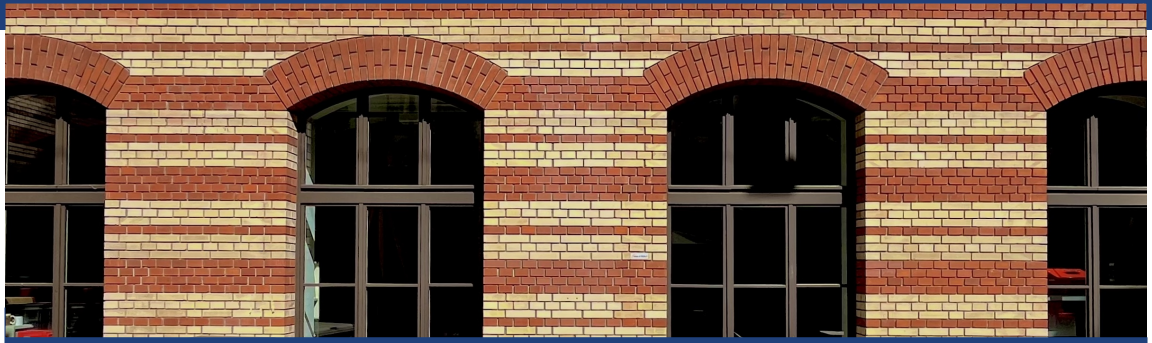


Datastructures and Algorithms

2-3 Trees, Red-Black Trees, Quadtrees

Note: *This session was held by Tristan Gabl (trgabl)*

Overview

Repetition theory
- 2-3 Trees
- Red-Black Trees

Quadtrees

Code-Example

n.ethz.ch/~agavranovic

Material

Webpage

Mail

Exercise Review: "Comparing Sorting Algorithms"

Bubblesort	min	max
Comparisons	$\Theta(n^2)$	$\Theta(n^2)$
Sequence	any	any
Swaps	0	$\Theta(n^2)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$

Exercise Review: "Comparing Sorting Algorithms"

InsertionSort	min	max
Comparisons	$\Theta(n)$	$\Theta(n^2)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$
Swaps	0	$\Theta(n^2)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$

Exercise Review: "Comparing Sorting Algorithms"

SelectionSort	min	max
Comparisons	$\Theta(n^2)$	$\Theta(n^2)$
Sequence	any	any
Swaps	0	$\Theta(n)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$

Exercise Review: "Comparing Sorting Algorithms"

QuickSort	min	max
Comparisons	$\Theta(n \log n)$	$\Theta(n^2)$
Sequence	complex	$1, 2, \ldots, n$
Swaps	$\Theta(n)$	$\Theta(n \log n)$
Sequence	$1, 2, \ldots, n$	complex

complex: Sequence must be made such that the pivot halves the sorting range in each step. For example ($n = 7$): $4, 5, 7, 6, 2, 1, 3$

Stable / In Situ

In-Situ

- QuickSort uses between $\Omega(\log n)$ and $\mathcal{O}(n)$ extra space to keep track of the recursive calls.
- MergeSort has to merge repeatedly parts of the array. There are complicated modifications to make MergeSort in-situ, but none that can be achieved by simple modifications of the standard algorithm.

Stable

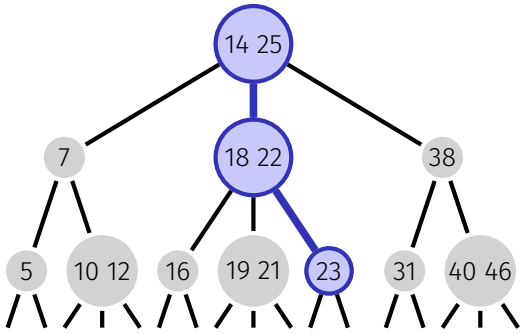
- Stability of a sorting algorithm only refers to the order of elements with the same value. Attribute each element with its original position and sort by value plus position for elements with equal values. Maximally one additional comparison per element (factor of 2), hence the asymptotic running time stays the same.

