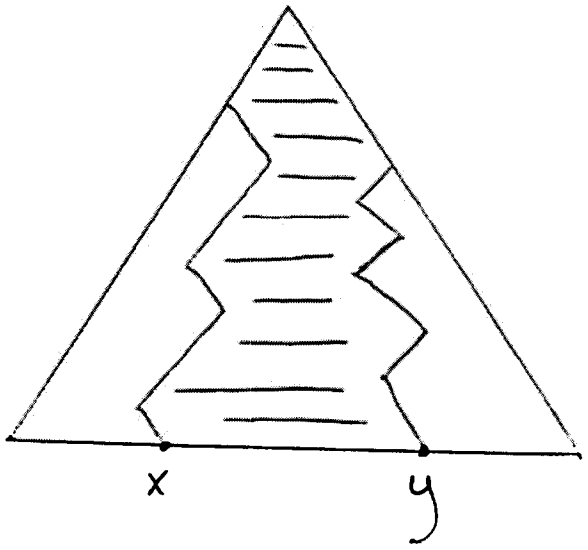
Binary search trees support access, insert, delete in $O(\text{depth})$ time.

A Binary Search Tree.

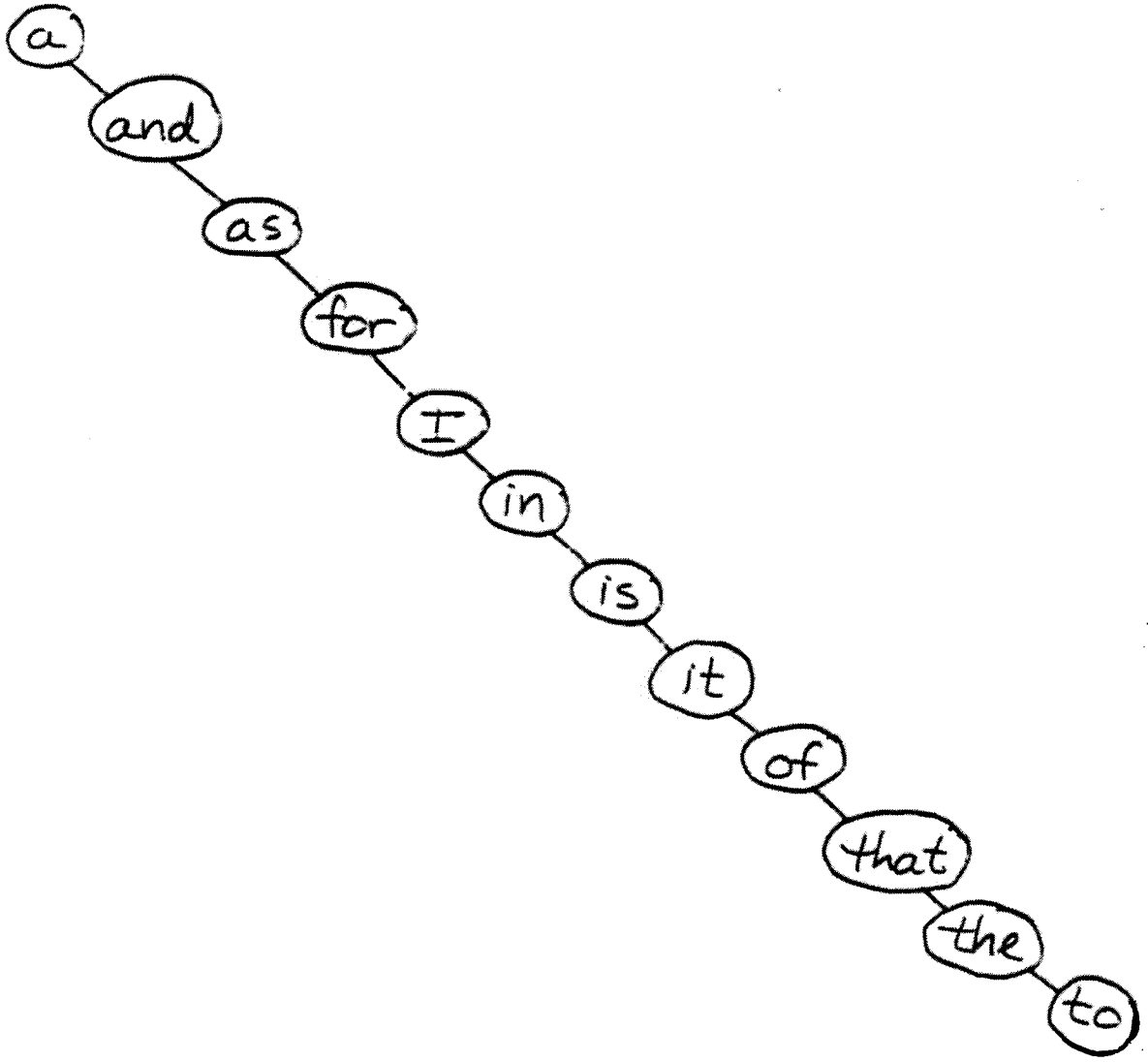


Search Trees Can Be Used For Range Queries

Report all entries between x and y:



Another binary search tree



How do we keep depth small?

