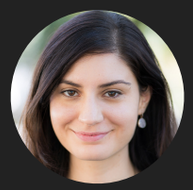
SEARCHING

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

19: 2–3 Search Trees



Alexandra Papoutsaki
she/her/hers

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 19: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance

Today we will continue with a special category of search trees called 2-3 search trees.

Visualization of insertion into a binary search tree

- 255 insertions in random order.

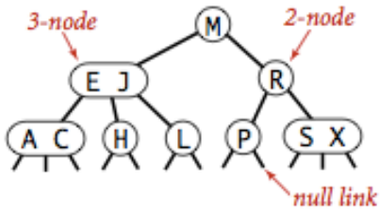
The main limitation of basic binary search trees is that they can become imbalanced. Here is a visualization of 255 random insertions. The tree can grow unpredictably and become quite long which will result to longer search times when we search for a value given a key.

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 tree



Anatomy of a 2-3 search tree

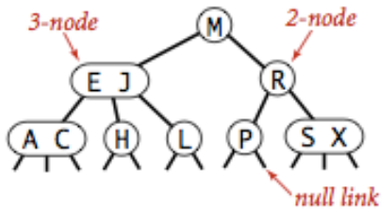
- Definition: A 2-3 tree is either empty or a
  - 2-node: one key (and associated value) and two links, a left to a 2-3 search tree with smaller keys, and a right to a 2-3 search tree with larger keys (similarly to standard BSTs), or a
  - 3-node: two keys (and associated values) and three links, a left to a 2-3 search tree with smaller keys than first key, a middle to a 2-3 search tree with keys between the node's keys, and a right to a 2-3 search tree with larger keys than the second key.
- Symmetric order: In-order traversal yields keys in ascending order.
- Perfect balance: Every path from root to null link (empty tree) has the same length.

To do so, we will consider 2-3 search trees. A 2-3 tree is either empty or 2- or 3-node. A 2-node is a node with one key (and its associated value) and two links, each to a 2-3 search tree. The left child contains smaller keys and the right tree larger keys (similar to the standard binary search tree). A 3-node contains two keys (and their associated values) and three links: a left to a 2-3 tree with smaller keys that the first key, a middle to a 2-3 search tree with keys in between the node's keys, and a right one to a 2-3 search tree with larger keys than the second key.

A 2-3 tree follows the symmetric order and has perfect balance. Symmetric order implies that if we were to do an in-order traversal, we would get keys in ascending order. Perfect balance implies that every path from root to a null link has the same length.

Example of a 2-3 tree

- 2-node, business as usual with BSTs.
  - (e.g., EJ are smaller than M and R is larger than M).
- In 3-node,
  - left link points to 2-3 search tree with smaller keys than first key,
    - (e.g., AC are smaller than E.)
  - middle link points to 2-3 search tree with keys between first and second key,
    - (e.g. H is between E and J.)
  - right link points to 2-3 search tree with keys larger than second key.
    - (e.g, L is larger than J).



Anatomy of a 2-3 search tree

Here is an example of a 2-3 tree. 2-nodes are like regular BSTs, e.g., EJ are smaller than M and R is larger than M. For 3-nodes, we said that the left link points to smaller keys (e.g., AC is smaller than E), middle points to keys in between (e.g., H is between E and J), and right link points to keys larger than second key, e.g., L is larger than J.

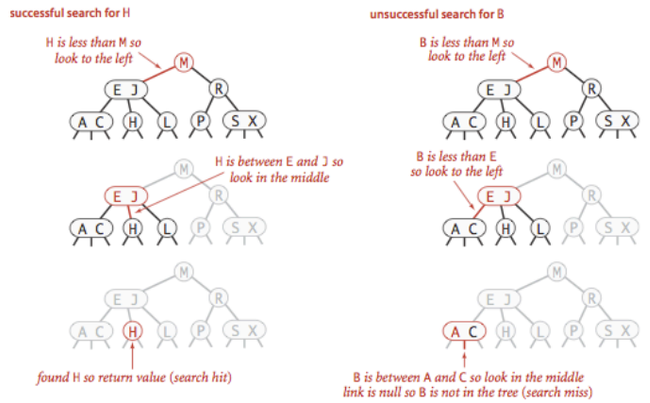
Lecture 19: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance

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

How to search for a key

- Compare search key against (every) key in node.
- Find interval containing search key (left, potentially middle, or right).
- Follow associated link, recursively.



