Übung Algorithmische Geometrie

Lineare Programmierung + Bereichsanfragen

LEHRSTUHL FÜR ALGORITHMIK I · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Andreas Gemsa

24.05.2011

Übersicht

Übungsblatt 4

Nachtrag zu Übungsblatt 3

Übungsblatt 5

Werbung

Aufgabe 1

Korrektheitsbeweis:

a) Zeige, dass jede mögliche Permutation von A gleich wahrscheinlich ist.

RandomPermutation(A)

Input: Array $A[1 \ldots n]$
Output: Array A, zufällig gleichverteilt permutiert
for $k \leftarrow n$ **to** 2 **do**
  $r \leftarrow$ Random(k)
  tausche $A[r]$ und $A[k]$

Aufgabe 1

Korrektheitsbeweis:

b) Zeige, dass die Aussage aus a) nicht stimmt, wenn wir in der 2. Zeile k mit n ersetzen.

RandomPermutation(A)

Input: Array $A[1 \ldots n]$
Output: Array A, zufällig gleichverteilt permutiert
for $k \leftarrow n$ **to** 2 **do**
  $r \leftarrow$ Random(k)
  tausche $A[r]$ und $A[k]$

Aufgabe 1

Korrektheitsbeweis:

b) Zeige, dass die Aussage aus a) nicht stimmt, wenn wir in der 2. Zeile k mit n ersetzen.

RandomPermutation(A)

Input: Array $A[1 \ldots n]$

Output: Array A, zufällig gleichverteilt permutiert ?

for $k \leftarrow n$ **to** 2 **do**

  $r \leftarrow$ Random(n)

  tausche $A[r]$ und $A[k]$

Aufgabe 2

ParanoidMax

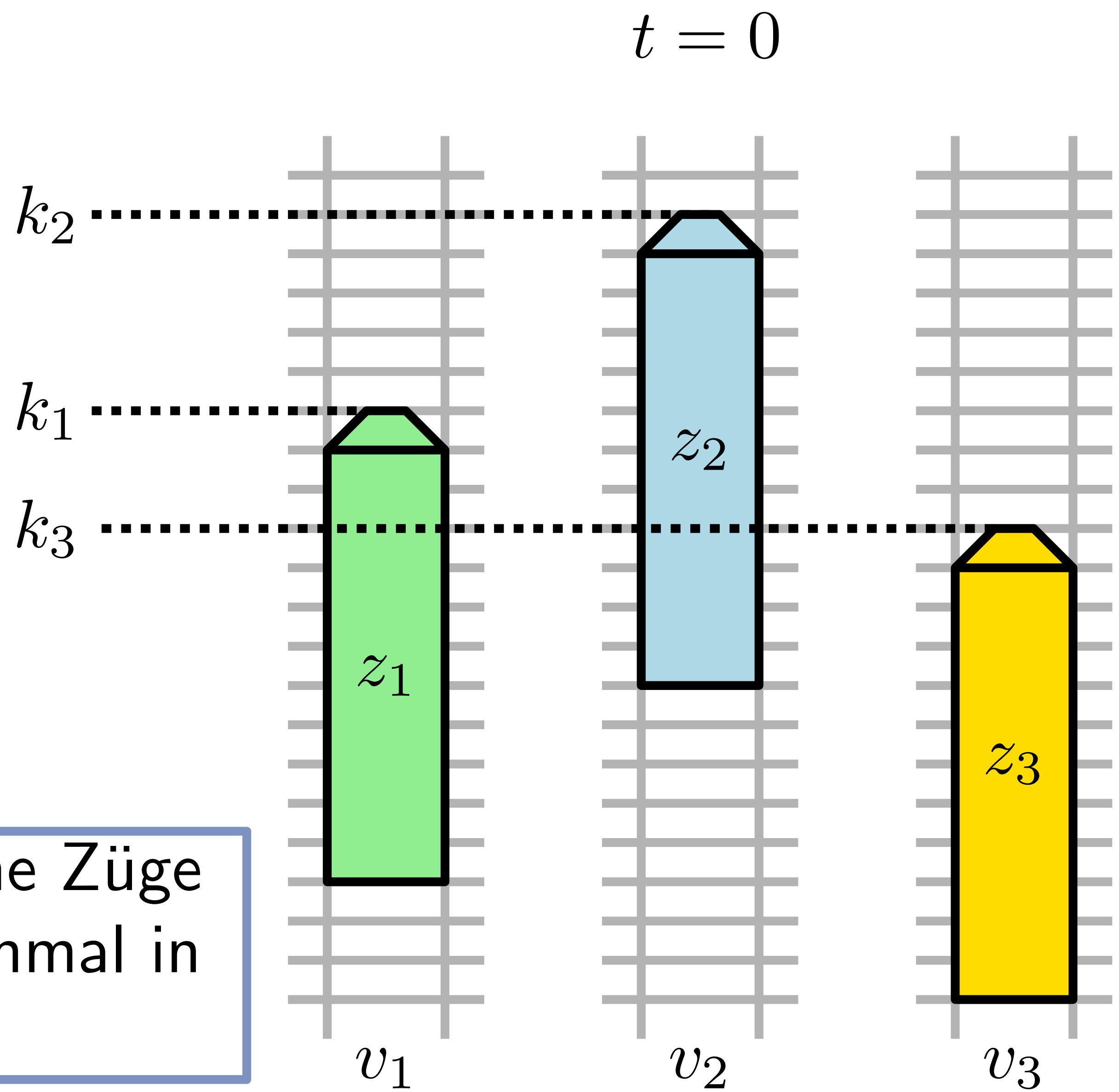
Input: Endliche Menge $A \subset \mathbb{R}$
Output: Maximum $\max_{a \in A} a$ der Menge

