Datenstrukturen und Algorithmen

Exercise 6

FS 2020

Program of today

1. Feedback of last exercise
2. Repetition theory
   - Binary Trees
3. Repetition Theory
   - AVL Condition
   - AVL Insert
   - Quadtrees

Feedback

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q$

Feedback

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ not suitable: $(k = 0) \mapsto 0$

Feedback

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ not suitable: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p$

Feedback

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ not suitable: $(k=0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ not suitable: $(k=0) \mapsto 0, (k=1) \mapsto 1$

Feedback

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ not suitable: $(k=0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ not suitable: $(k=0) \mapsto 0, (k=1) \mapsto 1$
- $s(j,k) = ((k \cdot j) \bmod q) + 1$

Feedback

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ not suitable: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ not suitable: $(k = 0) \mapsto 0, (k = 1) \mapsto 1$
- $s(j,k) = ((k \cdot j) \bmod q) + 1 \rightarrow$ not suitable: 1 if k is multiple of q, and range $p - q$ is not covered

Feedback

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- add 27, 2, 32

```
T_1: __, __, 27, __, __          T_2: __, __, __, __, __

T_1: __, __,  2, __, __          T_2: 27, __, __, __, __

T_1: __, __, 27, __, __          T_2:  2, 32, __, __, __
```

Feedback

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- add 7: infinite loop

```
    T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7: T_1: __, __,  7, __, __    T_2: 27, 32, __, __, __
 2: T_1: __, __,  2, __, __    T_2: 27,  7, __, __, __
32: T_1: __, __, 32, __, __    T_2:  2,  7, __, __, __
27: T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7:  ...
```

Feedback

Finding a Sub-Array

```
// calculating hash_a, hash_b, c_to_k
It1 window_end = from;
for(It2 current = begin; current != end;
    ++current, ++window_end) {
  if(window_end == to) return to;
  hash_b = (C * hash_b % M + *current) % M;
  hash_a = (C * hash_a % M + *window_end) % M;
  c_to_k = c_to_k * C % M;
}
```

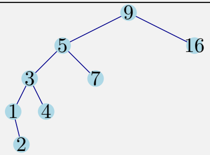
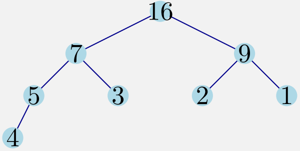
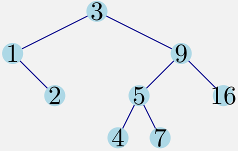
Feedback

Finding a Sub-Array

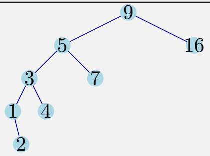
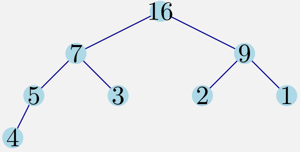
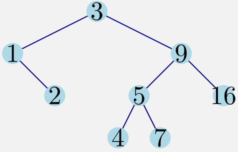
```
// looking for b and updating hash_a
for(It1 window_begin = from; ;
    ++window_begin, ++window_end) {
  if(hash_a == hash_b)
    if(std::equal(window_begin, window_end, begin, end))
      return window_begin;
  if(window_end == to) return to;
  hash_a = (C * hash_a % M + *window_end
            + (M - c_to_k) * *window_begin % M) % M;
}
```

2. Repetition theory

Comparison of binary Trees

	Search trees	**Heaps** Min- / Max-Heap	**Balanced trees** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Insertion	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Search	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Deletion	$\Theta(h(T))$	Search + $\Theta(\log n)$	$\Theta(\log n)$

Comparison of binary Trees

	Search trees	**Heaps** Min- / Max-Heap	**Balanced trees** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Insertion	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Search	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Deletion	$\Theta(h(T))$	Search $+\ \Theta(\log n)$	$\Theta(\log n)$

Recall: $\Theta(\log n) \leq \Theta(h(T)) \leq \Theta(n)$

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).

MinHeap

- Insert at the very back of the Array.
- Restore Heap-Condition: `siftUp` (climb successively).

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).

MinHeap

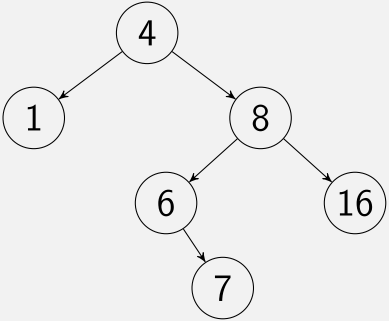
- Insert at the very back of the Array.
- Restore Heap-Condition: `siftUp` (climb successively).