Classical answer: Maintain a (local) balance condition.

Two properties:

(i) Implies $O(\log n)$ depth of a n-node tree.

(ii) Easily restorable after an updates $O(\log n)$ time by rebalancing along access path.

Since ~ 1962 many kinds of such

balanced search trees

have been discovered.

Classes of Balanced Trees

1. Height-balanced (AVL) trees
2. Weight-balanced (BB(α)) trees
3. 2,3 trees
4. B-trees
5. Brother trees
6. 2,4 trees

(3–6: not binary)

7. Symmetric binary B-trees
8. Red-black trees
9. Half-balanced trees

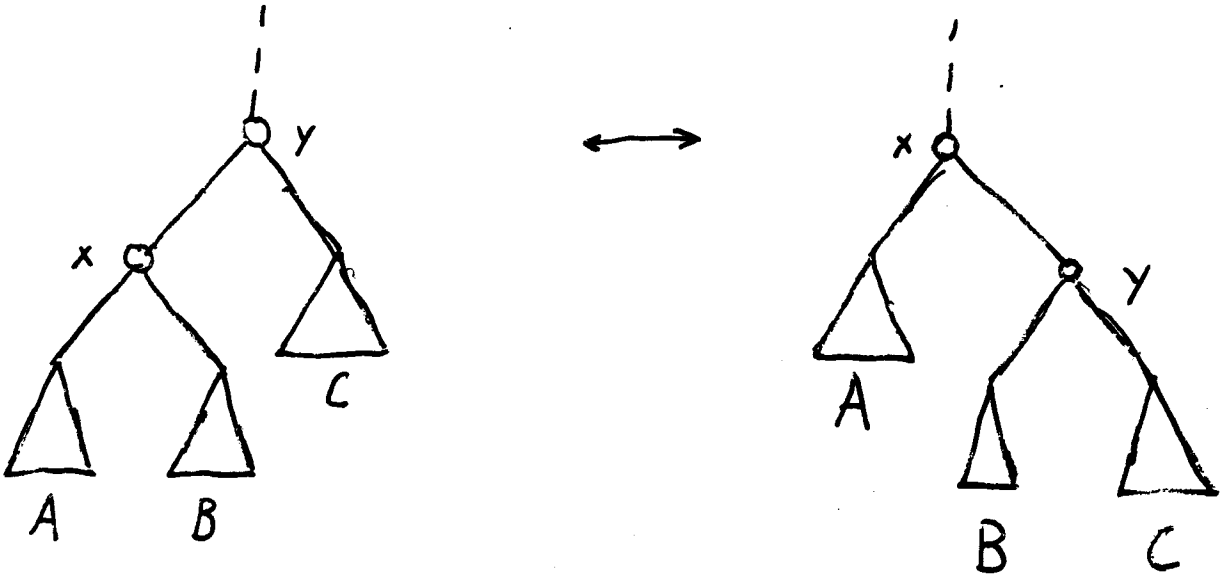
(7–9: equivalent)

etcetera

All achieve $O(\log n)$

access/insert/delete time

A Rotation



Changes depths of some nodes

Takes O(1) time (3 pointer changes)

Preserves symmetric order

Red-Black Trees

1. Each node is either red or black.
2. The root and all missing nodes are black.
3. There are no two red nodes in a row.
4. All paths from the root to a missing node have the same number of black nodes.

