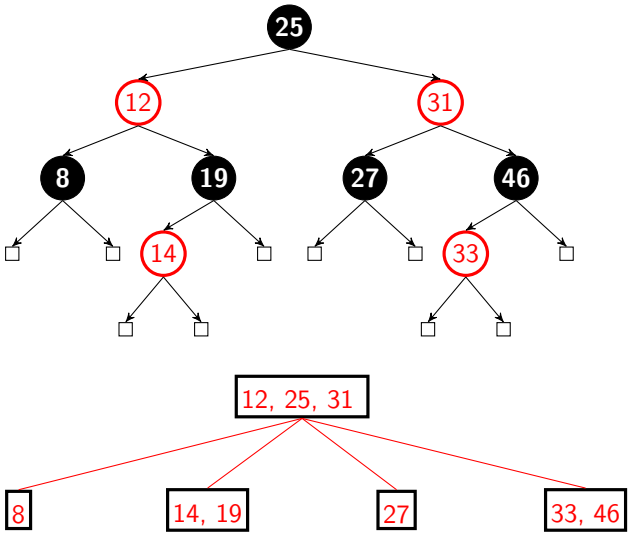
CS 61B Spring 2022

Graphs & LLRBs

Discussion 13: April 18, 2022

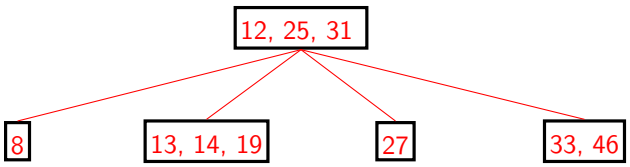
1 Balanced Search Trees

(a) Convert the red-black tree into a 2-4 tree. Solid nodes are black.

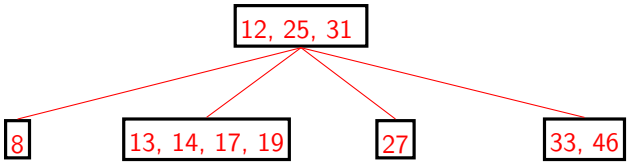


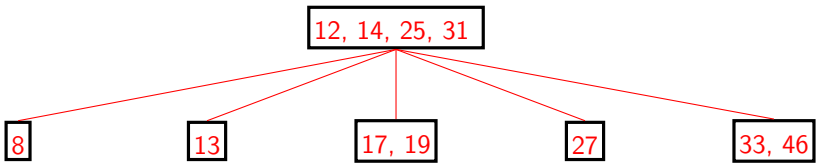
(b) Insert the keys 13 and 17 into the resulting 2-4 tree. Assume that, if a node has 4 keys, we choose to push up the left of the 2 middle keys (so the 2nd key from the left).

After adding 13:

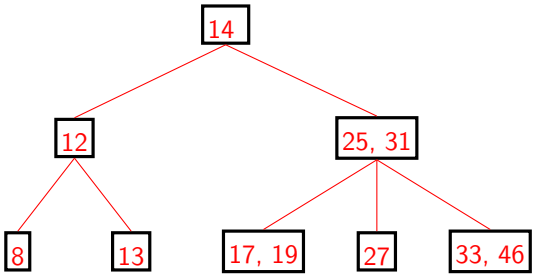


Intermediate steps after adding 17:

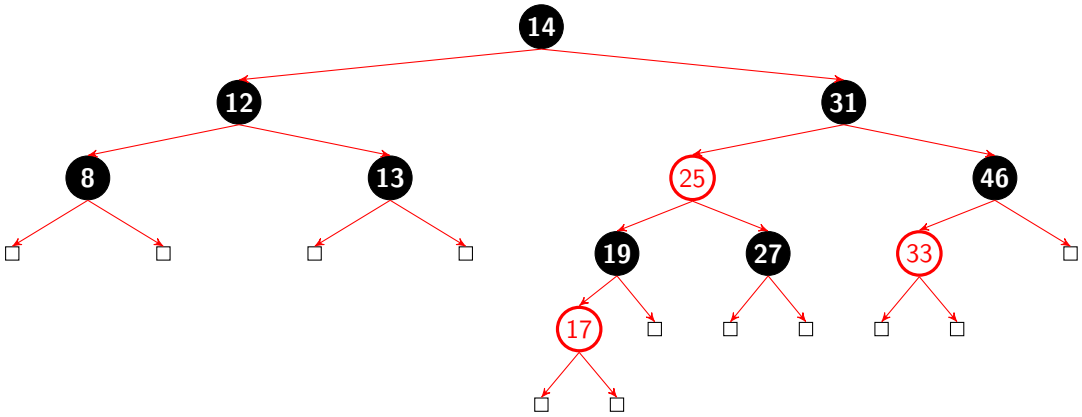




Final result after adding 13 and 17:



(c) Convert the resulting 2-4 tree into a valid left-leaning red-black tree.