if $|A| = 1$ **then**
  return einziges Element $a \in A$
else
  $a =$ zufällig gewähltes Element aus A
  $b =$ ParanoidMax($A \setminus \{a\}$)
  if $b \geq a$ **then**
    return b
  else
    prüfe unnötigerweise jedes Element aus $A \setminus \{a\}$, um sicherzugehen, dass a wirklich größer ist
    return a

a) Asymptotische w-c Laufzeit?

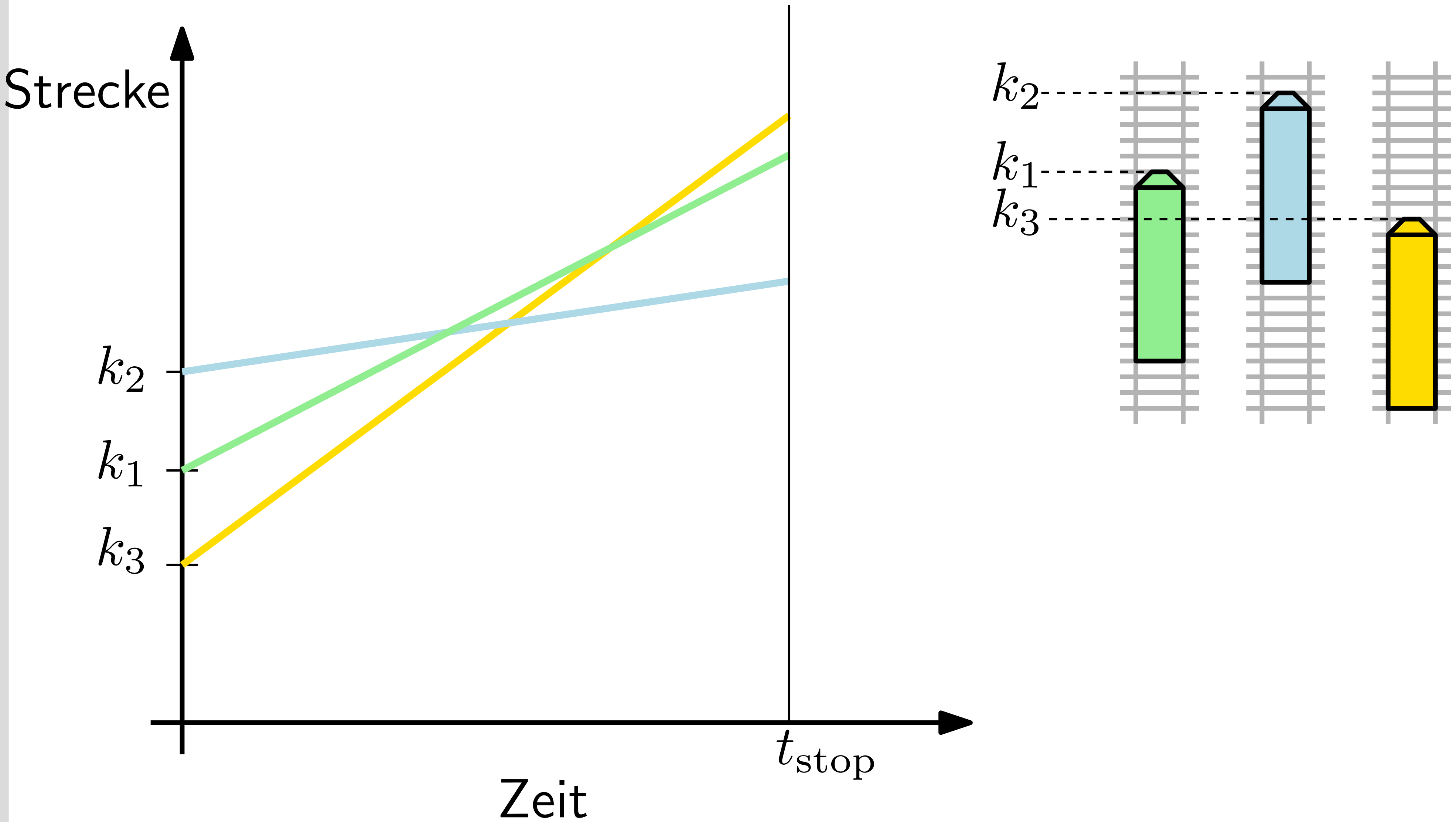
b) erwartete Laufzeit echt besser als w-c

