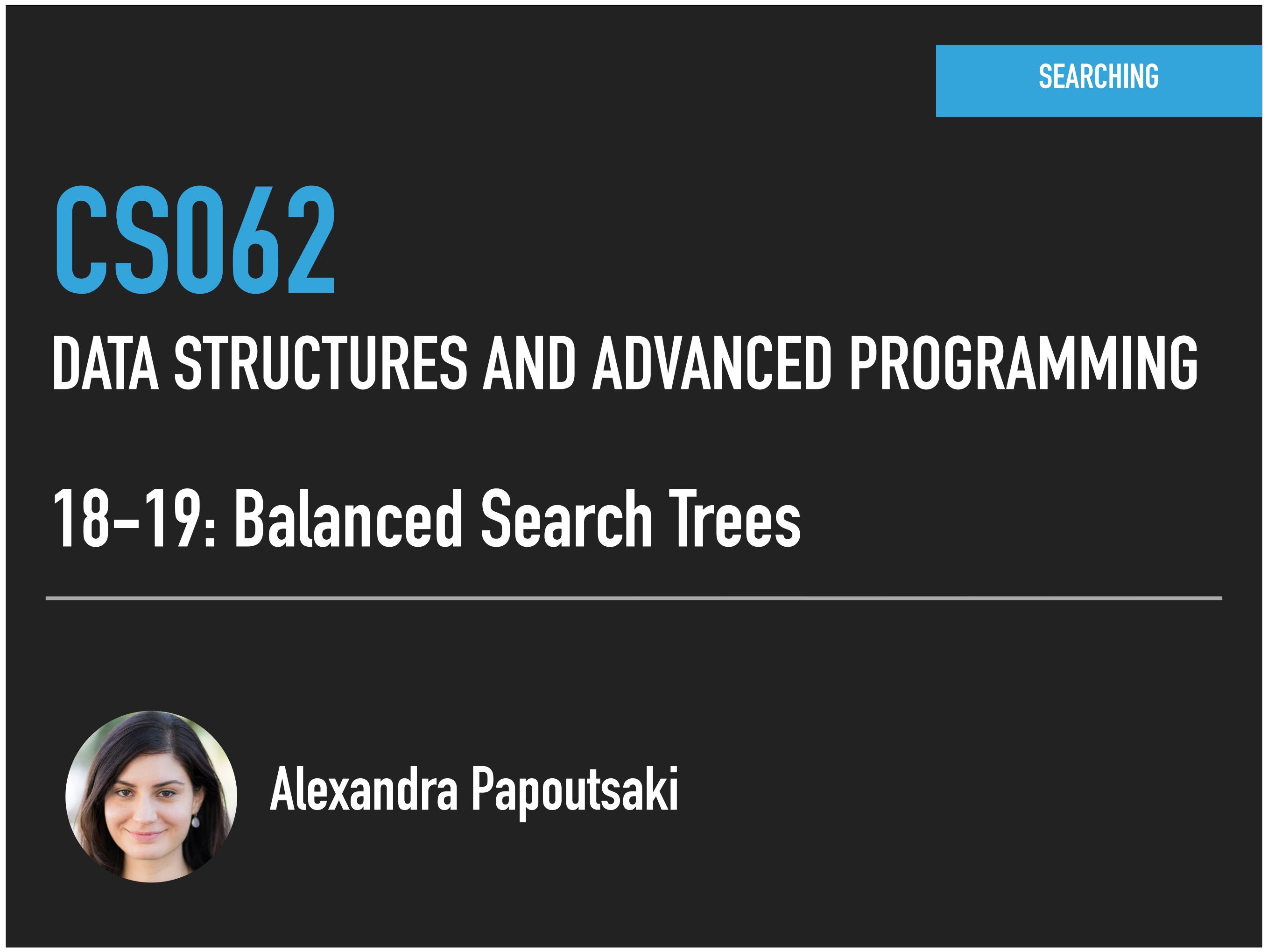


SEARCHING

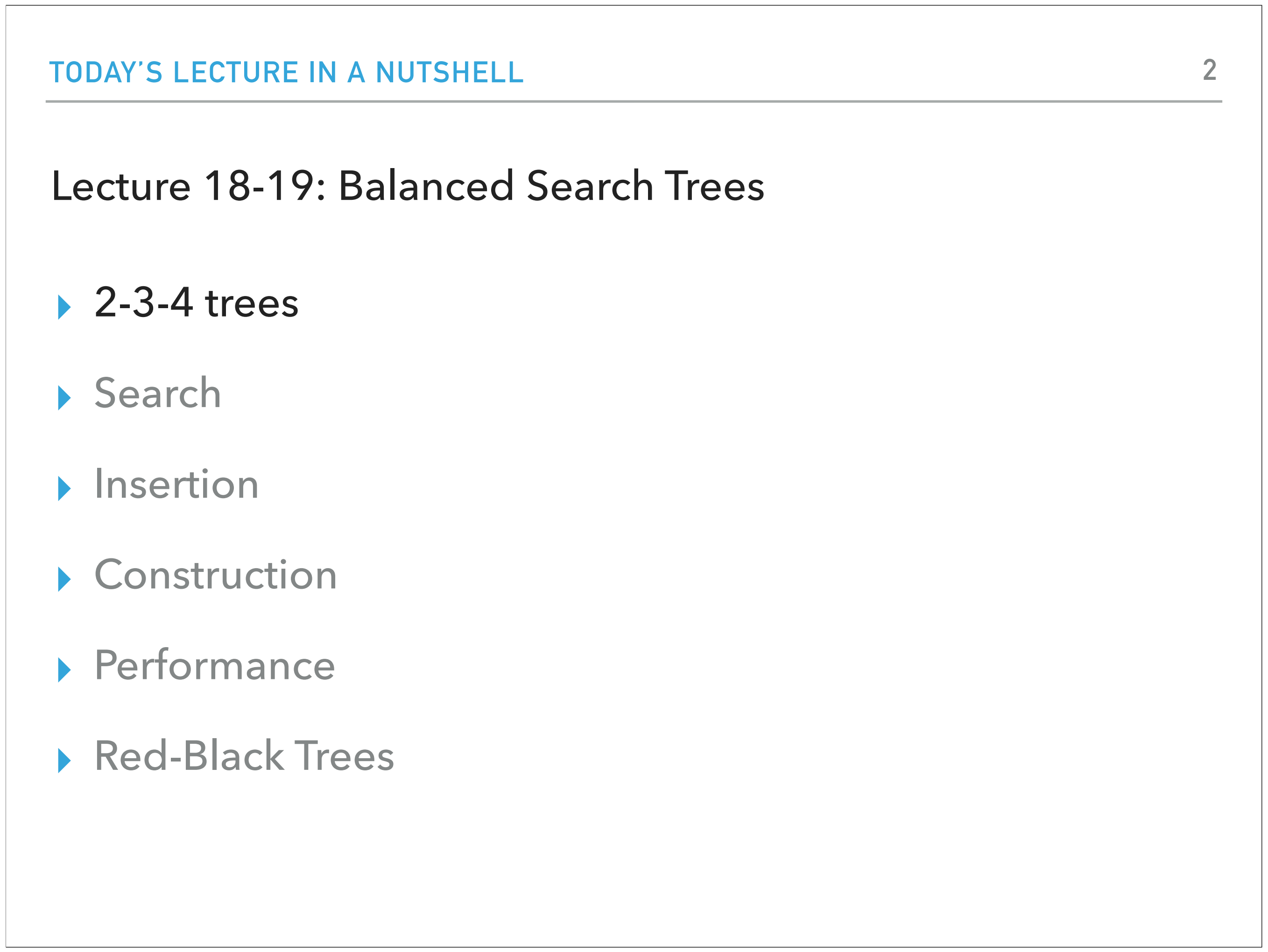
CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

18-19: Balanced Search Trees

Alexandra Papoutsaki

Last time, we talked about dictionaries, data structures that allow us to store key value pairs and search for a value given a key. We saw that binary search trees are one implementation of dictionaries.



Lecture 18-19: Balanced Search Trees

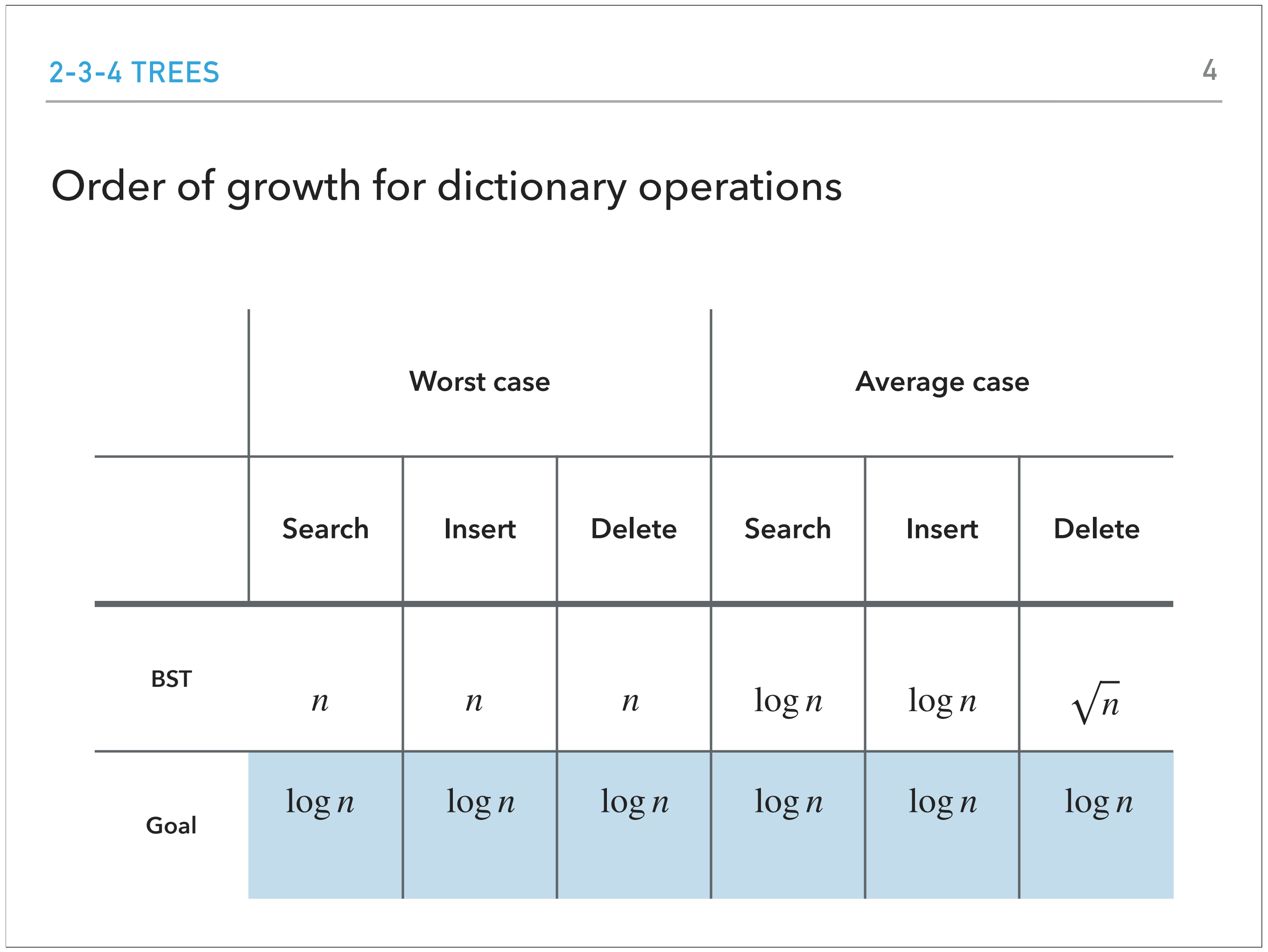
- 2-3-4 trees
- Search
- Insertion
- Construction
- Performance
- Red-Black Trees

