THINK BIG WE DO℠

CSC 212
Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

2-3 Trees

Housekeeping

Election Day / Veteran's Day

- Nov 7-11
- Class only meets Thursday, Nov 10
- Assignment 4 Due
- Lab 9: Balancing Act Due
    - In-person labs are canceled

Term Project

2-3 Trees

Allow 1 or 2 keys per node

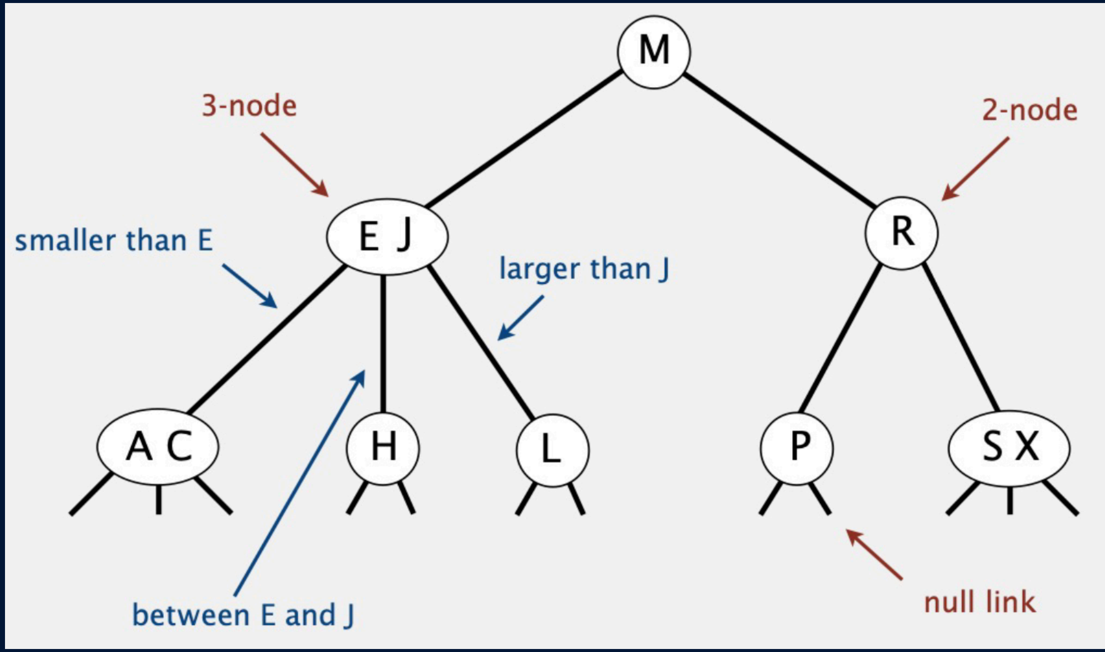
- 2-node: one key, two children
- 3-node: two keys, three children

Symmetric order

- Inorder traversal yields keys in ascending order

Perfect Balance

- Every path from the root null link has same length



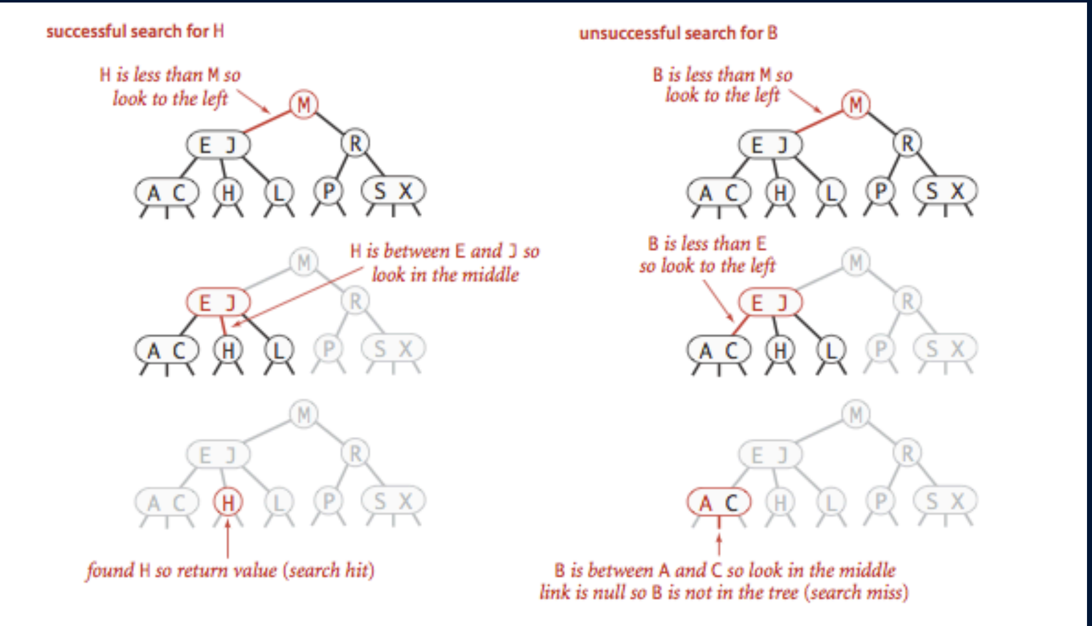
$$tree = \{M, E, R, P, S, X, A, J, C, H, L\}$$

search

Compare search key against key(s) in node

Find Interval containing search key

Follow associated link (recursively)

