Multi-Way Search Trees

Κ08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού
Κώστας Χατζηκοκολάκης
Motivation

• We keep the **ordering** idea of BSTs
  - **Fast search**, by excluding whole subtrees

• And add **more than two children** for each node
  - Gives more flexibility in restructuring the tree
  - And news ways to **keep it balanced**
Multi-way search trees

- $d$-node: a node with $d$ children

- Each **internal** $d$-node stores $d - 1$ **ordered** values $k_1 < \ldots < k_{d-1}$
  - **No duplicate** values in the whole tree

- All values in a **subtree** lie **in-between** the corresponding node values
  - For all values $l$ in the $i$-th subtree: $k_{i-1} < l < k_i$
  - Convention: $k_0 = -\infty$, $k_d = +\infty$

- $m$-way search tree: all nodes have **at most** $m$ children
  - A BST is a 2-way search tree
Example multi-way search tree

$m = 3$
Searching in a multi-way search tree

• Simple adaptation of the algorithm for BSTs
• Start from the root, traverse towards the leaves
• In each node, there is a single subtree that can possibly contain a value \( l \)
  - The subtree \( i \) such that \( k_{i-1} < l < k_i \)
  - Continue in that subtree
Example multi-way search tree
Search for value 12

Unsuccessful search
Search for value 24

Successful search
Insertion in a multi-way search tree

- Again, simple adaptation of BSTs
  - But: we don't always need to create a new node
  - We can insert in an existing one if there is space
- Start with a search for the value \( l \) we want to insert
- If found, stop (no duplicates)
- If not found, insert at the leaf we reached
  - If full, create an \( i \)-th child, such that \( k_{i-1} < l < k_i \)
Insert value 28

$\mathbf{m} = 3$
Value 28 inserted
Insert value 32

Unsuccessful search
Value 32 inserted
Insert value 12

Unsuccessful Search
Value 12 inserted
Deletion from a multi-way search tree

Left as an exercise.
Complexity of operations

• We need to traverse the tree from the root to a leaf

• The time spent at each node is constant
  - Eg. find $i$ such that $k_{i-1} < l < k_i$
  - Assuming $m$ is fixed!

• So as usual all complexities are $O(h)$
  - $O(n)$ in the worst-case
Balanced multi-way search trees

• Similarly to BSTs we need to keep the tree balanced
  - So that $h = O(\log n)$

• AVL where a kind of balanced BSTs

• We will study two kinds of balanced multi-way search trees:
  - 2-3 trees
  - 2-3-4 trees (also known as (2,4) trees)
2-3 trees

• A **2-3 tree** is a 3-way search tree which has the following properties

  • **Size property**
    - Each node contains **1 or 2 values**
      (so each **internal** node contains **2 or 3 children**)

  • **Depth property**
    - All **leaves** have the **same depth** (lie on the same level)
Example of 2-3 tree
Height of 2-3 trees

• All nodes at all levels except the last one are internal
  - And each internal node has at least 2 children
  - So at level $i$ we have at least $2^i$ nodes

• Hence $n \geq 2^h$, in other words $h = O(\log n)$

• So we can search for an element in time $O(\log n)$
  - Using the standard algorithm for $m$-way trees
Search for L
Insertion in 2-3-trees

• We can start by following the generic algorithm for $m$-way trees
• Search for the value $l$ we want to insert
• If found, stop (no duplicates)
• If not found, insert at the leaf we reached
Example: insert B
Example: insert B
Example: result
Insertion in 2-3-trees

• But what if there is **no space at the leaf** (overflow)?