Exercise: Insert $4, 8, 16, 1, 6, 7$ into empty Tree/Heap.

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



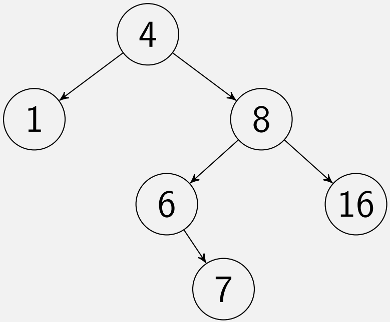
MinHeap

- Insert at the very back of the Array.
- Restore Heap-Condition: `siftUp` (climb successively).

Repetition: Binary Trees, Inserting a Key

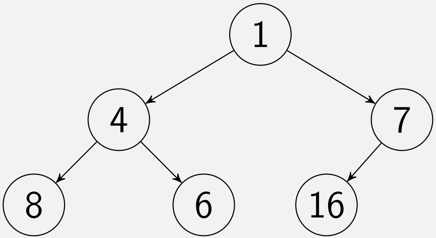
Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



MinHeap

- Insert at the very back of the Array.
- Restore Heap-Condition: `siftUp` (climb successively).



Repetition: Binary Trees, Deleting a Key

Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?

MinHeap

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.

Repetition: Binary Trees, Deleting a Key

Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?

MinHeap

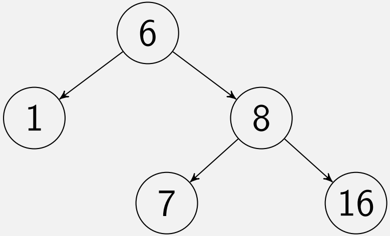
- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.

Exercise: Delete 4 from Example Tree/Heap.

Repetition: Binary Trees, Deleting a Key

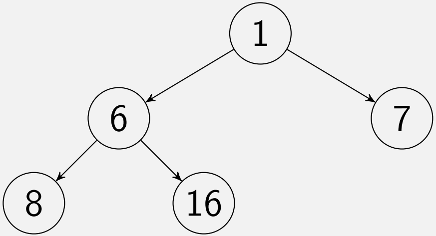
Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?



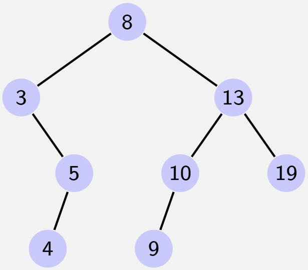
MinHeap

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.



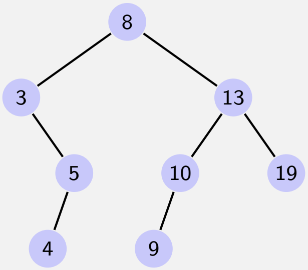
Traversal possibilities

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



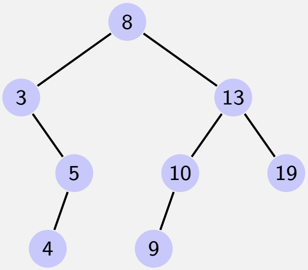
Traversal possibilities

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



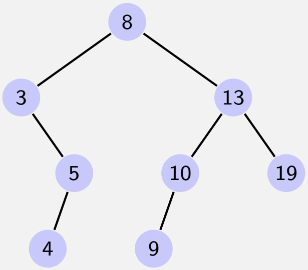
Traversal possibilities

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



Traversal possibilities

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19



Quiz

Draw a binary search tree each that represents the following traversals. Is the tree unique?

inorder	1 2 3 4 5 6 7 8
preorder	4 3 1 2 8 6 5 7
postorder	1 3 2 5 6 8 7 4

Provide for each order a sequence of numbers from $\{1, \ldots, 4\}$ such that it cannot result from a valid binary search tree

Answers

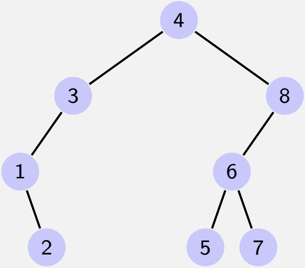
inorder: any binary search tree with numbers $\{1, \ldots, 8\}$ is valid.