Aufgabe 3 : Züge

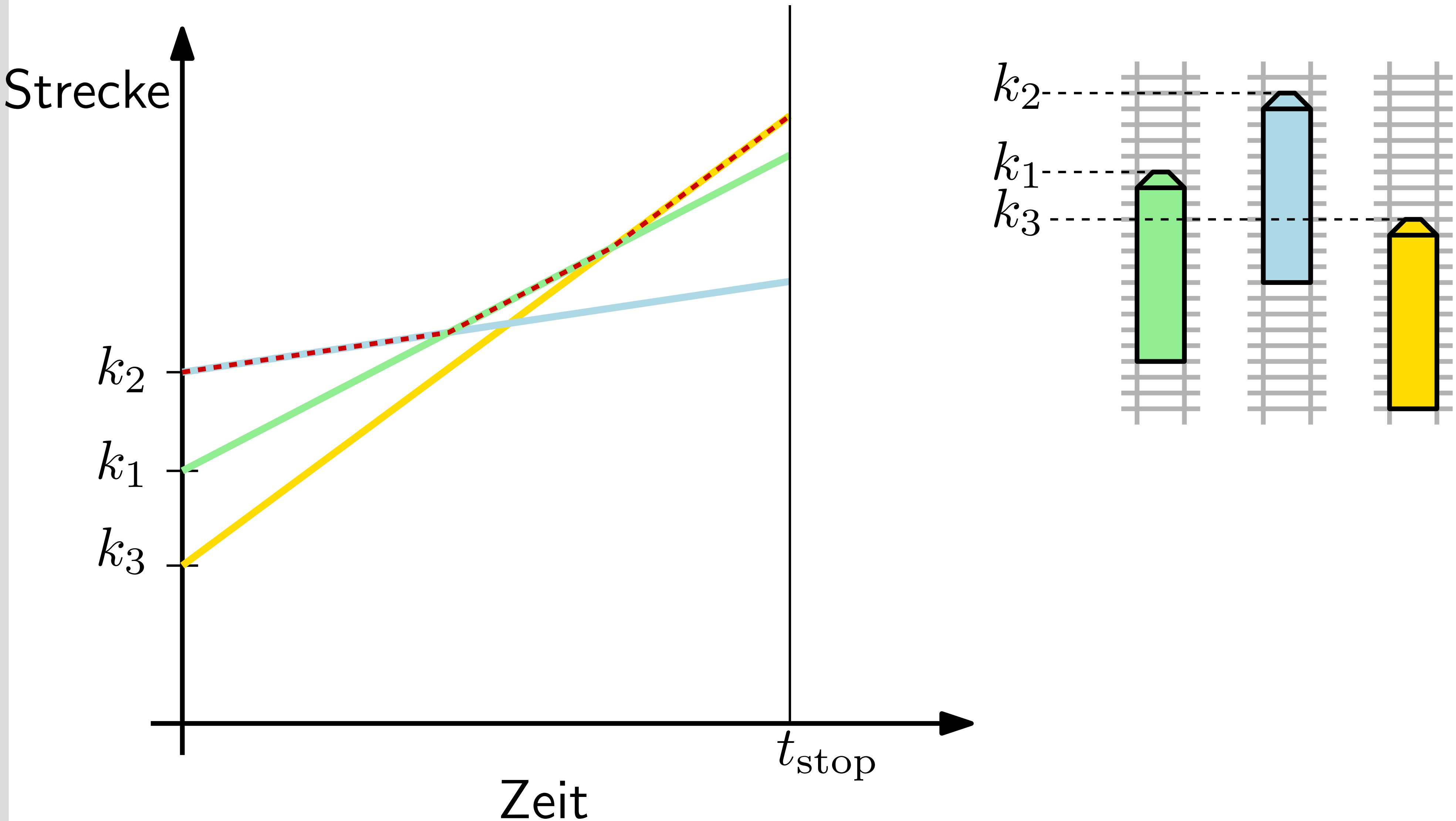


Algorithmus: welche Züge bis Zeitpunkt t_s einmal in Führung

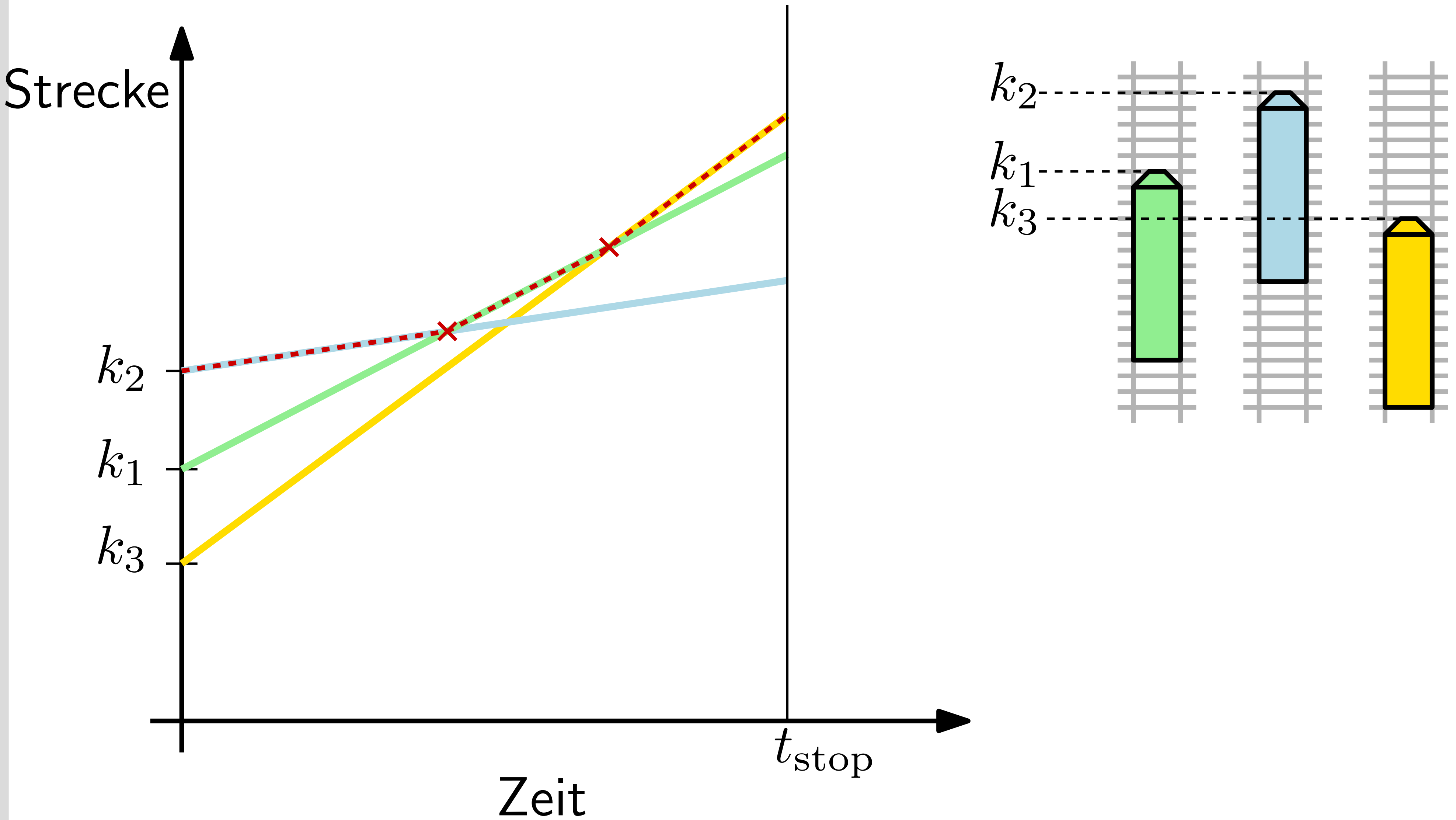
Aufgabe 3 : Züge