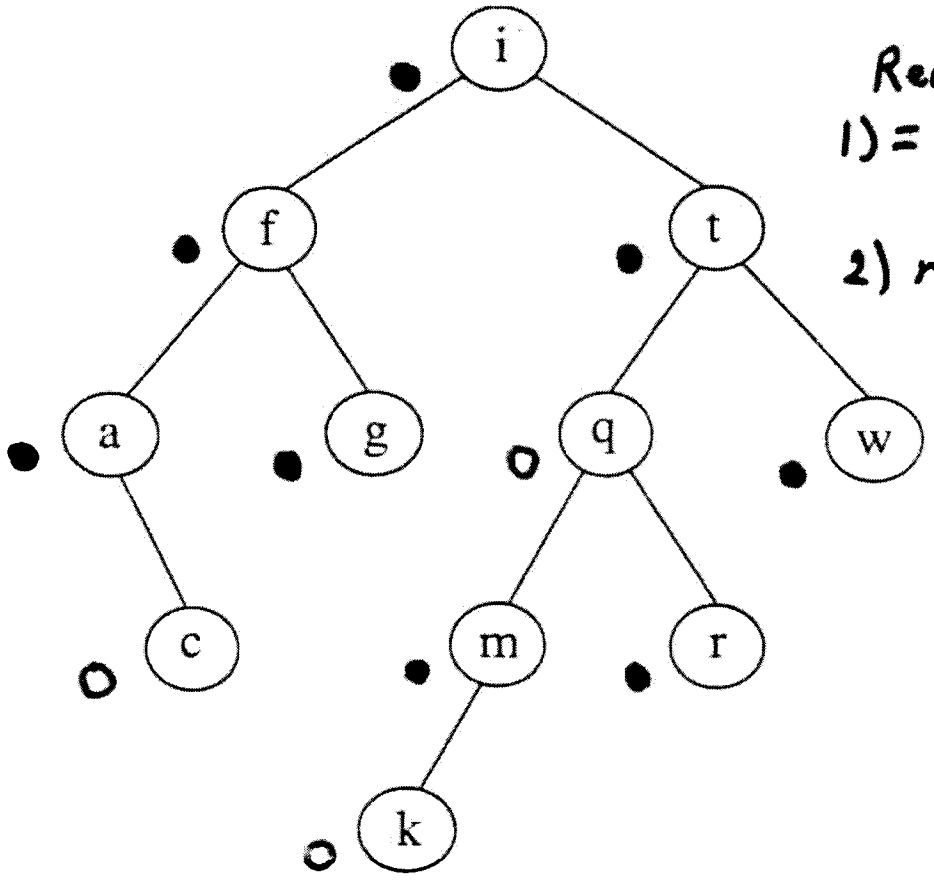
Equivalent to:

2,4 trees

Symmetric binary B-trees

Half-balanced trees

A Binary Search Tree



Red / Black:
1) = #s blacks on paths,
2) red nodes have black parents

Items in internal nodes, in symmetric order:

