CS062
DATA STRUCTURES AND ADVANCED PROGRAMMING

27: 2–3 Search Trees

Alexandra Papoutsaki

Mark Kampe
Lecture 27: 2-3 Search Trees

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

Some slides adopted from Algorithms 4th Edition or COS226
The story so far

- The symbol table is a fundamental data type.
- Naive implementations (arrays/linked lists sorted or unsorted) are way too slow.
- Binary search trees work well in the average case, but can grow too tall and imbalanced in the worst case.
- **Question of the day**: How to balance search trees?
# Order of growth for symbol table operations

<table>
<thead>
<tr>
<th>Goal</th>
<th>Worst case</th>
<th>Average case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert</td>
</tr>
<tr>
<td>Sequential search (unordered)</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Binary search (ordered array)</td>
<td>$\log n$</td>
<td>$n$</td>
</tr>
<tr>
<td>BST</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Goal</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
</tbody>
</table>
2-3 SEARCH TREES

2-3 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, 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.

- **Symmetric order**: Inorder traversal yields keys in ascending order.

- **Perfect balance**: Every path from root to null link (empty tree) 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).
Lecture 27: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance
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.
3.3 2–3 Tree Demo

- search
- insertion
- construction
Lecture 27: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance
How to insert into a 2-node

- Add new key to 2-node to create a 3-node.
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
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.
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.
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.
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.
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
Lecture 27: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance
2–3 tree demo: construction

insert R
Practice Time

- 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.
CONSTRUCTION

Answer

EASYQUTION

https://www.cs.usfca.edu/~galles/visualization/BTree.html
Lecture 27: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance
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 18 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!
## Summary for symbol table operations

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Worst case</th>
<th></th>
<th></th>
<th>Average case</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
</tr>
<tr>
<td>Sequential search</td>
<td>( n )</td>
<td>( n )</td>
<td>( n )</td>
<td>( n/2 )</td>
<td>( n )</td>
<td>( n/2 )</td>
</tr>
<tr>
<td>(unordered)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary search</td>
<td>( \log n )</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( n/2 )</td>
<td>( n/2 )</td>
</tr>
<tr>
<td>(ordered array)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td>( n )</td>
<td>( n )</td>
<td>( n )</td>
<td>( 1.39 \log n )</td>
<td>( 1.39 \log n )</td>
<td>?</td>
</tr>
<tr>
<td>2-3 search tree</td>
<td>( c \log n )</td>
<td>( c \log n )</td>
<td>( c \log n )</td>
<td>( c \log n )</td>
<td>( c \log n )</td>
<td>( c \log n )</td>
</tr>
</tbody>
</table>
Lecture 27: 2-3 Search Trees

- 2-3 Search Trees
- Search
- Insertion
- Construction
- Performance
ASSIGNED READINGS AND PRACTICE PROBLEMS

Readings:

▸ Textbook: Chapter 3.3 (Pages 424-431)

▸ Website:
  ▸ https://algs4.cs.princeton.edu/33balanced/

Practice Problems:

▸ 3.3.2-3.3.5