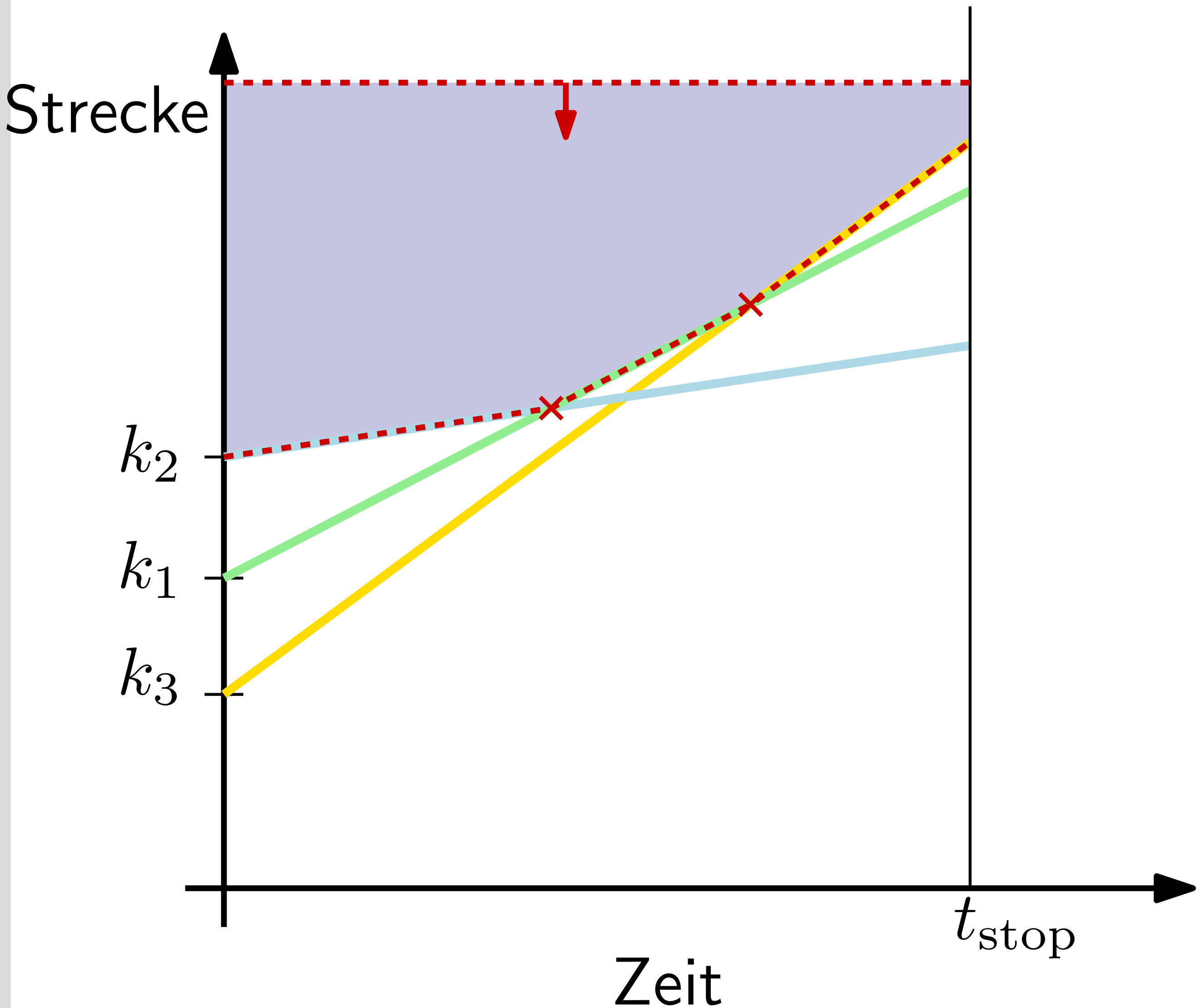
Aufgabe 3 : Züge

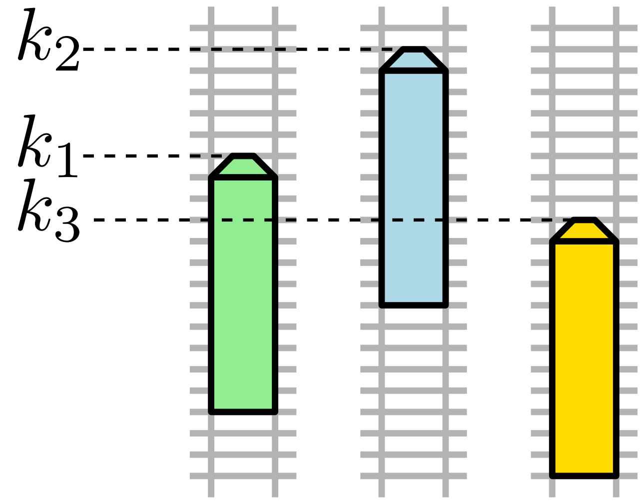


Aufgabe 3 : Züge

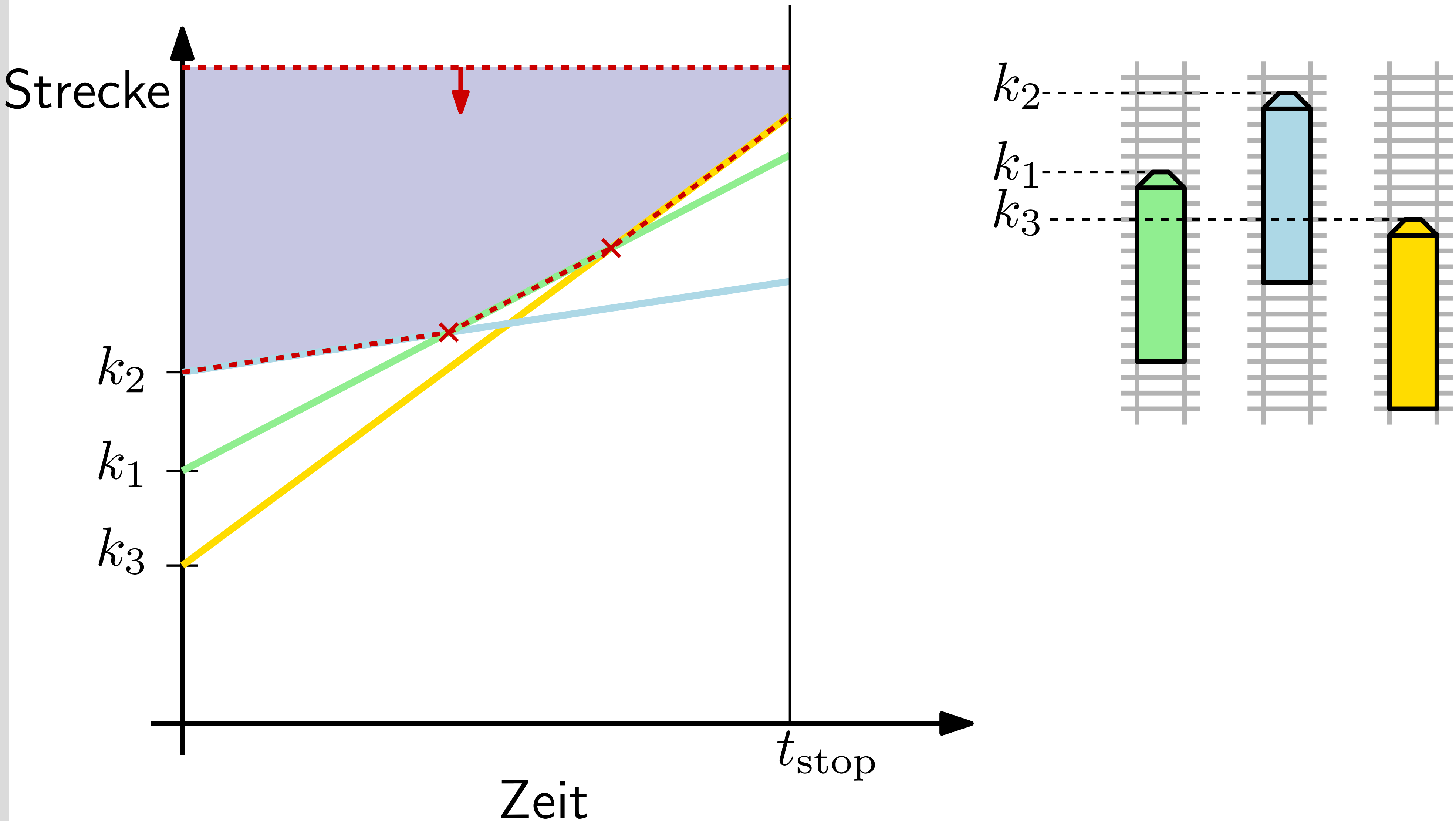