The tree is not unique

There is no search tree for any non-sorted sequence. Counterexample 1 2 4 3

Answers

preorder 4 3 1 2 8 6 5 7

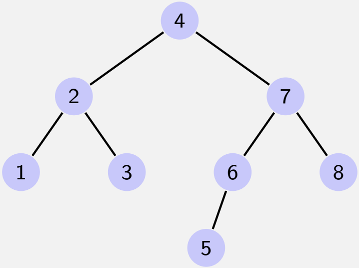


Tree is unique

It mus hold recursively that first there is a group of numbers with lower and then with higher number than the first value. Counterexample: 3 1 4 2

Answers

postorder 1 3 2 5 6 8 7 4

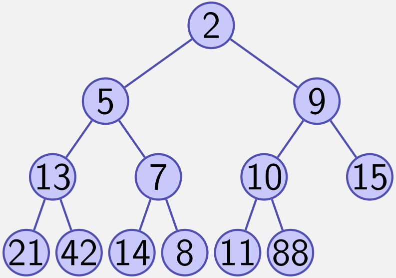


Tree is unique

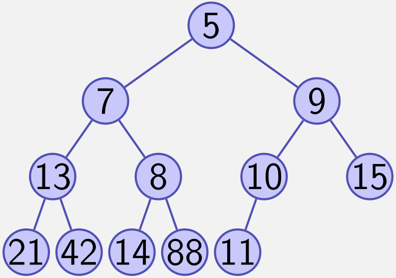
Construction here: `https://www.techiedelight.com/build-binary-search-tree-from-postorder-sequence/`, similar argument as before, but backwards. Counterexample 4 2 1 3

Heap

On the following Min-Heap, perform an extract-min operation, including re-establishing the heap-condition, as shown in class. What does the heap look like after the operation?



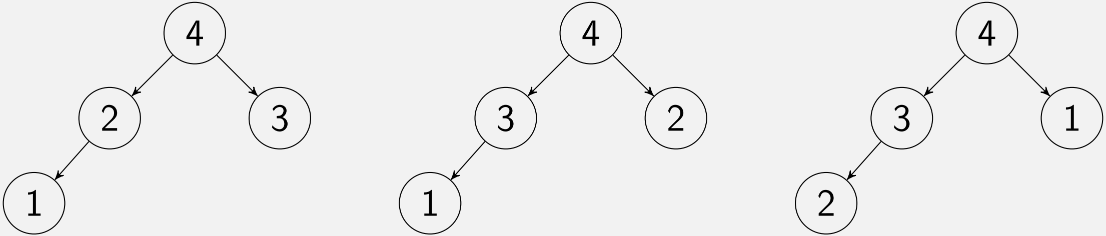
Solution



Number of MaxHeaps on n distinct keys

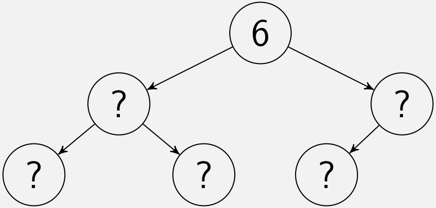
Let $N(n)$ denote the number of distinct Max-Heaps which can be built from all the keys $1, 2, \ldots, n$. For example we have $N(1) = 1,\ N(2) = 1,\ N(3) = 2,\ N(4) = 3$ und $N(5) = 8$.

Find the values $N(6)$ and $N(7)$.



Number of MaxHeaps on n distinct keys

A MaxHeap containing the elements $1, 2, 3, 4, 5, 6$ has the structure:



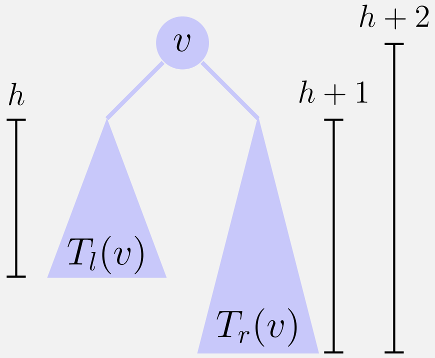
Number of combinations to choose elements for the left subtree: $\binom{5}{3}$.

$$\Rightarrow N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) 10 \cdot 2 \cdot 1 = 20.$$

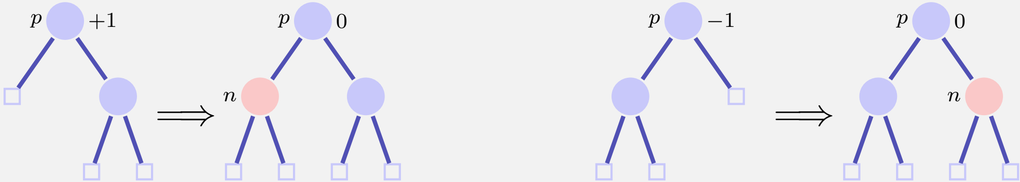
$$\text{and } N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80.$$