Today we will talk about balanced search trees and mostly about one such type of balanced search trees, 2-3-4 search trees. Why do we need balanced search trees?

Visualization of insertion into a binary search tree

- 255 insertions in random order.

The main limitation of basic binary search trees is that they can become imbalanced. Here is a visualization of 255 random insertions. The tree can grow unpredictably and become quite long which will result to longer search times when we search for a value given a key.



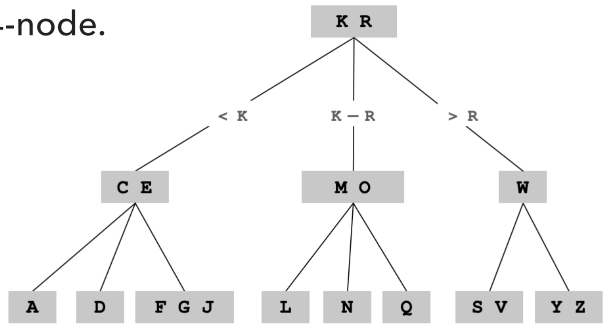
Order of growth for dictionary operations

	Worst case			Average case		
	Search	Insert	Delete	Search	Insert	Delete
BST	n	n	n	$\log n$	$\log n$	$\sqrt{n}$
Goal	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$

Our goal today will be to overcome the limitations imposed by BSTs when implementing dictionaries for the worst case so that instead of linear running time, we guaranteed logarithmic cost for searching, inserting, and deleting information from a dictionary.

2-3-4 tree

- Definition: A 2-3-4 search tree is either empty or consists of three types of nodes: 2-node, a 3-node, or a 4-node.
  - 2-node: one key, two children
  - 3-node: two keys, three children
  - 4-node: three keys, four children

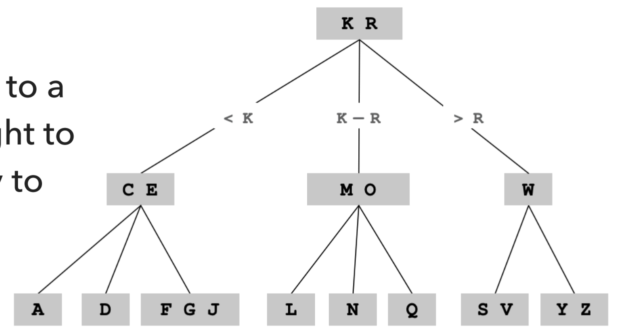


- Balanced 2-3-4 tree: A 2-3-4 search tree with with all paths from root to a null link has the same length, that is all leaves have the same depth.
  - From now on, 2-3-4 trees are assumed to be balanced.

To do so, we will consider 2-3-4 trees. A 2-3-4 tree is either empty or consists of three types of nodes: 2- or 3- or 4-nodes. The number indicates how many children each node has; also that number-1 corresponds to the number of keys it holds. Balanced 2-3-4 trees are 2-3-4 search trees that all paths from root to null link has the same length. I will use the term, 2-3-4 search tree to mean a balanced 2-3-4 search tree.

2-node

- Definition: a node with one key (and associated value) and two links, a left to a 2-3-4 tree with smaller keys, and a right to a 2-3-4 tree with larger keys (similarly to standard BSTs).

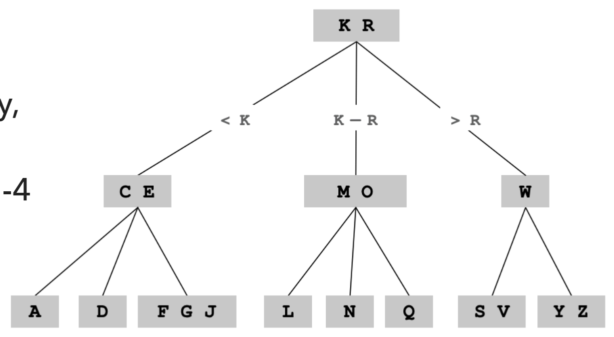


- E.g., the node that holds W is a 2-node. It has two children. Left child is a 2-3-4 tree with keys smaller than W and right child is a 2-3-4 tree with keys larger than W.

A 2-node is a node with one key (and its associated value) and two links, each to a 2-3-4 tree. The left child contains smaller keys and the right tree larger keys (similar to the standard binary search tree).

3-node

- Definition: a node with two keys (and associated values) and three links, a left to a 2-3-4 tree with smaller keys than the first key, a middle to a 2-3-4 tree with keys between the first and second key, and a right to a 2-3-4 tree with larger keys than the second key.

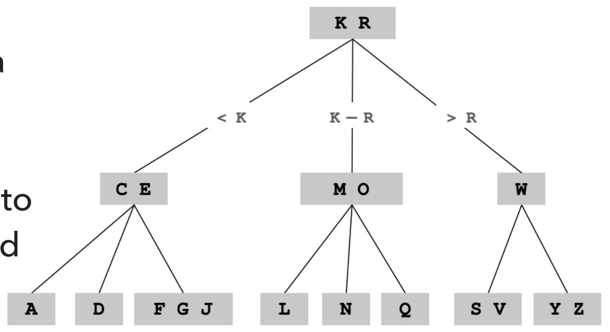


- E.g., the node that holds K and R is a 3-node. It has three children. Left child is a 2-3-4 tree with keys smaller than K, middle is a 2-3-4 tree with keys between K and and R, and right child is a 2-3-4 tree with keys larger than R.

A 3-node contains two keys (and their associated values) and three links: a left to a 2-3-4 tree with smaller keys that the first key, a middle to a 2-3-4 tree with keys in between the node's keys, and a right one to a 2-3-4 tree with larger keys than the second key.

4-node

- Definition: a node with three keys (and associated values) and four links, first to a 2-3-4 tree with smaller keys than the first key, a second to a 2-3-4 tree with keys between the first and second key, a third to a 2-3-4 tree with keys between the second and third key, and a fourth to a 2-3-4 tree with larger keys than the fourth key.



- E.g., the node that holds F, G, and J is a 4-node.

A 4-node is a node with three keys (and associated values) and four links, first to a 2-3-4 tree with smaller keys than the first key, a second to a 2-3-4 tree with keys between the first and second key, a third to a 2-3-4 tree with keys between the second and third key, and a fourth to a 2-3-4 tree with larger keys than the fourth key.

Lecture 18-19: Balanced Search Trees

- 2-3-4 trees
- Search
- Insertion
- Construction
- Performance
- Red-Black Trees

Let's see how can we search 2-3-4 trees.

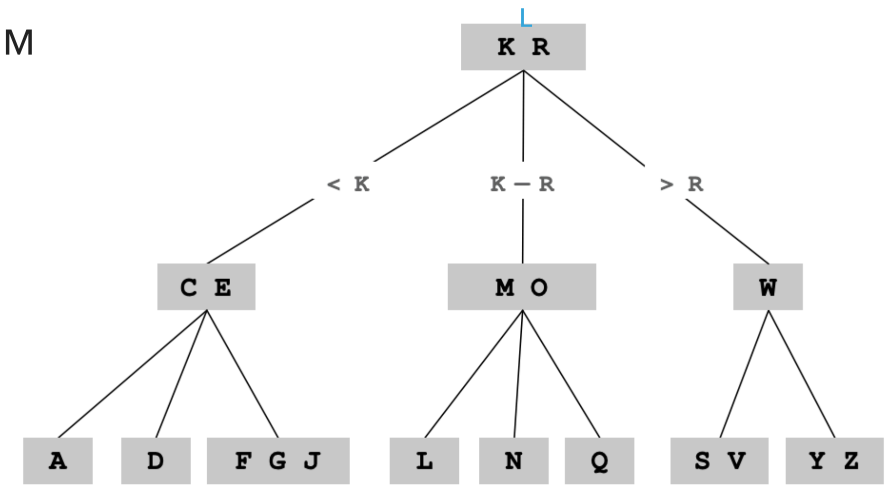
How to search for a key

- Compare search key against (every) key in node.
- Find interval containing search key
- Follow associated link, recursively.

The search algorithm for keys in a 2-3-4 tree directly generalizes the search algorithm for BSTs. To determine whether a key is in the tree, we compare it against the keys at the root. If it is equal to any of them, we have a search hit; otherwise, we follow the link from the root to the subtree corresponding to the interval of key values that could contain the search key. If that link is null, we have a search miss; otherwise we recursively search in that subtree.