Aufgabe 3 : Züge





Aufgabe 3 : Züge



Aufgabe 3 : Züge

IntersectHalfplanes(H)

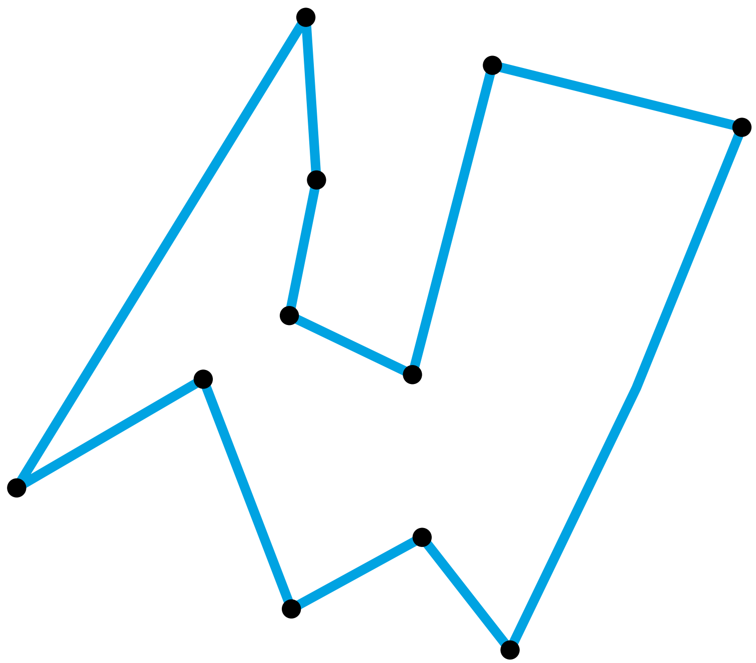
if $|H| = 1$ **then**
  $C \leftarrow H$
else
  $(H_1, H_2) \leftarrow$ SplitInHalves(H)
  $C_1 \leftarrow$ IntersectHalfplanes(H_1)
  $C_2 \leftarrow$ IntersectHalfplanes(H_2)
  $C \leftarrow$ IntersectConvexRegions(C_1, C_2)
return C