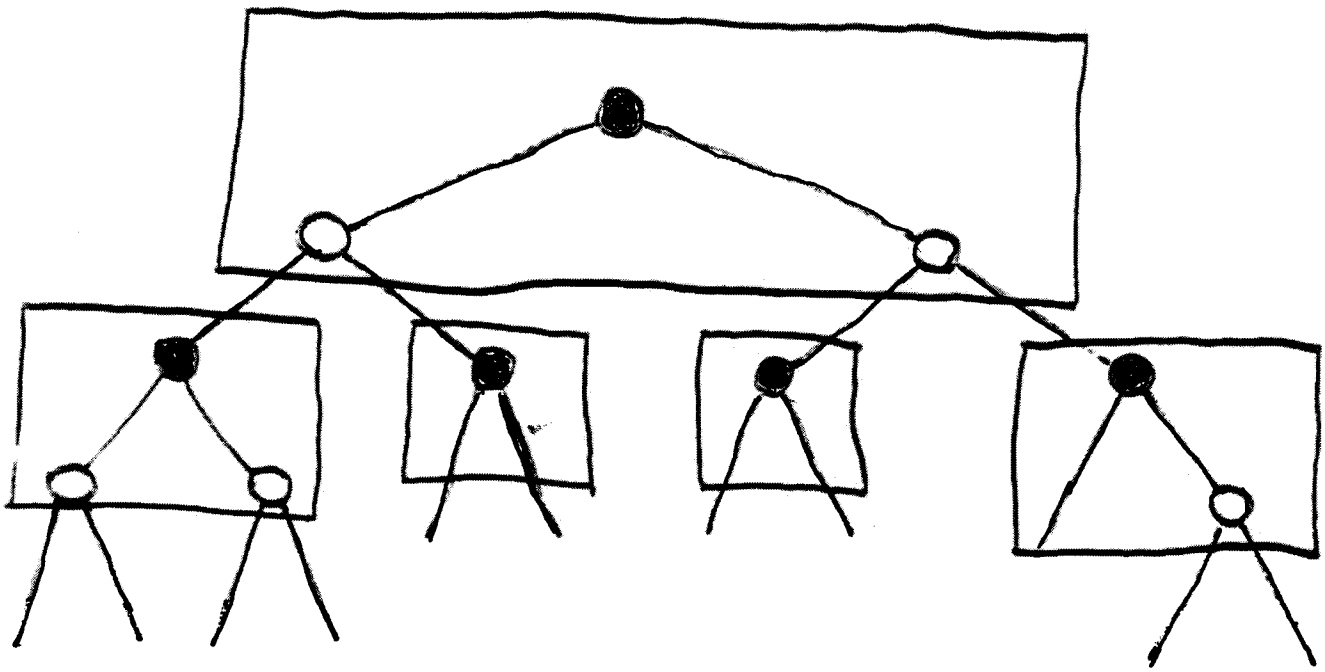
items in left subtree smaller,

items in right subtree larger.

Allows binary search for items

search time = 1+ depth.