Search Hit

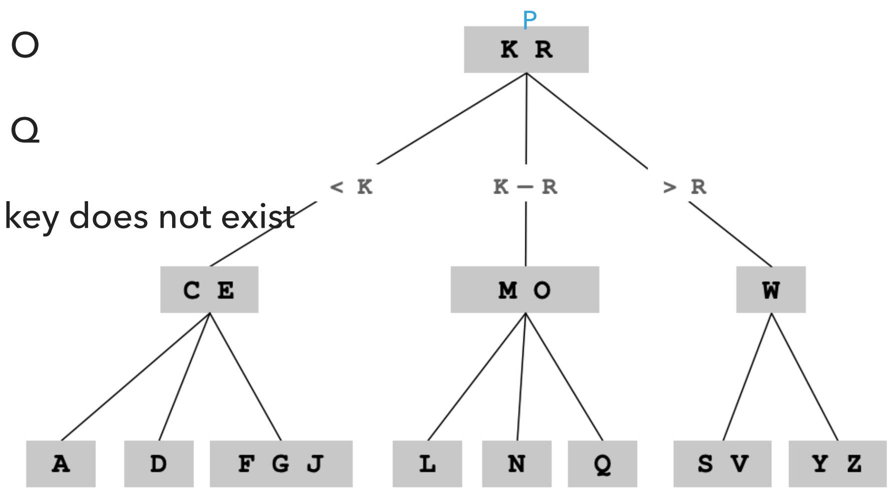
- Example: Search for key L
- L is greater than K and smaller than R
- L is smaller than M
- L is found!



Here is an example of searching for a key that does exist in our 2-3-4 key. L is compared with K and is larger and R and is smaller. So if it exists, it has to be in the 2-3-4 linked with the middle link. L is compared with M and is smaller so if it exists it has to be at the first link. L is compared against L and we have a match!

Search Miss

- Example: Search for key P
- P is greater than K and smaller than R
- P is greater than O
- P is smaller than Q
- Reached null so key does not exist



Here is an example of searching for a key that does NOT exist in our 2-3-4 key. P is compared with K and is larger and R and is smaller. So if it exists, it has to be in the 2-3-4 linked with the middle link. P is compared with M and is larger and with O and is larger so if it exists it has to be at the third link. P is compared against Q and is smaller so if it exists it has to be on the left link of Q. We reached a null link so we have a search miss.

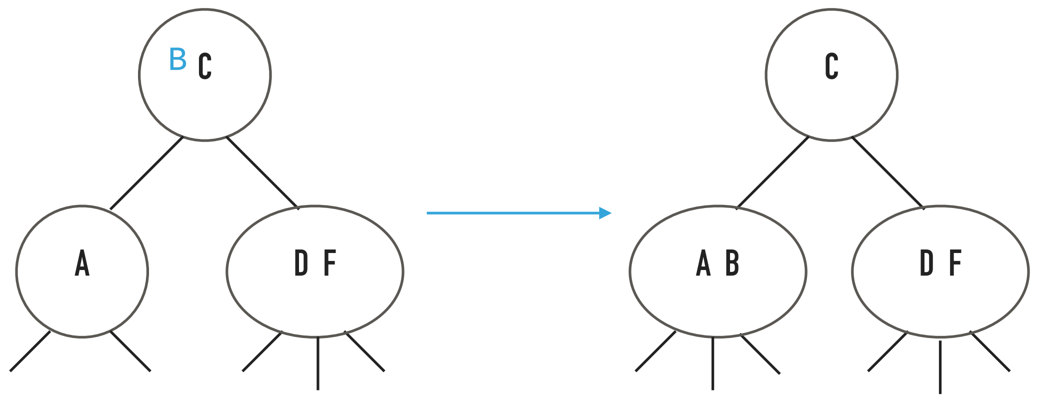
Lecture 18-19: Balanced Search Trees

- 2-3-4 trees
- Search
- Insertion
- Construction
- Performance
- Red-Black Trees

Let's look into how to insert a key (and its associated value) into a 2-3-4 tree.

How to insert into a 2-node

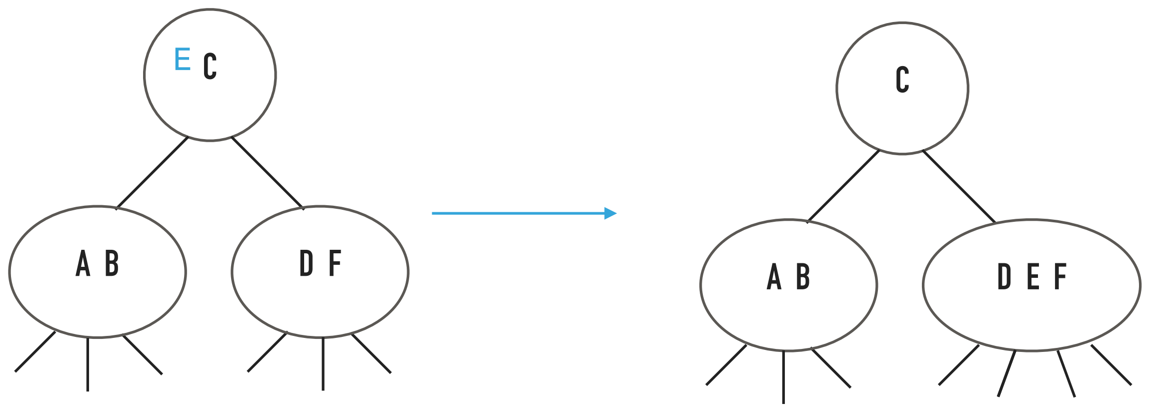
- Search for key to bottom and add new key to 2-node to create a 3-node.
- E.g., insert B in the following 2-3-4 tree.



We will first see how to insert into a 2-node. To insert a new key in a 2-3-4 tree, we might do an unsuccessful search and then hook on the node at the bottom, as we did with BSTs, but the new tree would not remain perfectly balanced. It is easy to maintain perfect balance if the node at which the search terminates is a 2-node: We just replace the node with a 3-node containing its key and the new key to be inserted. See how this plays out when we try to insert B. B is compared with C and we go to the left child of B. B is compared with A and it has to go to its right. We now convert the 2-node to a 3-node that holds both A and B.

How to insert into a 3-node

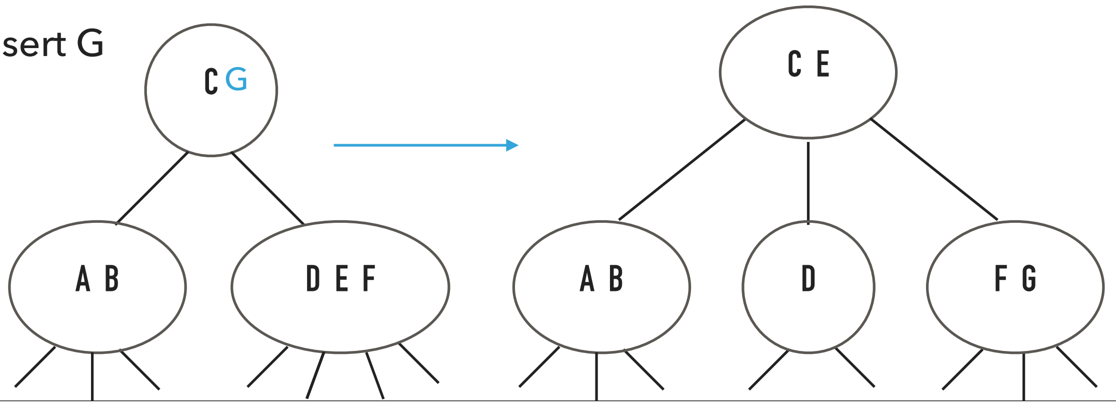
- Search for key to bottom and add new key to 3-node to create a 4-node.
- E.g., insert E in the following 2-3-4 tree.



Similarly to how we insert into a 2-node, we will look into how we insert into a 3-node. We will again search for key to the bottom and if we terminate our search at a 3-node, we will convert into a 4-node. E.g., if we want to insert the key E, we would compare it with C and it is greater so we dive into the right subtree. Then we compare it with D and F and it goes in between, converting the 3-node to a 4-node.

How to insert into a 4-node

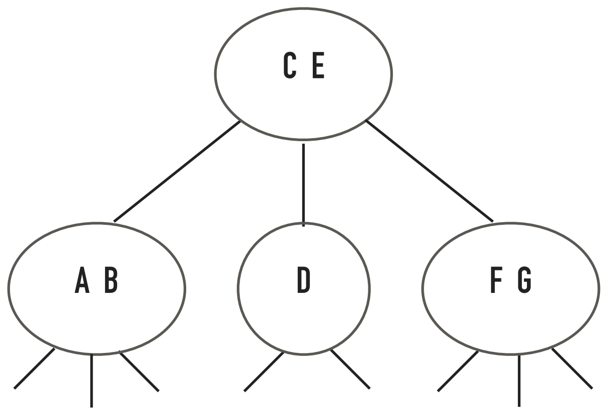
- Search for key to bottom. A node can only have up to three keys/ four children so what do we do?
- We split the node by moving the middle key to its parent node and then add the key to the appropriate child. If the parent node has more than three keys, we split it again.
- Ex. insert G



What about inserting into a 4-node? Here things are a bit different since we cannot have more than three keys/four children in a node. To resolve this we navigate to the node we would need to make the insertion, split it by moving the middle key to its parent node, and add the key of interest to the appropriate child. If the parent node has more than three keys, we split again, and so on. Let's see how this plays out with an example of adding G. G is compared with C and it goes to its right subtree. It is also larger than F but if we were to add it it would turn the 4-node into a 5-node. So we split the D, E, F node by pushing E to its parent (thus, converting the 2-node at the root into a 3-node holding C and E) and D and F become 2-nodes as the second and the third child of the root. We can now add the key G to the appropriate child being F, converting the 2-node to a 3-node.