AVL Condition

AVL Condition: for eacn node v *of a tree* $\mathrm{bal}(v) \in \{-1, 0, 1\}$



Balance at Insertion Point

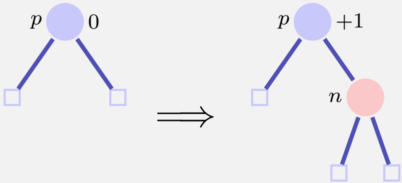


case 1: $\mathrm{bal}(p) = +1$

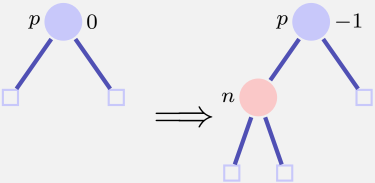
case 2: $\mathrm{bal}(p) = -1$

Finished in both cases because the subtree height did not change

Balance at Insertion Point



case 3.1: $\mathrm{bal}(p) = 0$ right



case 3.2: $\mathrm{bal}(p) = 0$, left

Not finished in both case. Call of `upin(p)`

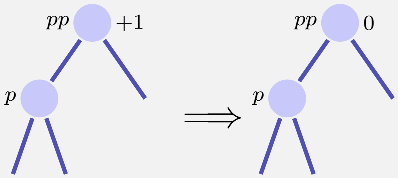
upin(p) - invariant

When **`upin(p)`** is called it holds that

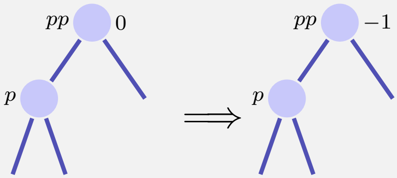
- the subtree from p is grown and
- $\mathrm{bal}(p) \in \{-1, +1\}$

upin(p)

Assumption: p is left son of pp[1]



case 1: $\mathrm{bal}(pp) = +1$, done.



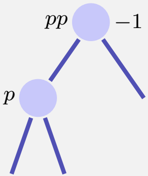
case 2: $\mathrm{bal}(pp) = 0$, `upin(pp)`

In both cases the AVL-Condition holds for the subtree from pp

[1] If p is a right son: symmetric cases with exchange of $+1$ and -1

upin(p)

Assumption: p is left son of pp



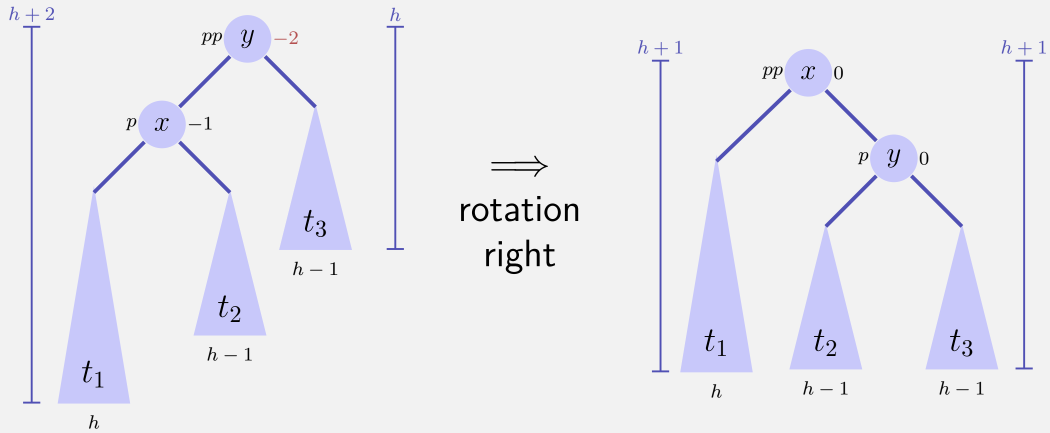
case 3: $\mathrm{bal}(pp) = -1$,

This case is problematic: adding n to the subtree from pp has violated the AVL-condition. Re-balance!