Aufgabe 4 : Polygon Partitionieren

VL: ALG zur Zerlegung eines Polygons in y-mon. Teilstücke

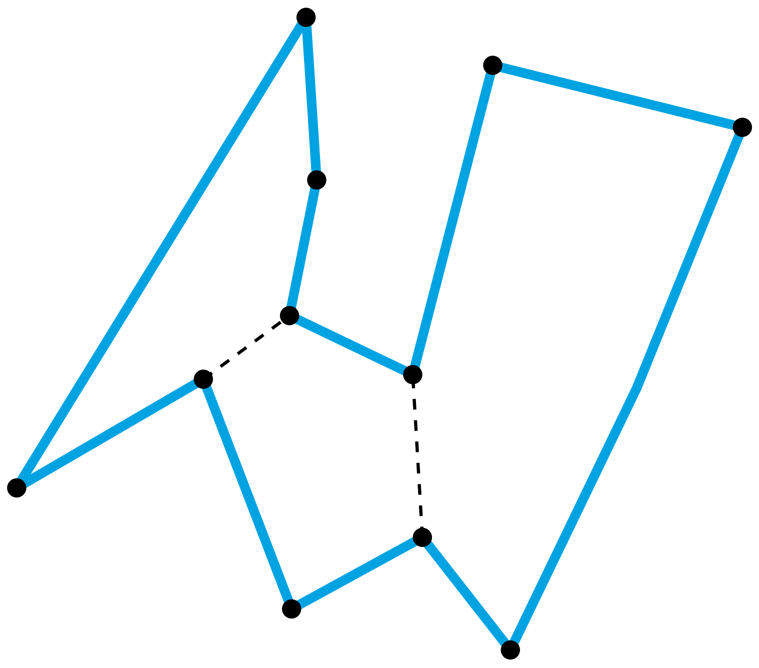
Aufgabe 4 : Polygon Partitionieren

VL: ALG zur Zerlegung eines Polygons in y-mon. Teilstücke



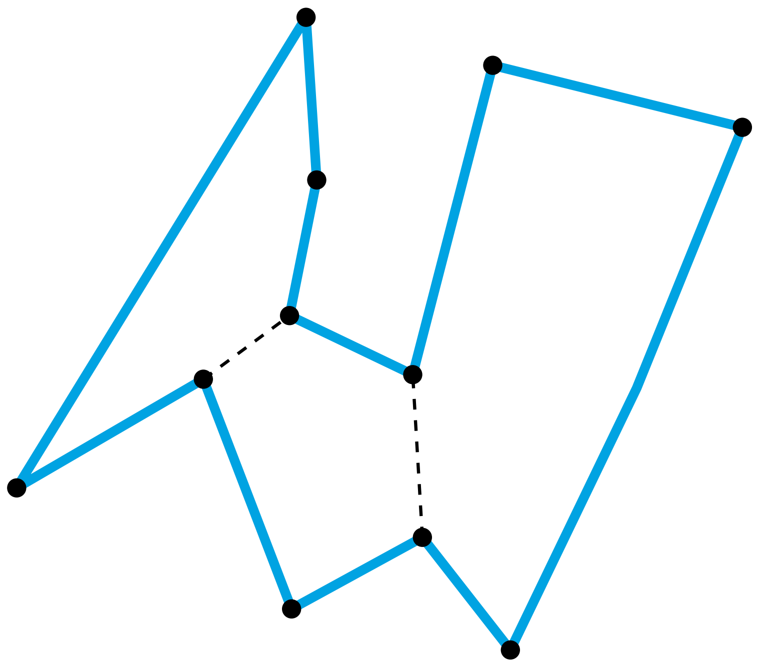
Aufgabe 4 : Polygon Partitionieren

VL: ALG zur Zerlegung eines Polygons in y-mon. Teilstücke



Aufgabe 4 : Polygon Partitionieren

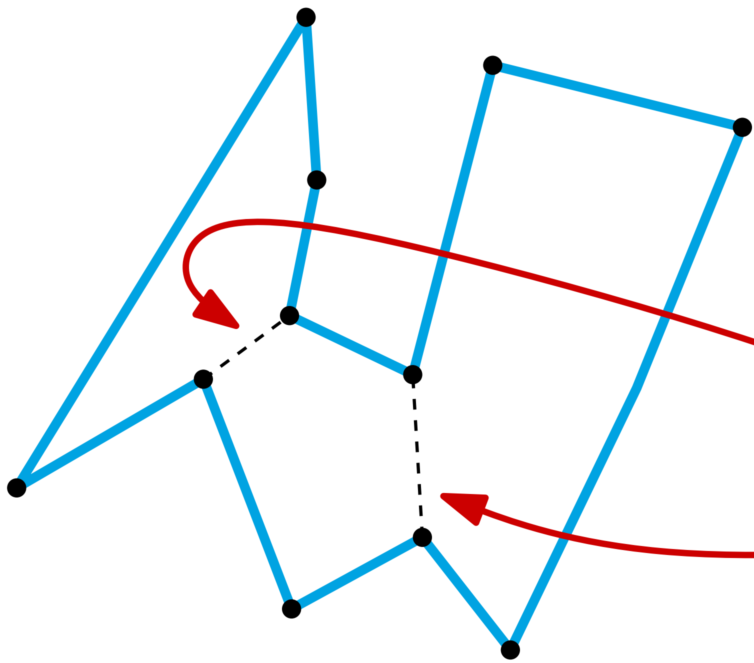
VL: ALG zur Zerlegung eines Polygons in y-mon. Teilstücke



Datenstruktur: Doppelt-verkettete Kantenliste (DCEL)