1. Repetition theory

1. Repetition theory

1.1. 2-3 Trees

Searching



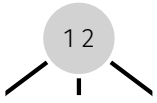
$\texttt{search}(23) \rightarrow \texttt{found}$

2-3 Tree: Insertion

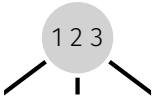
Insert the keys $1, \ldots, 7$ into an (initially empty) 2-3-tree. Draw the tree after every step (`split`/`propagate`, `join`, ...).

2-3 Tree: Insertion

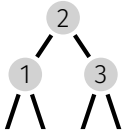
`insert(1)`:
new node



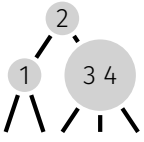
`insert(2)`:
`join`



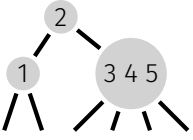
`insert(3)`:
4-node



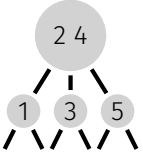
`insert(3)`:
`split/propagate`



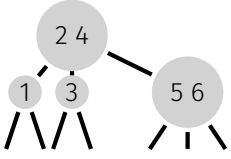
`insert(4)`:
`join`



`insert(5)`:
4-node

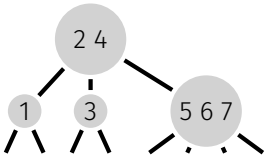


`insert(5)`:
`split/propagate`

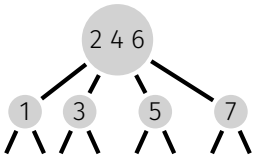


`insert(6)`:
`join`

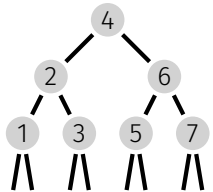
2-3 Tree: Insertion



`insert(7)`:
4-node

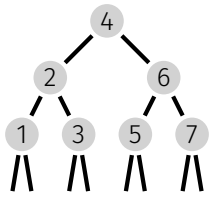


`insert(7)`:
`split/propagate`



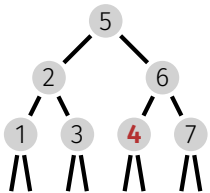
`insert(7)`:
`split/propagate`

2-3 Tree: Deletion

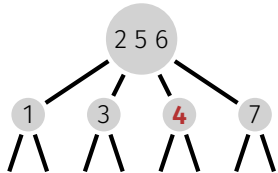


Delete key 4 from the resulting tree.

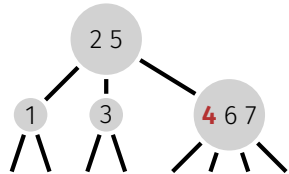
2-3 Tree: Deletion



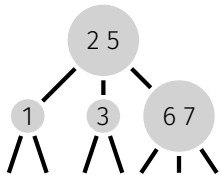
1. swap



2. create 4-node at root



3. combine with sibling



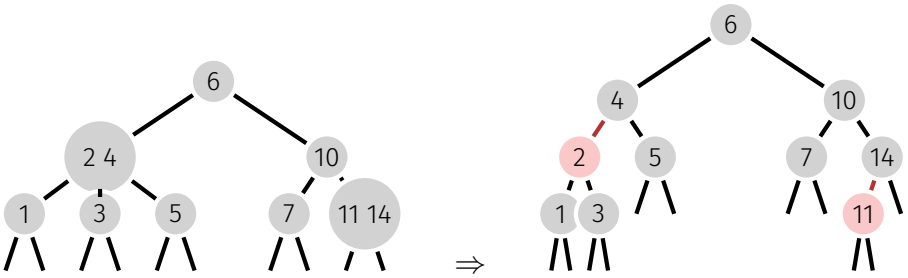
4. delete key

1. Repetition theory

1.2. Red-Black Trees

Red-Black Trees

Draw the following 2-3 tree as a red-black tree.



Red-Black Trees: True or False?

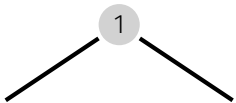
1. Right spine (right outermost path) has length $\lceil \log_2(n+1) \rceil$ (where n is the number of nodes in the corresponding 2-3 tree).
   Correct, since there are no right-leaning red edges and we have perfect black balance.
2. The number of red edges is at most the number of black edges.
   Wrong, a tree with 2 nodes and one edge must have a red edge but not black edge. (The correct upper bound for the number of red edges is the number of black edges + 1)
3. All nodes in the left subtree of a node are smaller than the node.
   Correct, since a red-black tree is a search tree.