Search hit (left) and search miss (right) in a 2-3 tree

The search algorithm for keys in a 2-3 search 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.

3.3 2–3 Tree Demo

- *search*
- *insertion*
- *construction*

Here is a visualization of search in 2-3 search trees.

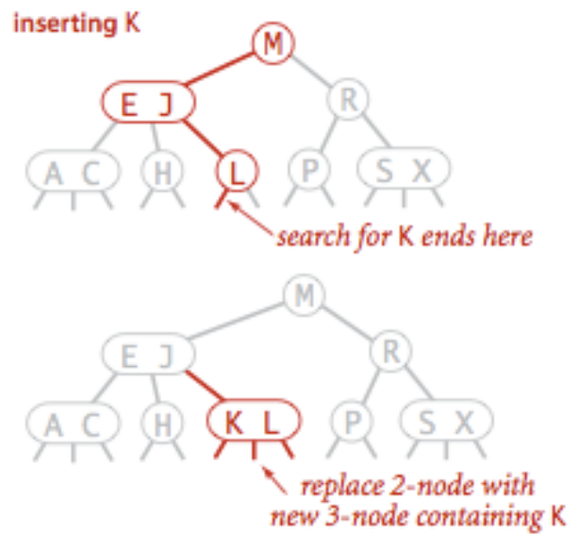
Lecture 19: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance

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

How to insert into a 2-node at bottom

- Search for key and add new key to 2-node to create a 3-node.



Insert into a 2-node

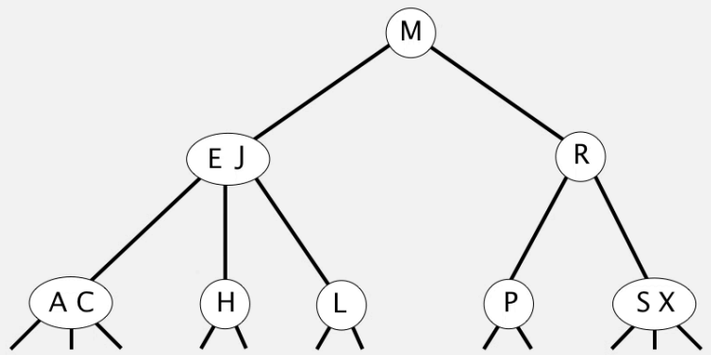
We will first see how to insert into a 2-node at bottom. To insert a new key in a 2-3 tree, we might do an unsuccessful search and then hook on the node at the bottom, as we did with BSTs, but the new tree would not remain perfectly balanced. It is easy to maintain perfect balance if the node at which 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.

2-3 tree demo: insertion

Insert into a 2-node at bottom.

- Search for key, as usual.
- Replace 2-node with 3-node.

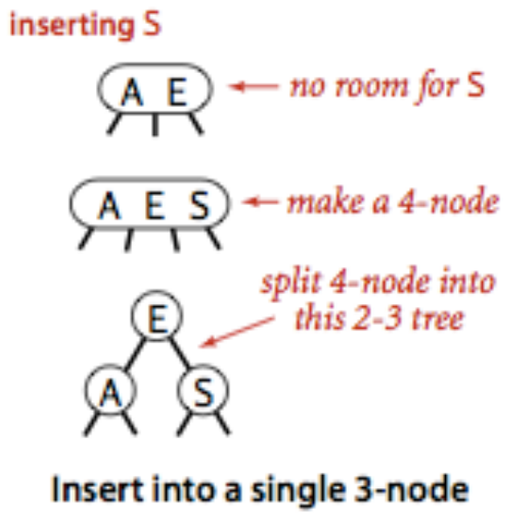
insert K



This is a visualization of what we just saw.