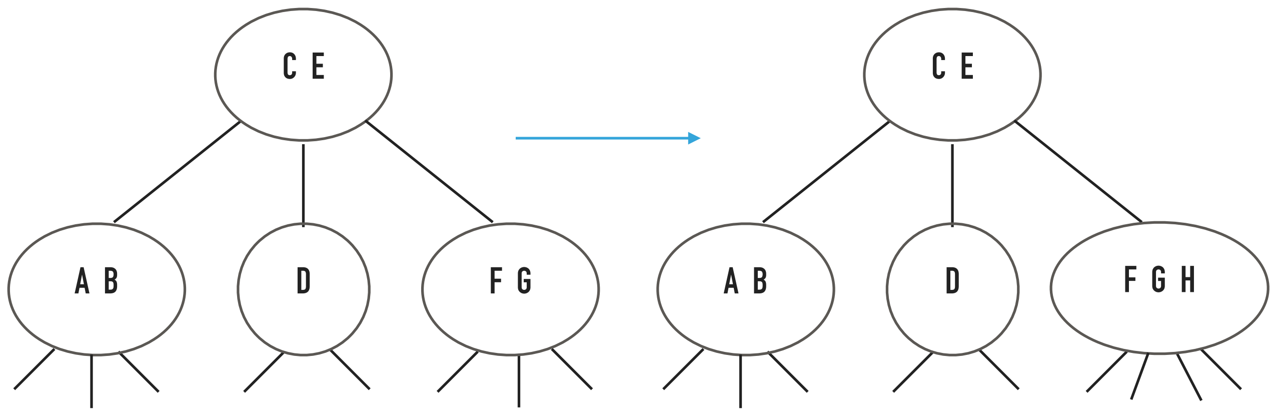
PRACTICE TIME

- Given the following 2-3-4 tree, insert keys H, I, J, K.



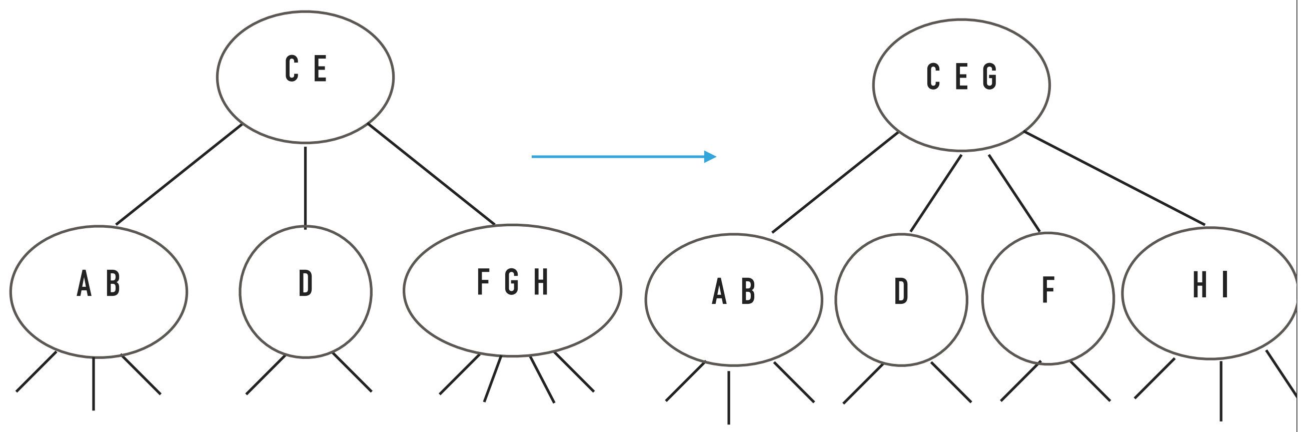
Let's practice. What would happen if you inserted the keys H, I, J, and K to the 2-3-4 tree above?.

ANSWER - Insert H



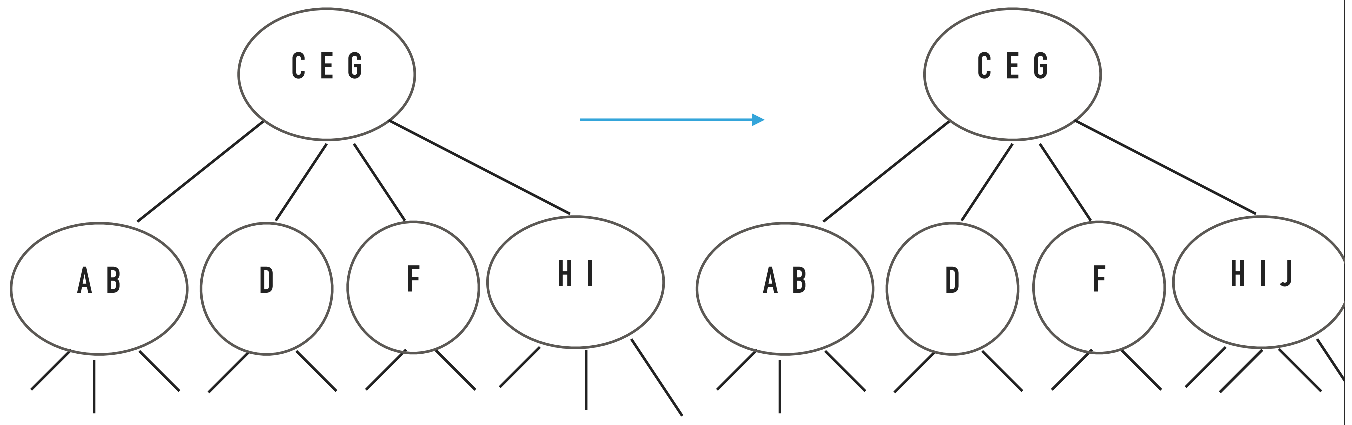
To insert H, we search to the bottom and we find out that it has to go on the right of G. We convert the 3-node to a 4-node.

ANSWER - Insert I



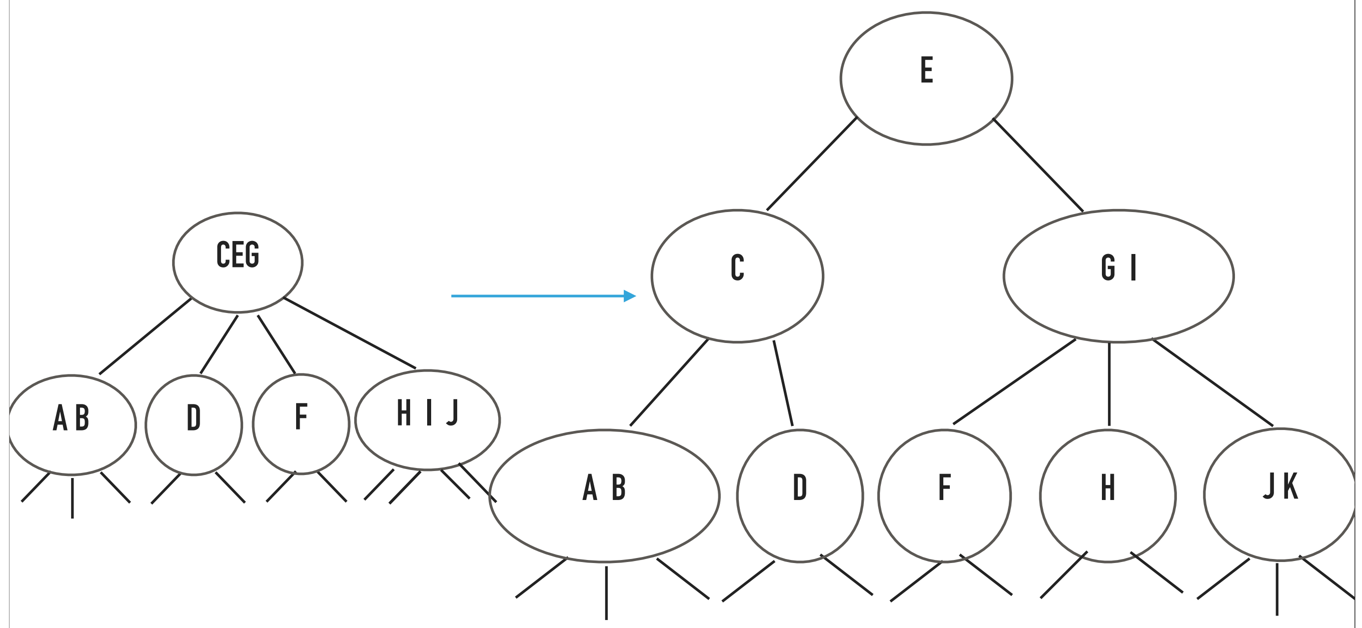
To insert I, we search to the bottom and we find out that it has to go on the right of H. But that would turn the 4-node into a 5-node. So we split the 4-node by pushing the middle key, G, to the parent (i.e. the root) and insert I to the right of H turning into a 3 node.