Red-Black Trees: Insertion

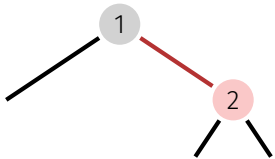
Insert the numbers $1, \ldots, 7$ into an (initially empty) red-black tree and draw the tree after every step.

Compare your steps with your result for the 2-3 tree before.

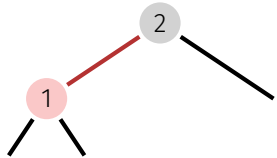
Red-Black Trees: Insertion



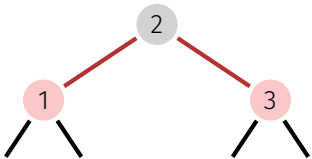
`insert(1)`



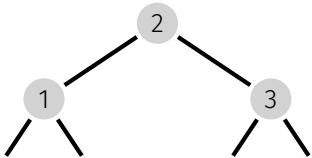
`insert(2): add`



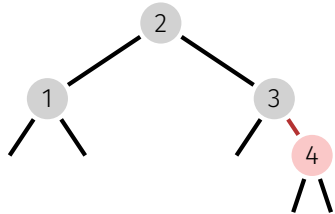
`insert(2): rotate_left`
(since right child red)



`insert(3): add`

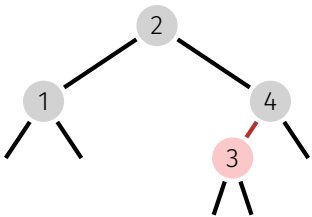


`insert(3): push_up`
(two red children)

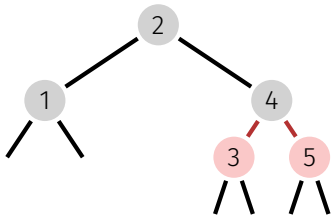


`insert(4): add`

Red-Black Trees: Insertion

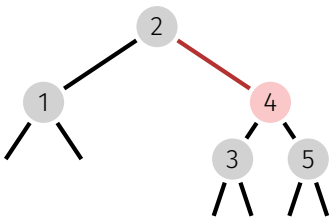


`insert(4): rotate_left`
(since right child red)

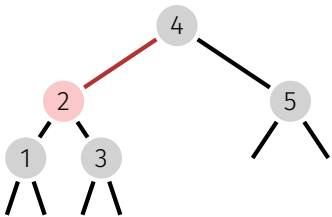


`insert(5): add`

Red-Black Trees: Insertion

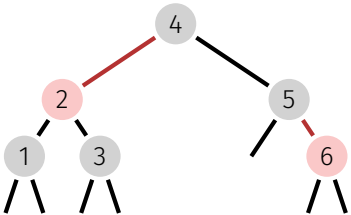


`insert(5): push_up`
(two children red)

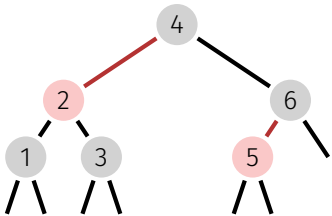


`insert(5): rotate_left`
(since right child red)

Red-Black Trees: Insertion

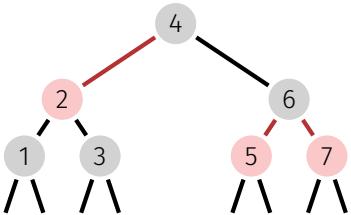


`insert(6): add`

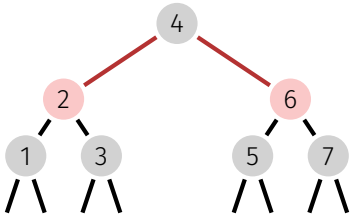


`insert(6): rotate_left`
(since right child red)

Red-Black Trees: Insertion

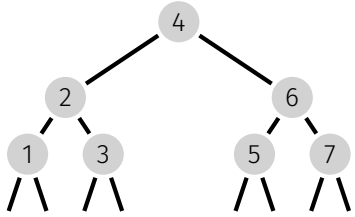


`insert(7): add`



`insert(7): push_up`
(since two children red)