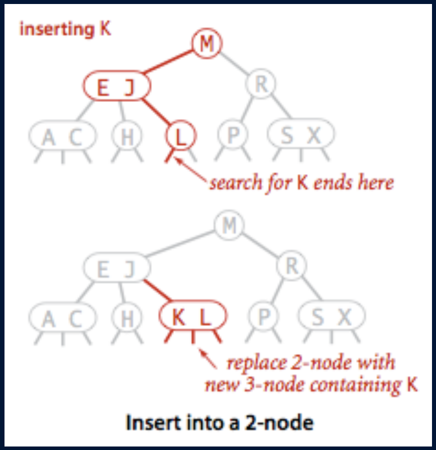


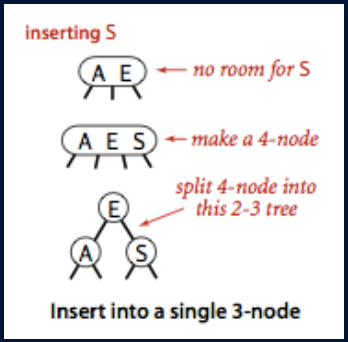
Search hit (left) and search miss (right) in a 2-3 tree

search for H & B

$$tree = \{M, E, R, P, S, X, A, J, C, H, L\}$$

insert





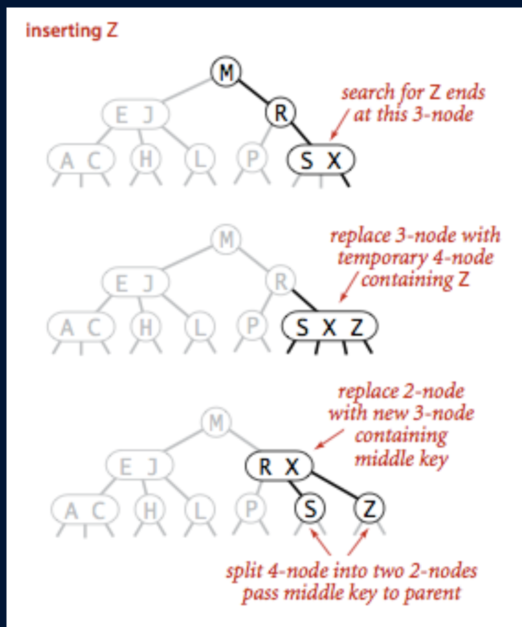
$Insert\ into\ a\ tree\ consisting\ of\ a\ single\ 3-node$

$Insert\ into\ a\ 2-node$

- add a new key to 2-node to create a 3-node

- create a 4-node and break it down into three 2-nodes

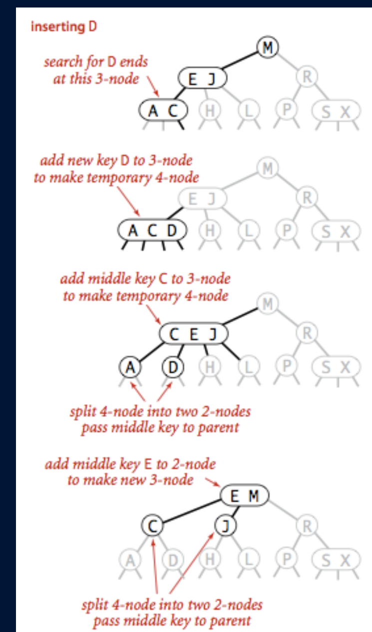
insert, con't



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

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

- create a temporary 4-node, remove middle child from 4-node, and add to parent to create 3-node



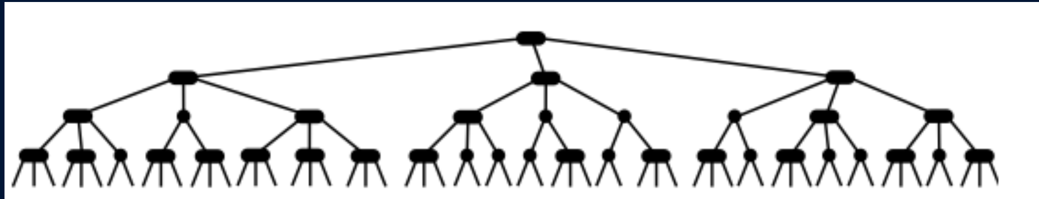
Insert into a 3-node whose parent is a 3-node

$Insert\ into\ a\ tree\ consisting\ of\ a\ single\ 3-node$

- create a 4-node and break it down into three 2-nodes

Performance

Perfect balance: Every path from the root to null has the same length



Typical 2-3 tree built from random keys

Tree height:

- Min: $log_3 \ n \approx 0.631 \ \ log_2 \ n$
- Max: $log_2 \ n$
- Between 12 and 20 for a million nodes
- Between 18 and 30 for a billion nodes

Bottomline: Guarenteed logarithmic performance for search and insert

Summary

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search	insert	delete		
sequential search (unordered list)	n	n	n	n	n	n		`equals()`
binary search (ordered array)	$\log n$	n	n	$\log n$	n	n	✔	`compareTo()`
BST	n	n	n	$\log n$	$\log n$	$\sqrt{n}$	✔	`compareTo()`
2-3 tree	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	✔	`compareTo()`

but hidden constant c is large (depends on implementation)

Implementation

Direct implementation is complicated, because:

- Maintaining multiple node types is cumbersome
- Need Multiple compares to move down tree
- Need to move back up the tree to split 4-nodes
- Large number of cases for splitting

```
void put(Key key, Value val) {
  Node x = root;
  while (x.getCorrectChild(key) != null) {
    x = x.getCorrectChildKey();
    if (x.is4Node()) x.split();
  }
  if (x.is2Node()) x.make3Node(key, val)
  else if (x.is3Node()) x.make4Node(key, val)
}
```

Bottomline: Could do it, but there’s a better way