ANSWER - Insert J

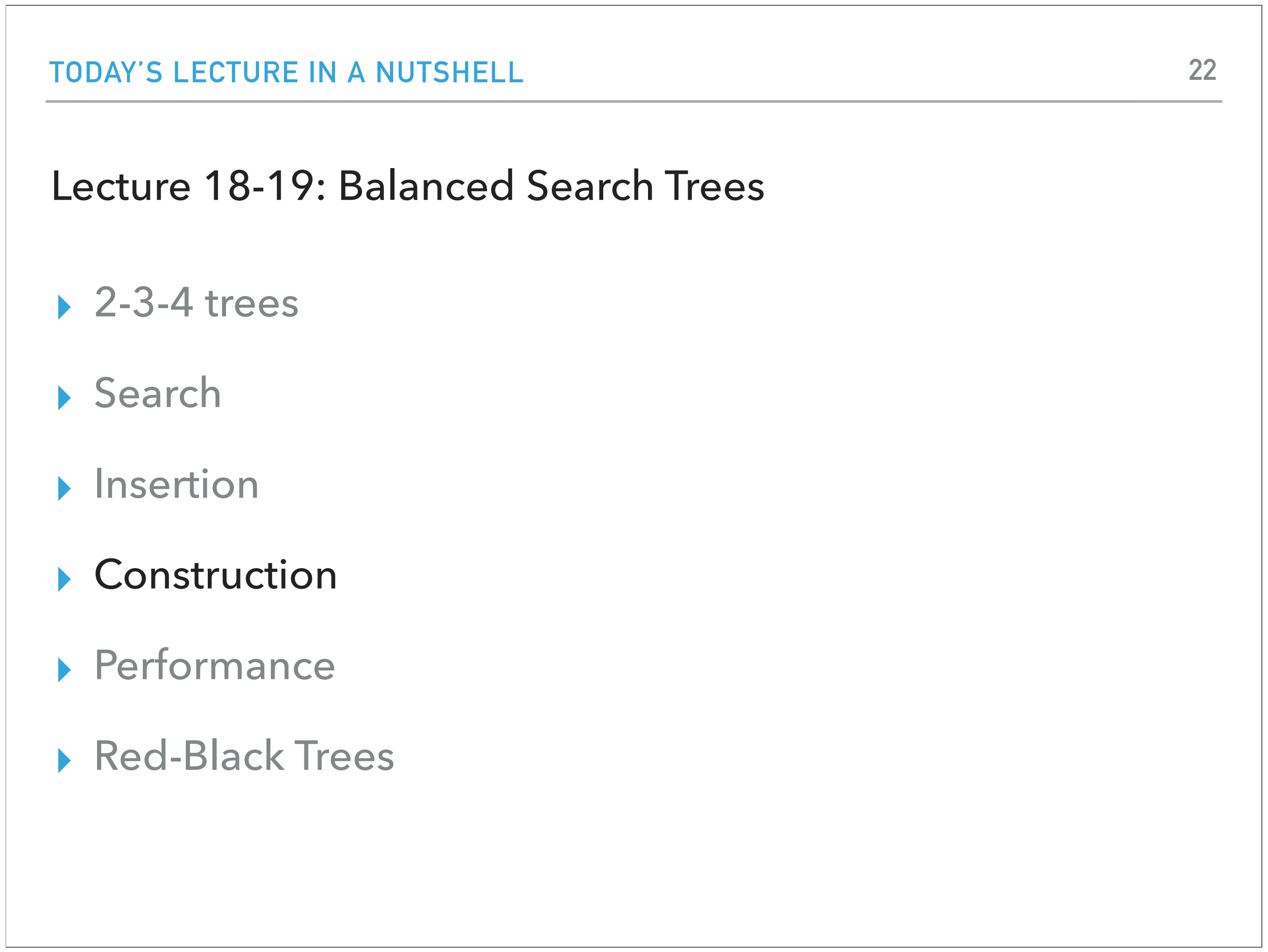


To insert J, we search to the bottom and we find out that it has to go on the right of I turning into a 4-node.

ANSWER - Insert K



To insert K, we search to the bottom and we find out that it has to go on the right of J. But that would turn into a 5-node. So we split the H I J 4-node by moving I to its parent. But that would turn the root (C E G) into a 5-node. So we have to break the root by pushing E one level up, with its left child being C and the right child being G and the newly moved up I. We can now add K with J into a 3-node.



Lecture 18-19: Balanced Search Trees

- 2-3-4 trees
- Search
- Insertion
- Construction
- Performance
- Red-Black Trees

How do we go about constructing a 2-3-4 tree given a number of keys?

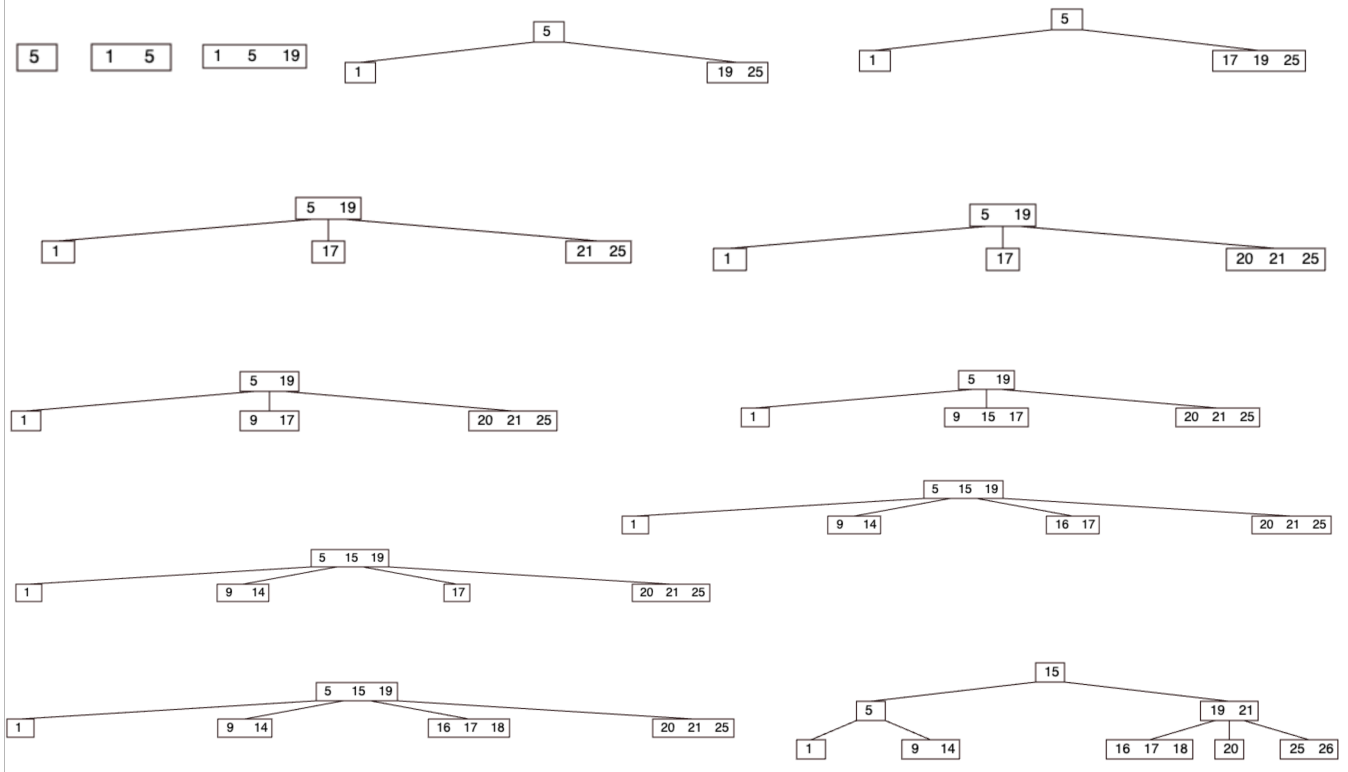
Practice Time - Worksheet

- Draw the 2-3-4 tree that results when you insert the keys:
  5, 1, 19, 25, 17, 21, 20, 9, 15, 14, 16, 18, 26 in that order in an initially empty tree.

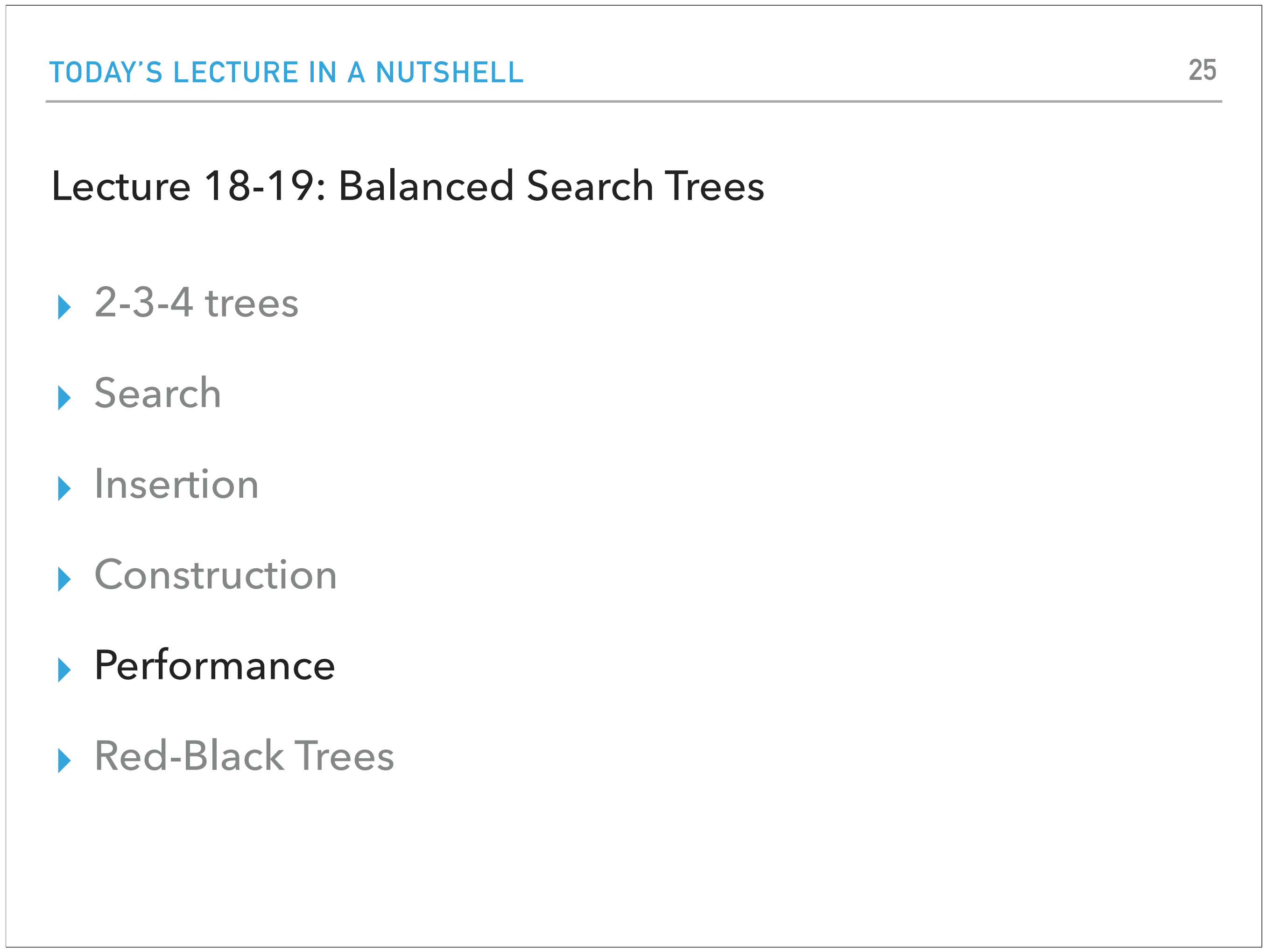
Let's try this problem together.