• The standard algorithm will insert a child at the leaf
   - But this **violates the depth property**!
   - The new leaf is not at the same level

• Different strategy
  - **split** the overflowed node into two nodes
  - pass the **middle value** to the parent (**separator** of the two nodes)

• The middle value might **overflow the parent**
  - Same procedure: split and send the middle value up
Example: insert M
Example: insert M

M overflows this node.
Example: insert M

The node is split in two and L is passed to the parent node.
Example: insert M

Loverflows this node

A B

D E F

H

I

J K

N M

O P

L
Example: insert M

The node is split in two and L is passed up to the parent
Example: result

Lis inserted in the root node
Example: insert Q

Q overflows this node
Example: insert Q

This node is split up and P is passed up
Example: result
Example: insert R

R is inserted in the node with Q where there is space.
Insertion in 2-3-trees

- The root might also **overflow**
- Same procedure
  - Split it
  - The middle value moves up, creating a **new root**
- This is the **only** operation that **increases** the tree's **height**
  - It increases the depth of **all nodes** simultaneously
  - 2-3-trees grow at the root, not at the leaves!
Example: insert S

S overflows this node

S overflows this node
Example: insert $S$

This node is split and $R$ is sent up
Example: insert S
Example: insert S

This node is split up and P is sent up
Example: insert S

P overflows the root

A B  D  E F  I K  J  M  N O Q R  S
Example: result

The root splits and L becomes the new root
Complexity of insertion

- We traverse the tree
  - From the root to a leaf when searching
  - From the leaf back to the root while splitting
- Each split takes constant time
  - We do at most $h + 1$ of them
- So in total $O(h) = O(\log n)$ steps
  - Recall, the tree is balanced
(2,4) trees

- A (2,4) tree (or 2-3-4 tree) is a 4-way search tree with 2 extra properties

- **Size property**
  - Each node contains between 1 and 3 values
    (so each internal node contains between 2 and 4 children)

- **Depth property**
  - All leaves have the same depth (lie on the same level)

- Such trees are balanced
  - $h = O(\log n)$
  - Proof: exercise
Insertion in (2,4) trees

• Same as for 2-3-trees
  - Search for the value
  - Insert at a leaf

• In case of an overflow (5-node)
  - Split it into a 3-node and a 2-node
  - Move the separator value $k_3$ to the parent
Overflow at a 5-node
The separating value is sent to the parent node
Node replaced with a 3-node and a 2-node
Example: insert 4
Example: insert 6
Example: insert 12

4 6 12
Example: insert 15 - overflow

4 6 12 15
Creation of new root node
Split
Example: insert 3
Example: insert 5 - overflow
5 is sent to the parent node
Split
Example: insert 10
Example: insert 8
Example

Inserted 11, 13 and 14.
Example: insert 17 - overflow
Split and send 15 to the parent node
The root overflows
Creation of new root
Split
Final tree
Complexity

• Same as for 2-3-trees
  - At most $h$ splits
  - Each split is constant time

• $O(\log n)$
  - Because the tree is balanced
Removal in (2,4) trees

• To remove a value $k_i$ from an internal node
  - Replace with its predecessor (or its successor)
  - Right-most value in the $i$-th subtree
  - Similar to the BST case of nodes with two children

• To remove a value from a leaf
  - We simply remove it
  - But it might violate the size property (underflow)
Fixing underflows

Two strategies for fixing an underlow at \( \nu \)

- Is there an **immediate sibling** \( w \) with a “spare” value? (2 or 3 values)
- If so, we do a **transfer** operation
  - Move a value of \( w \) to its parent \( u \)
  - Move a value of the parent \( u \) to \( \nu \)
- If not, we do a **fusion** operation
  - Merge \( \nu \) and \( w \), creating a new node \( \nu' \)
  - Move a value from the parent \( u \) to \( \nu' \)
  - This might **underflow the parent**, continue the same procedure there
Initial tree
Remove 4
After the transfer
Remove 12
Remove 12

\[ \text{Diagram:} \]

```
  12
 /\  \
6 10 11
 /    /
5 w  8
 / \  \
 v  13 14
    /  \
   17
```
Fusion of and
After the fusion
Remove 13
After the removal of 13
Remove 14 - underflow
Fusion
Underflow at
Fusion
Remove the root
Final tree
Readings

• T. A. Standish. *Data Structures, Algorithms and Software Principles in C*. Section 9.9

• M. T. Goodrich, R. Tamassia and D. Mount. *Data Structures and Algorithms in C++*. Section 10.4

• R. Sedgewick. *Αλγόριθμοι σε C*. 3η Αμερικανική Έκδοση. Εκδόσεις Κλειδάριθμος. Section 13.3