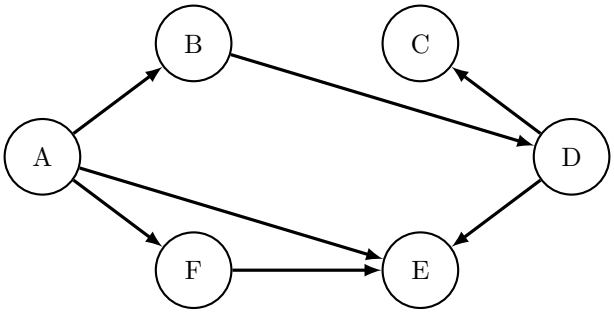


(d) Given a 2-4 tree containing N keys, describe how you can obtain the keys in sorted order in worst case $O(N)$ time.

Generalize an in-order traversal: traverse the left (first) child of the node, emit the first key of the node, traverse the second child of the node, emit the second key of the node, etc.

(e) If a 2-4 tree has depth H (that is, the leaves are at a distance of H from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

$2H$ comparisons. The number of black nodes from root to leaf is the same for all nodes in a red-black tree and it is equal to the height of its equivalent 2-4 tree. The maximum number of comparisons occurs on the longest root to leaf path (i.e. the path with the most nodes). We can also have at most one red node after each black node (we can't have more than two red nodes in a row in a red-black tree), resulting in a longest possible path that is $2H$ nodes long. As a result, we will make at most $2H$ comparisons.



2 Graph Representation

Represent the graph above with an adjacency list and an adjacency matrix representation.

Depending on the convention being used, nodes may or may not have edges to themselves. For this problem, we stick with the convention that nodes do **not** have an edge to themselves.

Adjacency List

A → [B, E, F]
B → [D]
C → []
D → [C, E]
E → []
F → [E]

FROM / TO

	A	B	C	D	E	F
A	0	1	0	0	1	1
B	0	0	0	1	0	0
C	0	0	0	0	0	0
D	0	0	1	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	1	0

(in the above matrix 0 means false and 1 means true)

Note: Edge lists and adjacency lists are **not** the same! An edge list is a list stored in each node that contains all successors and possibly predecessors of that node (see lecture). An adjacency list is more of a table (or map data structure) that lists the adjacent vertices for each vertex in the graph; it is a representation of the graph as a whole. Graphs are commonly represented using adjacency lists and matrices but be aware of what is meant by an edge list.

3 Searches and Traversals

Run depth first search (DFS) preorder, DFS postorder, and breadth first search (BFS) on the graph above, starting from node A. List the order in which each node is first visited. Whenever there is a choice of which node to visit next, visit nodes in alphabetical order.

DFS has both a preorder and postorder traversal. Preorder means we visit the node *before* visiting its children. Postorder order means we visit the node only *after* visiting its children.

DFS preorder: A, B, D, C, E, F

DFS postorder: C, E, D, B, F, A

BFS: A, B, E, F, D, C

4 Topological Sorting

Give a valid topological ordering of the graph. Is it unique?

A topological ordering is a linear ordering of nodes such that for every directed edge S → T, S is listed before T. For this problem, the topological ordering of the graph is **not** unique. Below, we list two valid topological orderings for the graph.

- One valid ordering: A, B, D, C, F, E
  - Explanation: One way to approach this problem is to take any node with no edges leading to it and return it as the next node. After returning a node, we delete it and any edges leaving from it and look for a node with no incoming edges in the updated graph. We can repeat this until we have no nodes left. If at any point in this process we have a multiple choices for which node to return then the topological ordering is not unique.
- Another possible valid ordering: A, F, B, D, E, C
  - Explanation: Note that this ordering is just the reverse of DFS postorder traversal. Reverse DFS postorder will always be a valid topological ordering. This is because a DFS postorder traversal visits nodes only after all successors have been visited, so the reverse traversal visits nodes only after all predecessors have been visited.

Note: Only Directed Acyclic Graphs (DAGs), which are directed graphs that do not contain any cycles, have topological orderings. This is because within any given cycle, no one node comes before another. There are no valid topological orderings for undirected graphs because there is no direction associated with any edge. No one node comes before another, so it does not make sense to have a topological ordering for undirected graphs.