ANSWER

- 5, 1, 19, 25, 17, 21, 20, 9, 15, 14, 16, 18, 26



These are the intermediate steps you should have gotten.



Lecture 18-19: Balanced Search Trees

- 2-3-4 trees
- Search
- Insertion
- Construction
- Performance
- Red-Black Trees

We talked about how insert and search information in 2-3-4 trees, but how fast are those operations?

Height of 2-3-4 trees

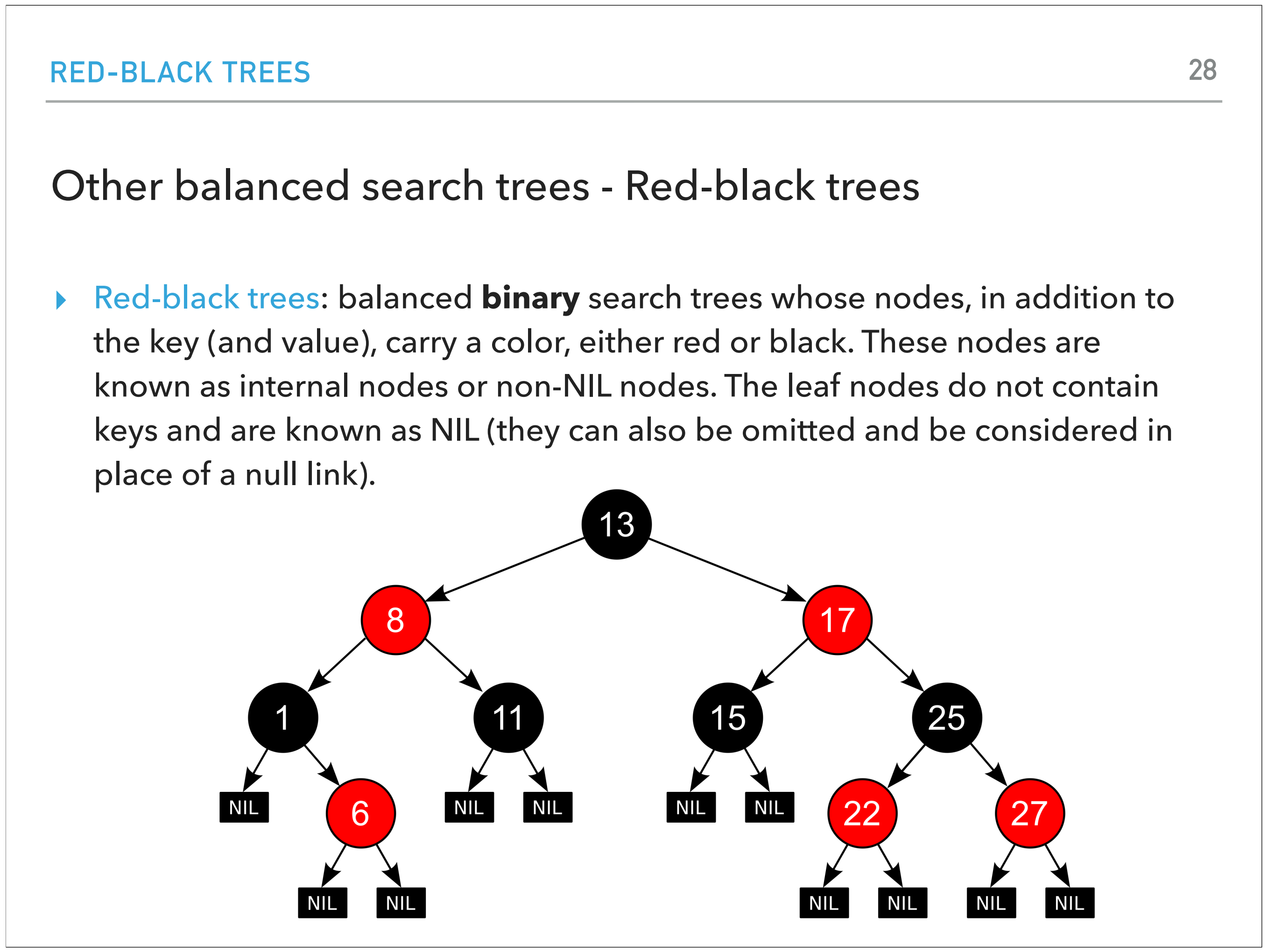
- Worst case: $\log_2(n+1)$ (all 2-nodes).
- Best case: $\log_4(n+1) = 0.5\log_2(n+1)$ (all 4-nodes)
  - That means that storing a million nodes will lead to a tree with height between 10 and 20, and storing a billion nodes to a tree with height between 13 and 27 (not bad!).
- Search and insert (and deletion) are $O(\log n)$!
- But implementation is a pain because of multiple node types and the overhead incurred could make the algorithms slower than standard BST search and insert.

Both depend on the height of the 2-3-4 tree. If we have only 2-nodes (worst case), the height is log_2(n). Best-case is only 4-nodes, that is the height is log_4(n) which is 0.5log_2(n). That means that storing a million nodes will lead to a tree with height between 10 and 20, and storing a billion nodes to a tree with height between 13 and 27 (not bad!).
So we can guarantee that search and insertion are logarithmic but the problem is that implementation is a pain and the overhead incurred could make the algorithms slower than standard BST search and insert. Before we say how we can solve it..

2-3-4 trees are a special type of B-trees

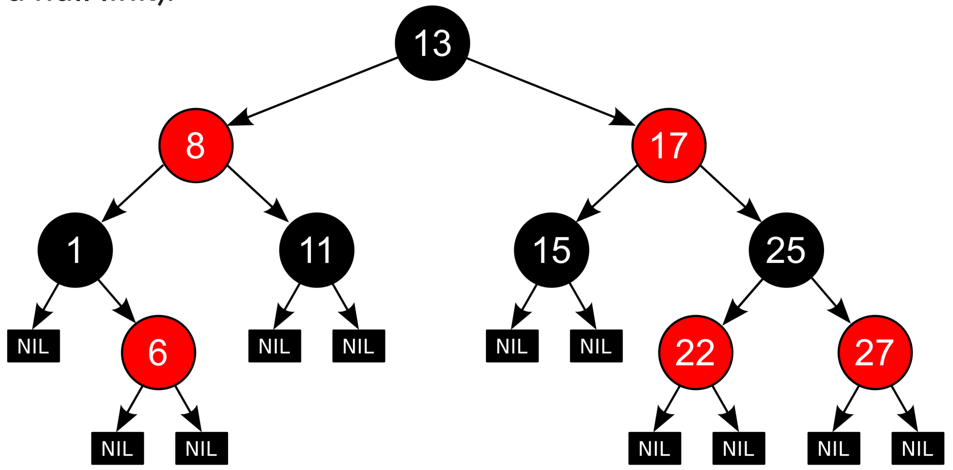
- B-trees: trees of order m where every node has at most m links to children (and equivalently m-1 keys).
  - 2-3-4 search trees are a specific subcategory of B-trees with m=4.
  - Broadly used in file systems and databases.

Just a note that 2-3-4 trees are a special type of B-trees. B-trees of order m have each node having at most m links to children and m-1 keys. That means, that 2-3-4 search trees are just B-trees with m=4. B-trees are not as popular for dictionary implementations but they are broadly used in file systems and databases so you might encounter them if you take electives on the topic. Far more likely, you will encounter red-black trees.



Other balanced search trees - Red-black trees

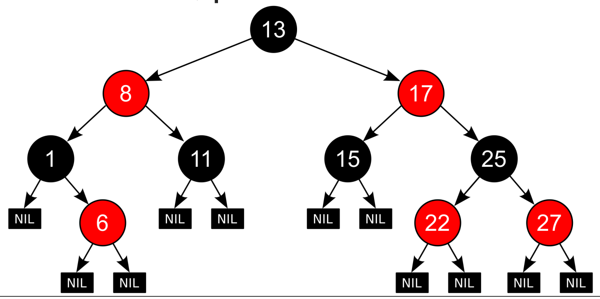
- Red-black trees: balanced **binary** search trees whose nodes, in addition to the key (and value), carry a color, either red or black. These nodes are known as internal nodes or non-NIL nodes. The leaf nodes do not contain keys and are known as NIL (they can also be omitted and be considered in place of a null link).



Most tree-based implementations of dictionaries use other balanced search trees with the most popular being the red-black tree. Red-black trees are balanced binary search trees whose nodes, carry a color (either red or black in addition to the key (and possibly associated value)). These nodes are known as internal/non-NIL. Any null link points to a leaf node or NIL node. Here is an example of a red-black tree.