How to insert into a tree consisting of a single 3-node

- Add new key to 3-node to create a temporary 4-node.
- Move middle key in 4-node into parent.
- Split 4-node into two 2-nodes.
- Height went up by 1.

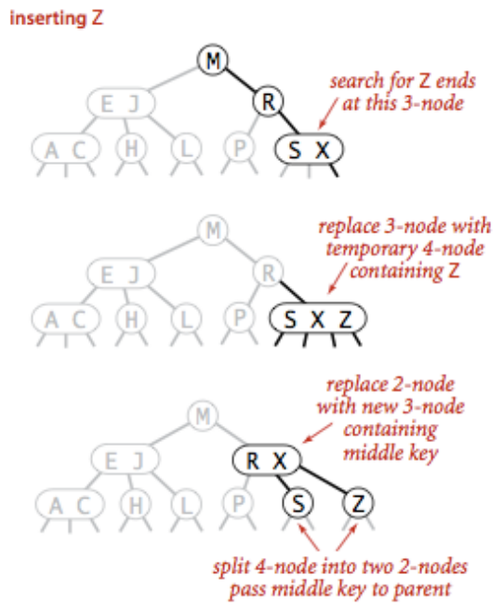


Insert into a single 3-node

How do we insert into a tree consisting of a single 3-node? Such a tree has two keys, but no room for a new key in its one node. To be able to perform the insertion, we temporarily put the new key into a 4-node, a natural extension of our node type that has three keys and four links. Creating the 4-node is convenient because it is easy to convert it into a 2-3 tree made up of three 2-nodes, one with the middle key (at the root), one with the smallest of the three keys (pointed to by the left link of the root), and one with the largest of the three keys (pointed to by the right link of the root). Before the insertion, the height was 0 and after is 1. That's a trivial case but worth illustrating the height growth in 2-3 search trees.

How to insert into a 3-node whose parent is a 2-node

- Add new key to 3-node to create a temporary 4-node.
- Split 4-node into two 2-nodes and pass middle key to parent.
- Replace 2-node parent with 3-node.

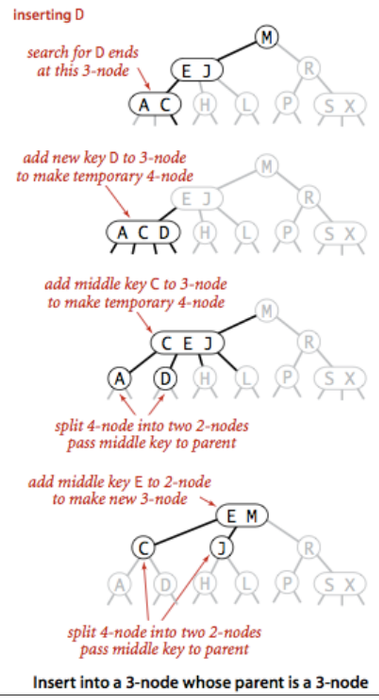


Insert into a 3-node whose parent is a 2-node

How do we insert into a 3-node whose parent is a 2-node? In this case, we can still make room for the new key while maintaining perfect balance in the tree, by making a temporary 4-node as just described, then splitting the 4-node as just described, but then, instead of creating a new node to hold the middle key, moving the middle key to the nodes parent.

How to insert into a 3-node whose parent is a 3-node

- Add new key to 3-node to create a temporary 4-node.
- Split 4-node into two 2-nodes and pass middle key to parent creating a temporary 4-node.
- Split 4-node into two 2-nodes and pass middle key to parent.
- Repeat up the tree, as necessary.