Two cases $\mathrm{bal}(p) = -1$, $\mathrm{bal}(p) = +1$

Rotations

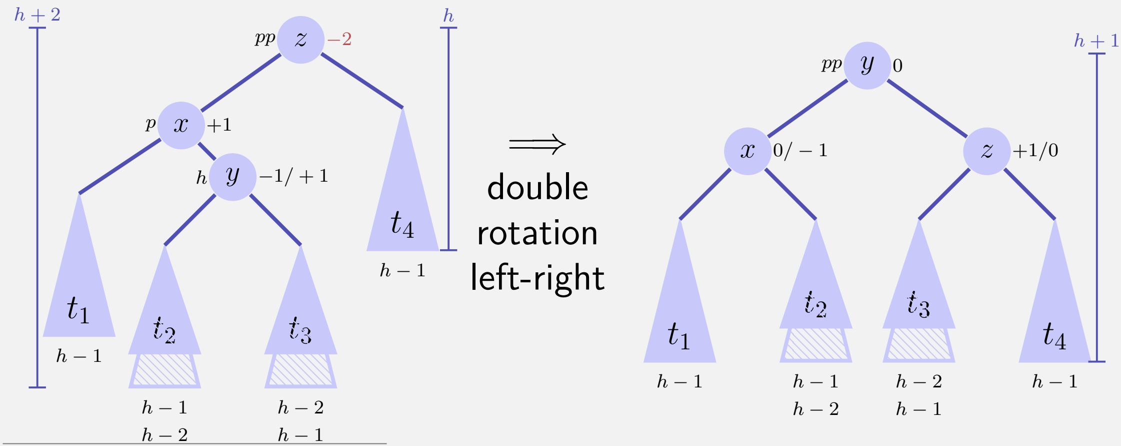
case 1.1 $\mathrm{bal}(p) = -1.$ [2]



[2] p right son: $\Rightarrow \mathrm{bal}(pp) = \mathrm{bal}(p) = +1$, left rotation

Rotations

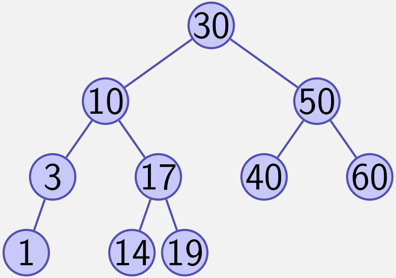
case 1.1 $\mathrm{bal}(p) = -1$. [3]



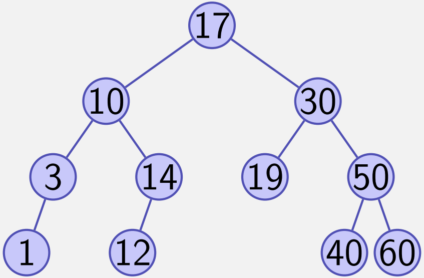
[3] p right son $\Rightarrow$ $\mathrm{bal}(pp) = +1$, $\mathrm{bal}(p) = -1$, double rotation right left

Quiz

In the following AVL tree, insert key 12 and rebalance (as shown in class). What does the AVL tree look like after the operation that has been shown in class?



Solution



Minimization of a functional for signal segmentation

$\mathcal{P}$ Partition $\gamma \geq 0$ regularization parameter

$f_{\mathcal{P}}$ approxmation z image $=$ 'data'

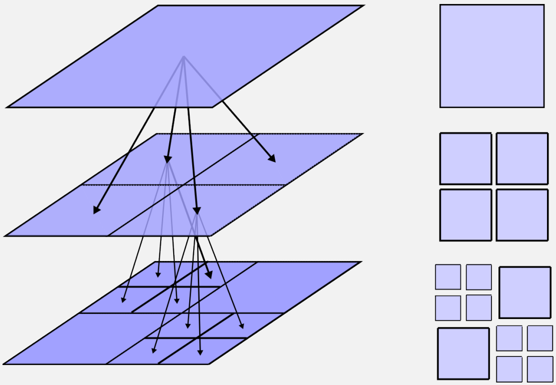
Goal: Efficient mimization of the functional

$$H_{\gamma,z} : \mathfrak{S} \to \mathbb{R}, \quad (\mathcal{P}, f_{\mathcal{P}}) \mapsto \gamma \cdot |\mathcal{P}| + \|z - f_{\mathcal{P}}\|_2^2.$$

Result $(\hat{\mathcal{P}}, \hat{f}_{\hat{\mathcal{P}}}) \in \operatorname{argmin}_{(\mathcal{P}, f_{\mathcal{P}})} H_{\gamma,z}$ can be interpreted as *optimal compromise between regularity and fidelity to data.*

Minimization of a Functional using Quadtrees

Separation of a two-dimensional range into 4 equally seized parts.

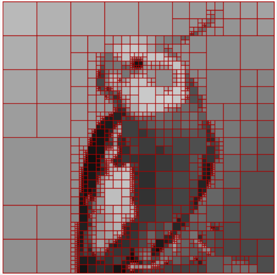


Algorithmus: Minimize(z,r,γ)

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

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

Minimization of a Functional using Quadtrees



Questions?