Properties of red-black Trees

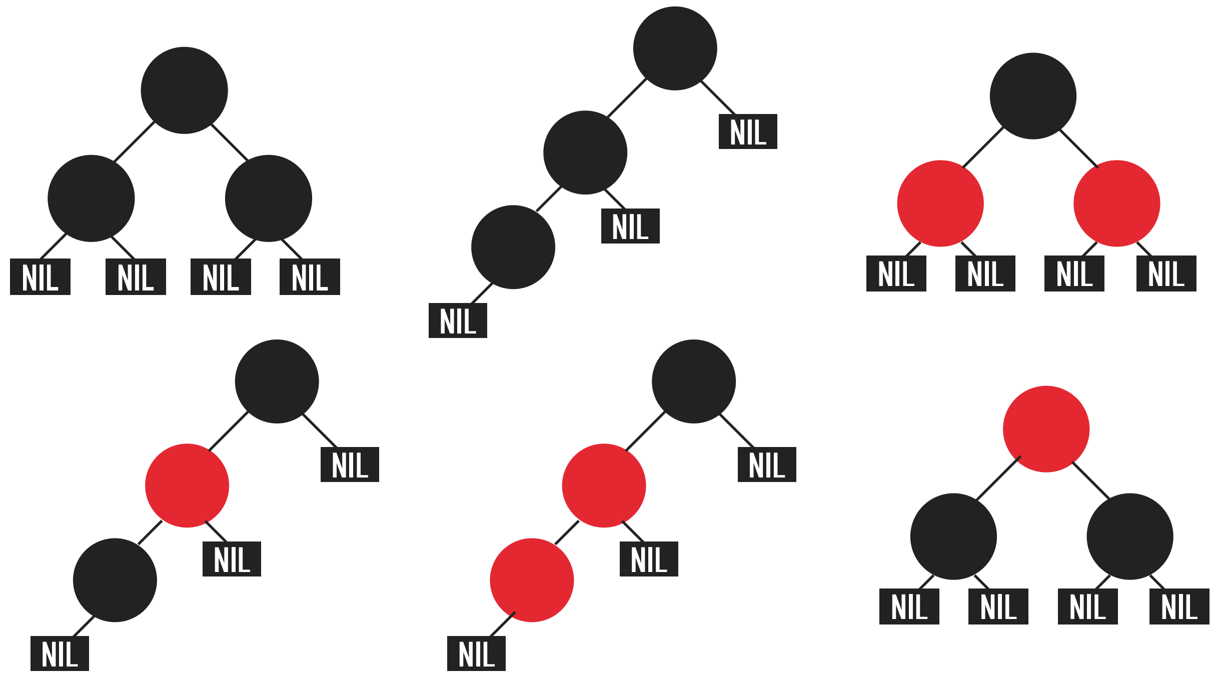
- Every interval node can either be red or black.
- The root has to be black.
- All leaves/NIL nodes are considered black.
- A red node cannot have a red child, i.e. no two consecutive red nodes on any path.
- Any root-leaf (i.e. NIL node) path has the same number of black nodes.



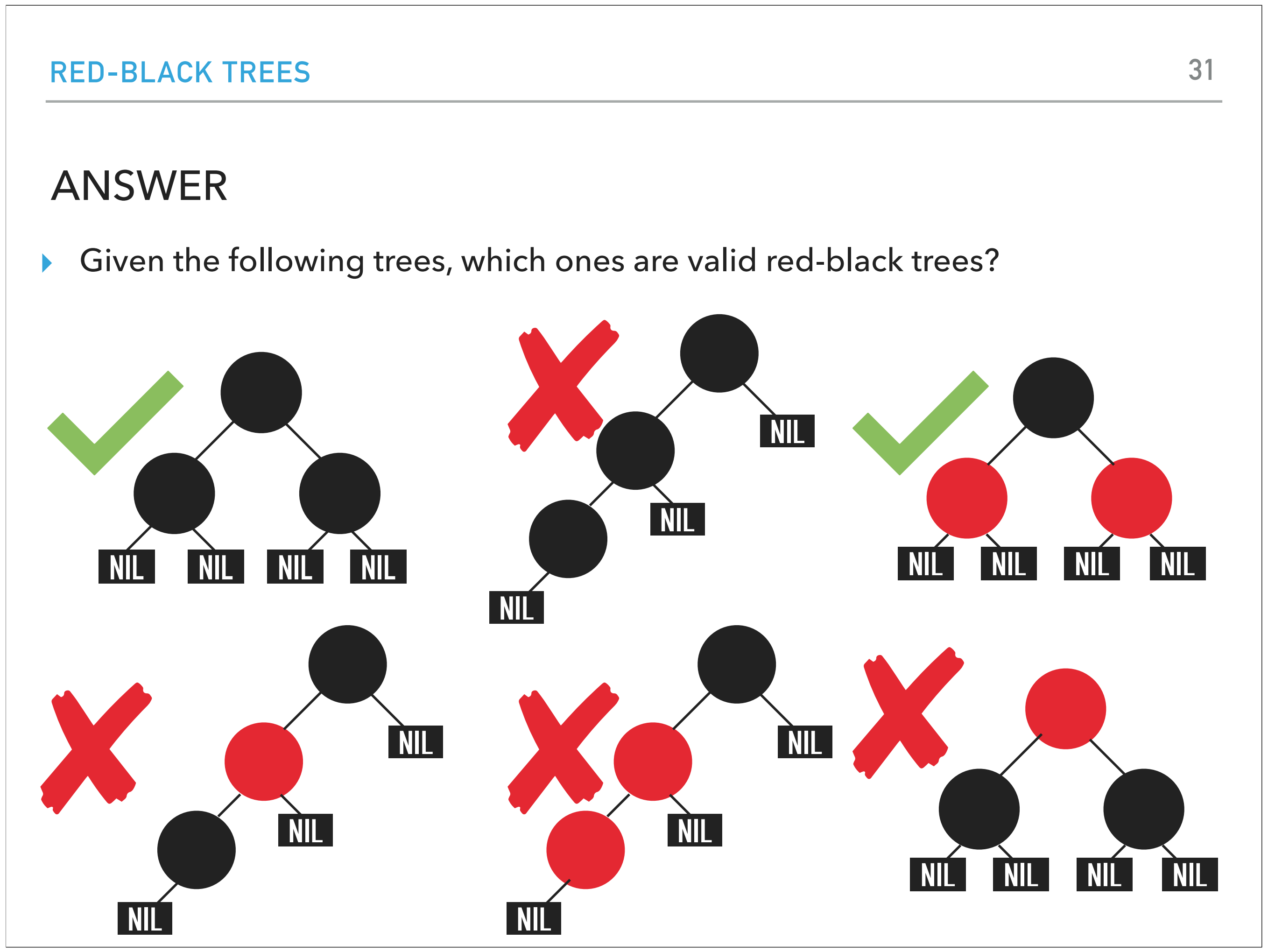
Red-black trees have a number of important properties they need to follow to stay balanced. As we already said, every node is either red or black. The root has to be black. Leaves/NIL nodes are considered black. A red node cannot have a red child, that means you cannot have two consecutive red on any path. And any root to a leaf path (that is to a NIL node) has the same number of black nodes.

Practice Time

- Given the following trees, which ones are valid red-black trees?

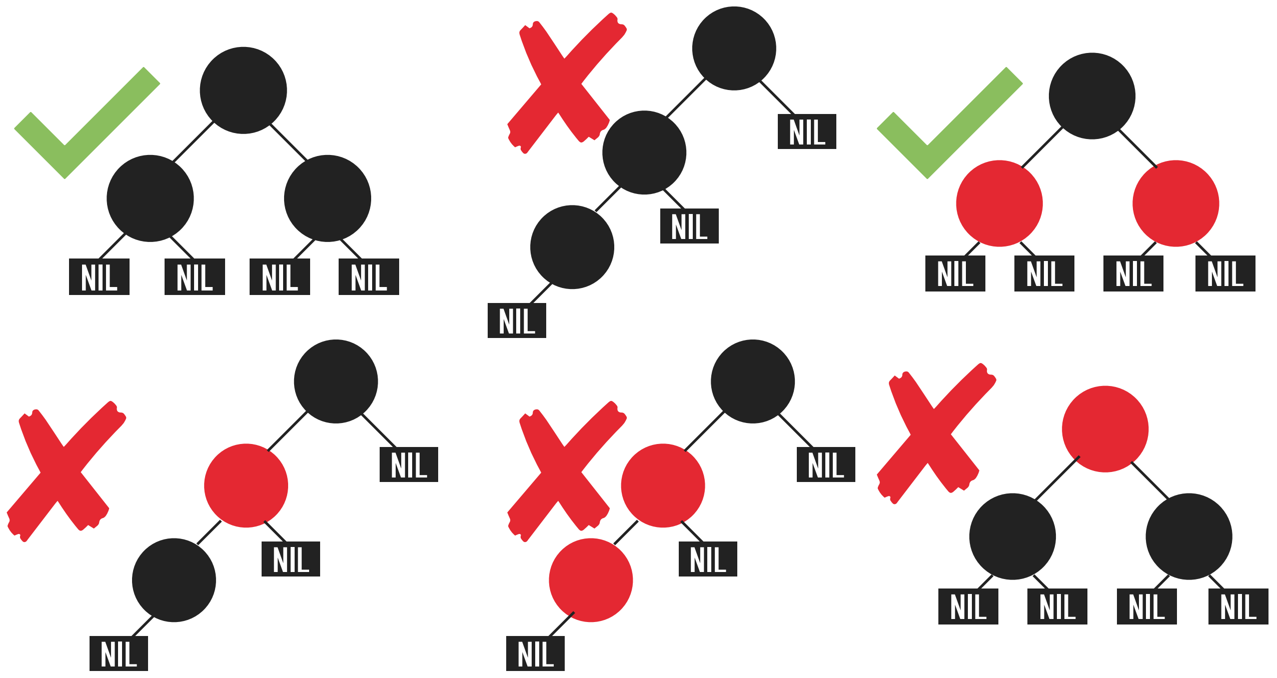


Let's think about this question for a moment.



ANSWER

- Given the following trees, which ones are valid red-black trees?



Only the first and third in the top row are valid red-black trees as they satisfy all properties (root is black, no two red nodes in a row, and number of black nodes from root to NIL nodes is the same for any path).

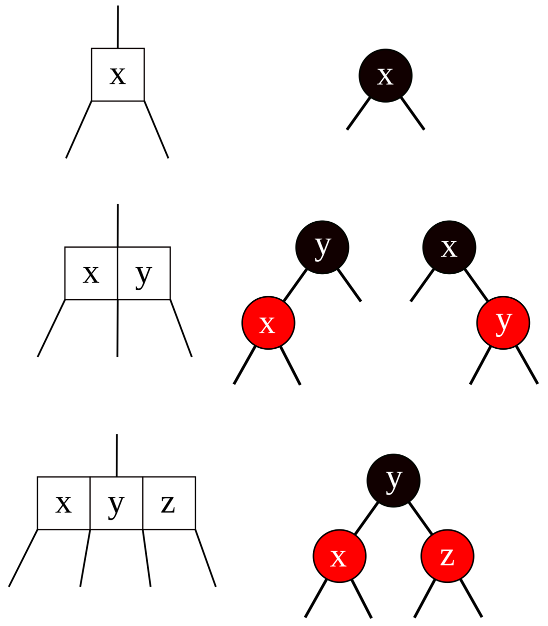
Height of red-black trees

- Stay tuned for proofs in CS140!
- The height is between $\log(n+1) \leq h \leq 2log(n+1)$.
- This implies that search (and insertion/deletion) is $O(\log n)$!