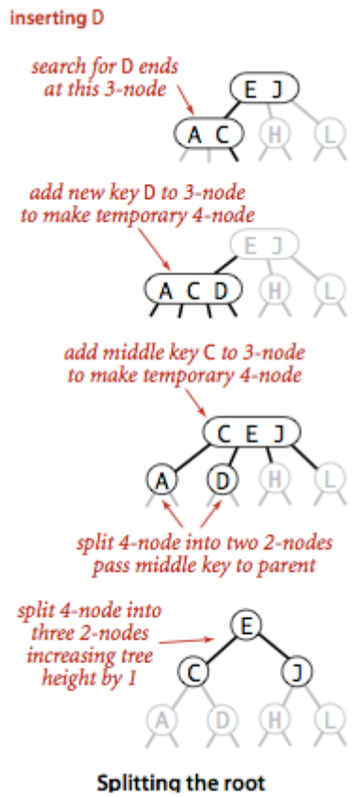


Insert into a 3-node whose parent is a 3-node

Now suppose that the search ends at a 3-node whose parent is a 3-node. Again, we make a temporary 4-node as just described, then split it and insert its middle key into the parent. The parent was a 3-node, so we replace it with a temporary new 4-node containing the middle key from the 4-node split. Then, we perform precisely the same transformation on that node. That is we split the new 4-node and insert its middle key into its parent. Extending to the general case is clear: we continue up the tree, splitting 4-nodes and inserting their middle keys in their parents until reaching a 2-node, which we replace with a 3-node that does not to be further split, or until reaching a 3-node at the root.

Splitting the root

- If end up with a temporary 4-node root, split into three 2-nodes.
- Increases height by 1 but perfect balance is preserved.



Splitting the root

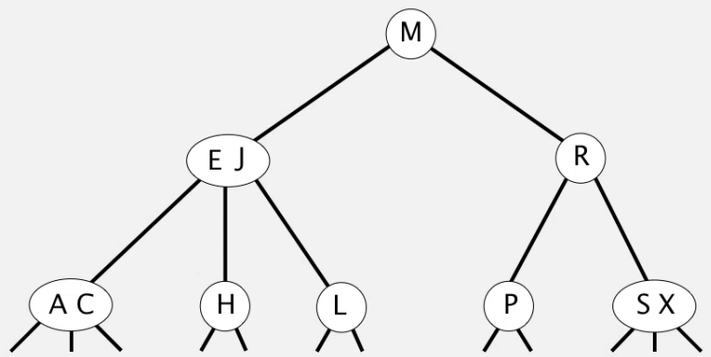
If we have 3-nodes along the whole path from the insertion point to the root, we end up with a temporary 4-node at the root. In this case we split the temporary 4-node into three 2-nodes, increasing the height of the tree by 1.

2-3 tree demo: insertion

Insert into a 2-node at bottom.

- Search for key, as usual.
- Replace 2-node with 3-node.

insert K



This is a visualization of what we just saw.

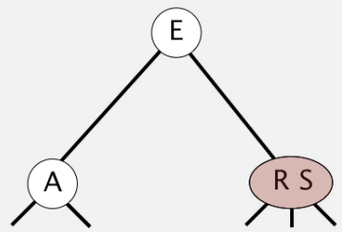
Lecture 19: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance

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

2-3 tree demo: construction

insert R



This is how it would work.