Red-Black Trees: Insertion

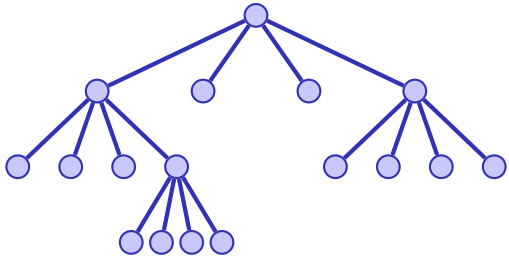


`insert(7): push_up`
(since both children red)

2. Quadtrees

Quadtrees

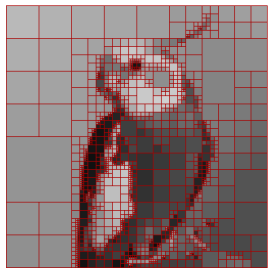
Quadtrees are trees where each node has at most **four** children.

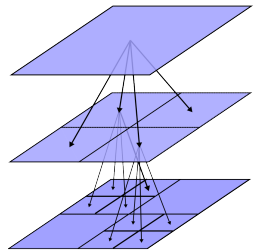


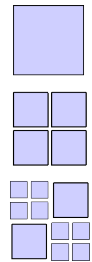
Main application: Image processing.

Quadtrees for Image Compression

Insight: (1) Divide image recursively into four regions, (2) map the regions to nodes in a quadtree and (3) assign each leaf the average color of its region.







Quadtrees for Image Compression

When and where to stop the recursion?

Too early? Is this better? **Question**: Should we stop when each node is mapped to a pixel? **Answer**: We would get the original image but gain no storage efficiency.

Quadtrees for Image Compression

We want

- as close approximation as possible, and
- as few nodes as possible.

This can be expressed as an optimization problem:

$$H_\gamma(T, \boldsymbol{y}) := \gamma \cdot \underbrace{|L(T)|}_{\text{Number of leaves}} + \underbrace{\sum_{r \in L(T)} \|\boldsymbol{y}_r - \boldsymbol{\mu}_r\|_2^2}_{\text{Cumulative approximation error of all leaves}}$$

where T is a quadtree, $\boldsymbol{y}$ is the image data, and $\gamma \geq 0$ is a regularization parameter. For a given γ we seek the optimal solution $\arg\min_T H_\gamma(T, \boldsymbol{y})$.

Quadtrees for Image Compression

$$H_\gamma(T, \boldsymbol{y}) := \gamma \cdot \underbrace{|L(T)|}_{\text{Number of leaves}} + \underbrace{\sum_{r \in L(T)} \|\boldsymbol{y}_r - \boldsymbol{\mu}_r\|_2^2}_{\text{Cumulative approximation error of all leaves}}$$

Question: What is the effect of a low value of γ?

Answer: Improves the approximation at the expense of increasing the size of the quadtree.

Algorithm: Minimize($\boldsymbol{y}$,r,γ)

Input: Image data $\boldsymbol{y} \in \mathbb{R}^S$, rectangle $r \subset S$, regularization $\gamma > 0$.
Output: $\min_T \gamma|L(T)| + \|\boldsymbol{y} - \boldsymbol{\mu}_{L(T)}\|_2^2$

if $|r| = 0$ **then return** 0
$m \leftarrow \gamma + \sum_{s \in r}(y_s - \mu_r)^2$
if $|r| > 1$ **then**
    Split r into r_{ll},r_{lr},r_{ul},r_{ur}
    $m_1 \leftarrow \text{Minimize}(y, r_{ll}, \gamma)$; $m_2 \leftarrow \text{Minimize}(y, r_{lr}, \gamma)$
    $m_3 \leftarrow \text{Minimize}(y, r_{ul}, \gamma)$; $m_4 \leftarrow \text{Minimize}(y, r_{ur}, \gamma)$
    $m' \leftarrow m_1 + m_2 + m_3 + m_4$
else
    $m' \leftarrow \infty$
if $m' < m$ **then** $m \leftarrow m'$
return m

3. Code-Example

Code-Example

Exercise class 06: Binary Trees on Code-Expert

- Augmenting a Binary Search Tree