So the advantage of red-black trees is twofold: 1) they are binary which means we don't have to worry about different types of nodes. 2) they still offer O(logn) search guarantees because their height is bounded to O(long).

Red-black trees and 2-3-4 search trees

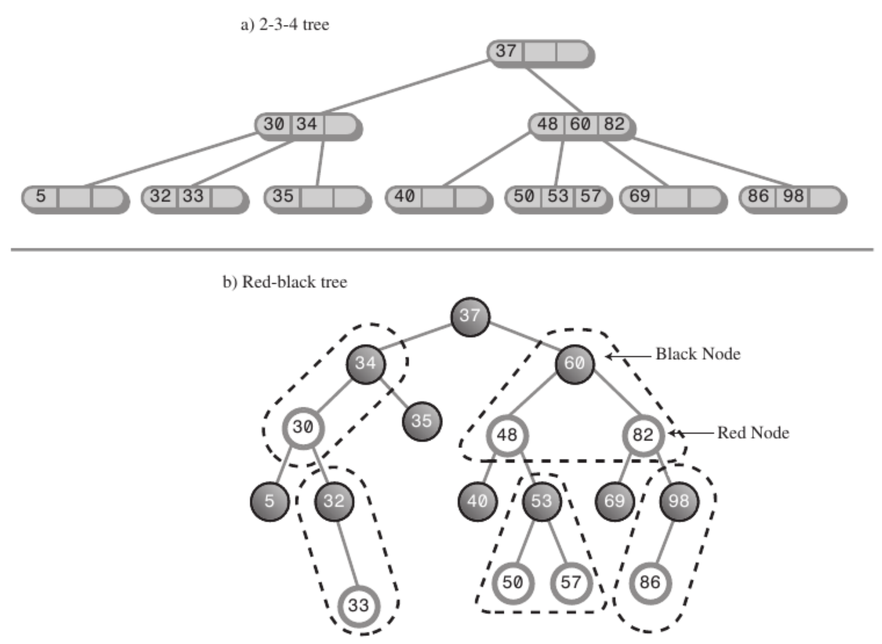
- There is a (not 1-1) correspondence between 2-3-4 search trees and red-black trees.



https://en.m.wikipedia.org/wiki/Red%E2%80%93black_tree#/media/File%3ARed-black_trees_and_2%E2%80%933%E2%80%934_trees.svg

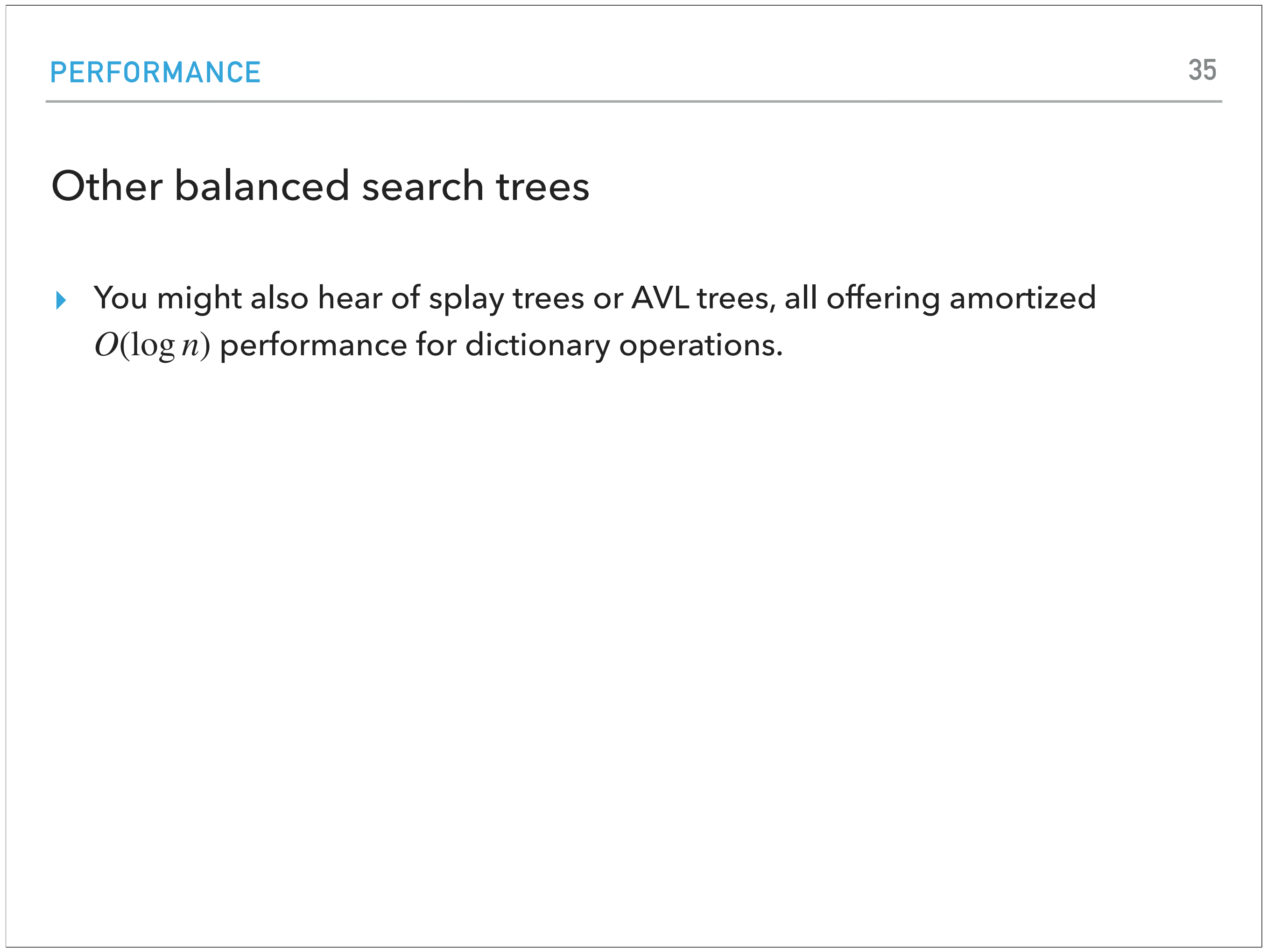
We won't be seeing how to do insertions in red-black trees as they can be complex, but I wanted to draw your attention to the fact that there is a correspondence between red-black trees and 2-3-4 search trees and you can convert each 2-, 3-, or 4-node to an equivalent red-black tree. Note that this correspondence is not 1-1 since a 3-node has two possible outcomes.

2-3-4 search tree and red-black tree correspondence



Lafore - Data Structures and Algorithms in Java

But given a 2-3-4 search tree, you could create a valid red-black tree as in the example above (note that this textbook does not visualize the NIL nodes).



Other balanced search trees

- You might also hear of splay trees or AVL trees, all offering amortized $O(\log n)$ performance for dictionary operations.

Finally, you might also hear of splay trees or AVL trees which are different types of balanced search trees. which offer amortized O(long) performance for standard dictionary operations like search/insertion/deletion.

Summary for dictionary operations

	Worst case			Average case		
	Search	**Insert**	**Delete**	**Search**	**Insert**	**Delete**
BST	n	n	n	$\log n$	$\log n$	$\sqrt{n}$
Balanced search trees	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$

We will start building this table further over the next week.

Readings:

- Recommended Textbook: Chapter 3.3 (Pages 424-447)
- Website:
  - https://algs4.cs.princeton.edu/33balanced/
- Visualization:
  - https://yongdanielliang.github.io/animation/web/24Tree.html

Worksheet:

- Lecture worksheet

Problem 1

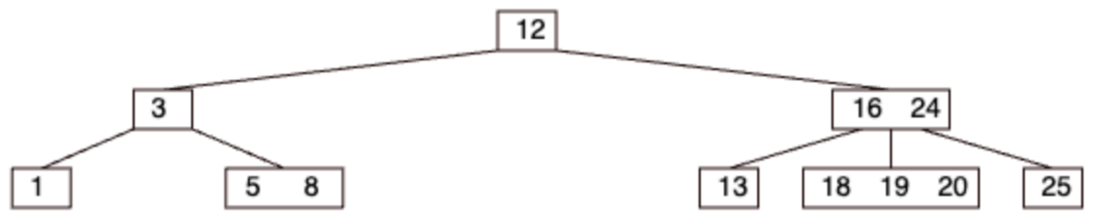
- Draw the 2-3-4 tree that results when you insert the keys 25, 12, 16, 13, 24, 8, 3, 18, 1, 5, 19, 20 in that order into an initially empty tree.

Problem 2

- Find an insertion order for the keys 19, 5, 1, 18, 3, 8, 24, 13 that leads to a 2-3-4 tree of height 1.

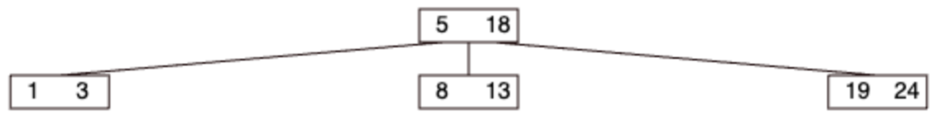
ANSWER 1

- Draw the 2-3-4 tree that results when you insert the keys 25, 12, 16, 13, 24, 8, 3, 18, 1, 5, 19, 20 in that order into an initially empty tree.



ANSWER 2

- Find an insertion order for the keys 19, 5, 1, 18, 3, 8, 24, 13 that leads to a 2-3-4 tree of height 1.
- Example of insertion order: 5 1 13 24 18 3 8 19


5 18
1 3
8 13
19 24