A Red-Black Tree



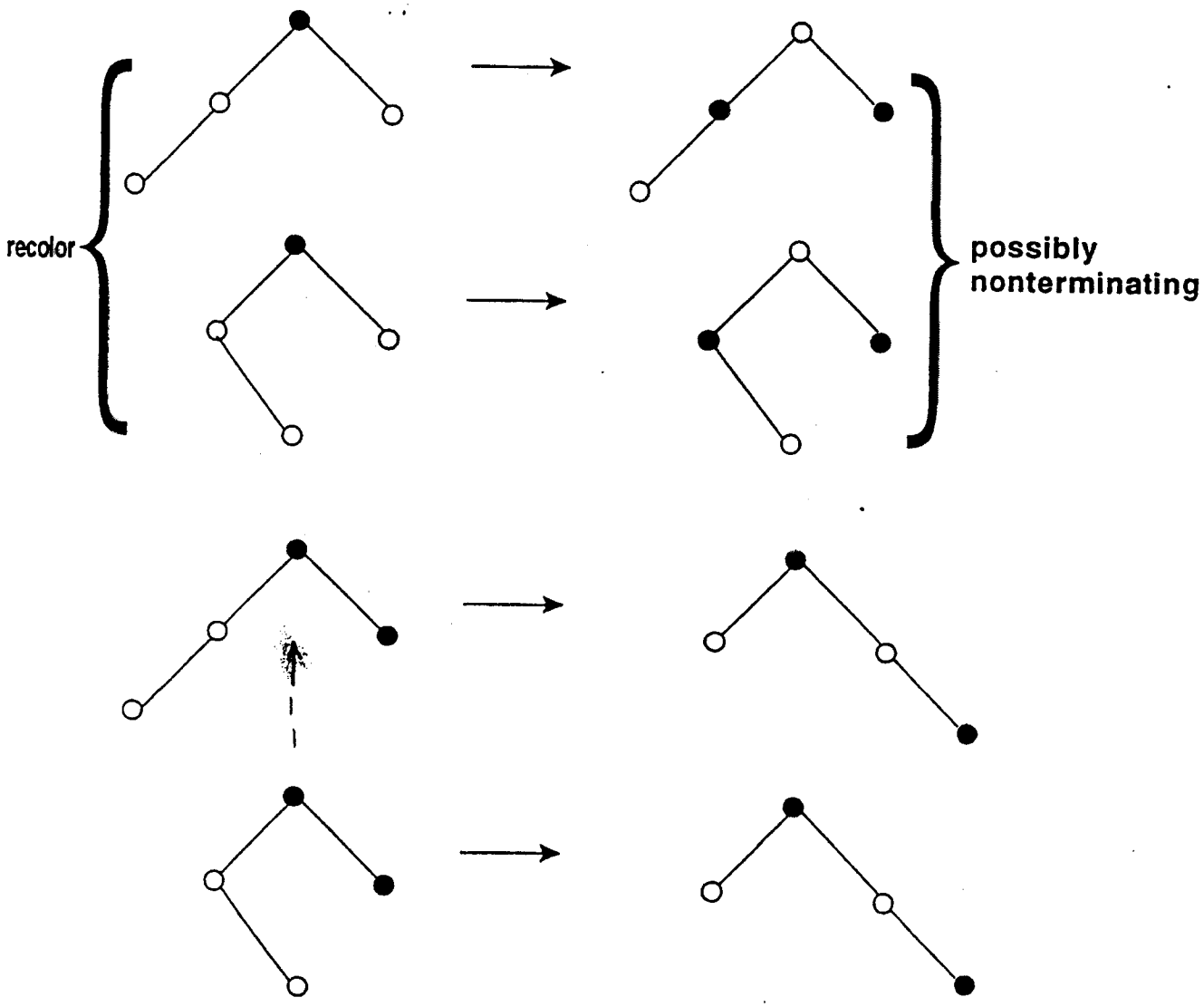
A 2,4 Tree

Red-black tree updates

● black

○ red

Insert ○ root ⟶ ●

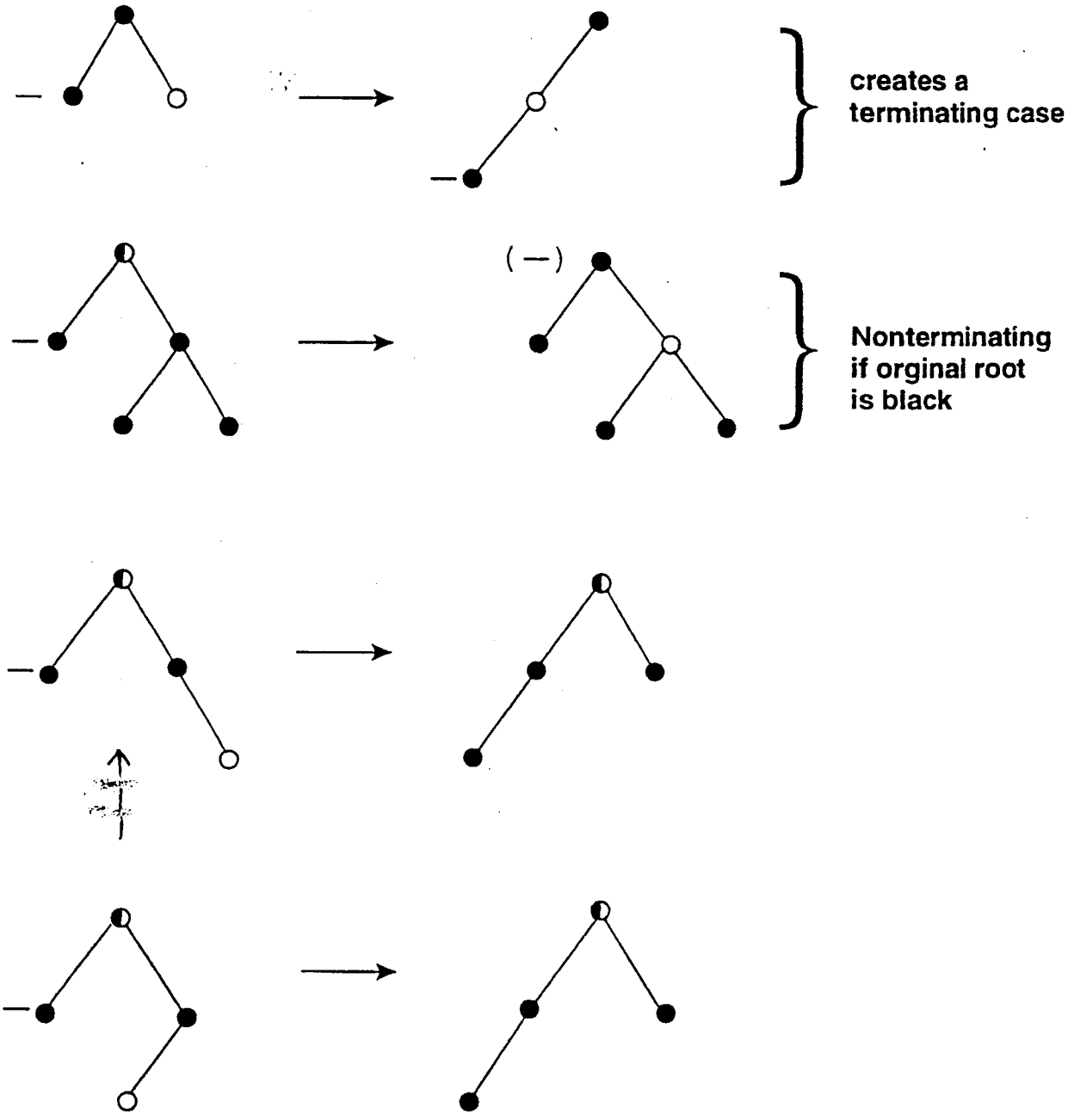


Delete — short node (all paths down lack one black node)

◐ red or black node (color preserved)

—● root ⟶ ●

—○ ⟶ ●



O(log n) recolorings; 0, 1, 2, or 3 rotations

O(1) amortized recoloring time for insert/delete:

Φ = 3/2 for  , 1/2 for