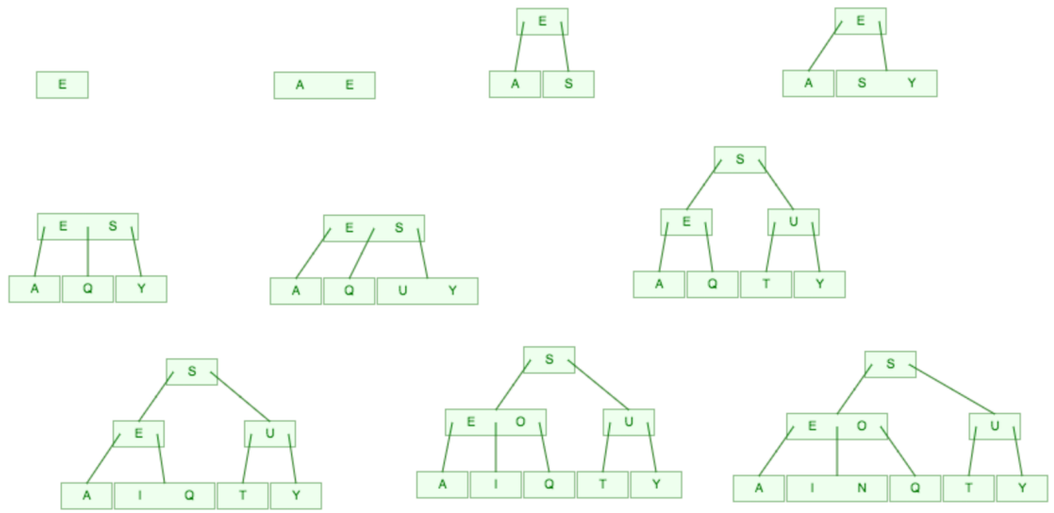
Practice Time - Worksheet #19

- Draw the 2-3 tree that results when you insert the keys:
  E A S Y Q U T I O N in that order in an initially empty tree.

Let's try this problem together.

ANSWER

- EASYQUTION



https://www.cs.usfca.edu/~galles/visualization/BTree.html

These are the intermediate steps you should have gotten.

Lecture 19: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance

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

Height of 2-3 search trees

- Worst case: $\log n$ (all 2-nodes).
- Best case: $\log_3 n = 0.631 \log n$ (all 3-nodes)
  - That means that storing a million nodes will lead to a tree with height between 12 and 20, and storing a billion nodes to a tree with height between 19 and 30 (not bad!).
- Search and insert are $O(\log n)$!
- But implementation is a pain and the overhead incurred could make the algorithms slower than standard BST search and insert.
- We did provide insurance against a worst case but we would prefer the overhead cost for that insurance to be low. Stay tuned!

Both depend on the height of the 2-3 search tree. If we have only 2-nodes (worst case), the height is log_2(n). Best-case is only 3-nodes, that is the height is log_3(n). That means that storing a million nodes will lead to a tree with height between 12 and 20, and storing a billion nodes to a tree with height between 19 and 30 (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. We did provide insurance against a worst case but we would prefer the overhead cost for that insurance to be low. Stay tuned!

Summary for dictionary operations

	Worst case			Average case		
	Search	Insert	Delete	Search	Insert	Delete
BST	n	n	n	$\log n$	$\log n$	$\sqrt{n}$
2-3 search trees	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$

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

Readings:

- Recommended Textbook: Chapter 3.3 (Pages 424-447)
- Website:
  - https://algs4.cs.princeton.edu/33balanced/
- Visualization:
  - https://www.cs.usfca.edu/~galles/visualization/BTree.html (for 2-3 trees)

Worksheet:

- Lecture 19 worksheet

Problem 1 (Problem 3.3.2 in the book)

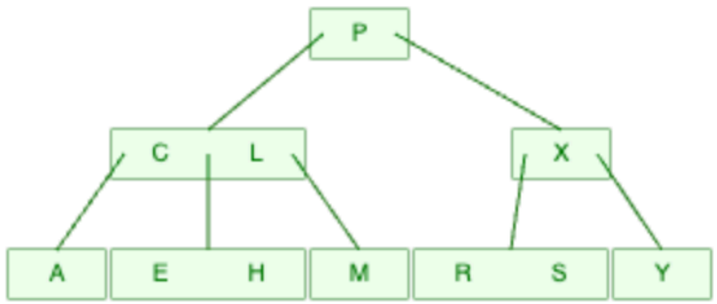
- Draw the 2-3 tree that results when you insert the keys Y, L, P, M, X, H, C, R, A, E, S) in that order into an initially empty tree.

Problem 2 (Problem 3.3.3 in the book)

- Find an insertion order for the keys S, E, A, R, C, H, X, M that leads to a 2-3 search tree of height 1.

ANSWER 1 (Problem 3.3.2 in the book)

- Draw the 2-3 tree that results when you insert the keys Y, L, P, M, X, H, C, R, A, E, S) in that order into an initially empty tree.



ANSWER 2 (Problem 3.3.3 in the book)

- Find an insertion order for the keys S, E, A, R, C, H, X, M that leads to a 2-3 search tree of height 1.
- Insertion order: E A M X R C H S
-