Aufgabe 4 : Polygon Partitionieren

VL: ALG zur Zerlegung eines Polygons in y-mon. Teilstücke

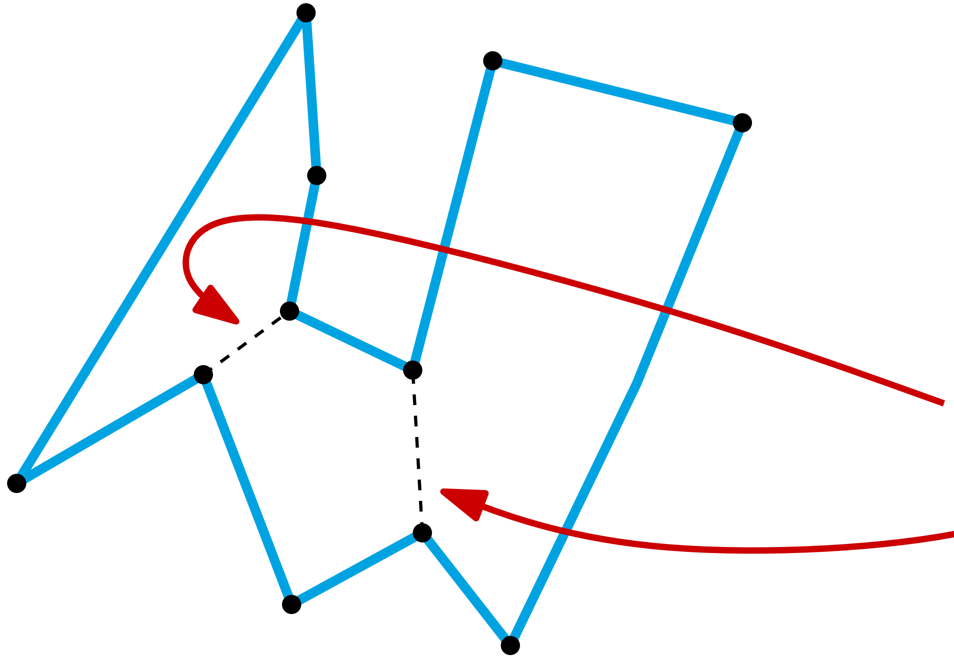


Behauptung:
Einfügen von einer Diagonalen in $O(1)$ möglich

Datenstruktur: Doppelt-verkettete Kantenliste (DCEL)

Aufgabe 4 : Polygon Partitionieren

VL: ALG zur Zerlegung eines Polygons in y-mon. Teilstücke

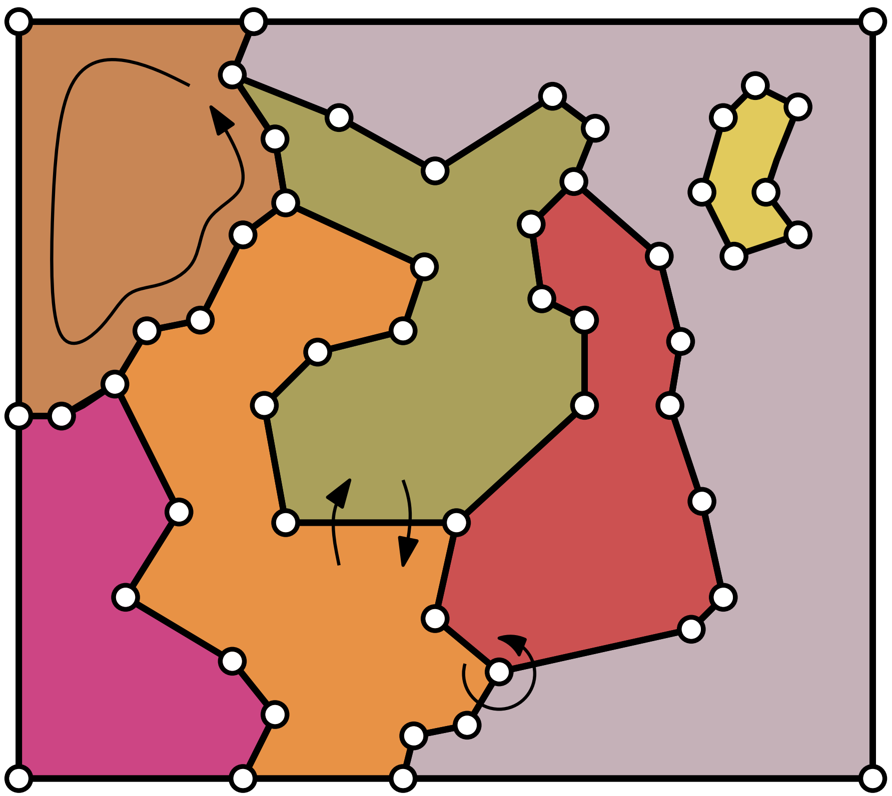


Behauptung:
Einfügen von einer Diagonalen in $O(1)$ möglich

Datenstruktur: Doppelt-verkettete Kantenliste (DCEL)

Datenstruktur für Landkarten

VL



- (politische) Landkarte entspricht Unterteilung der Ebene in Polygone
- Unterteilung entspricht Einbettung eines planaren Graphen mit
  - Knoten
  - Kanten
  - Facetten

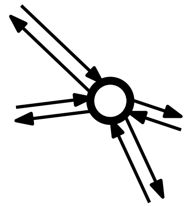
Welche Operationen sollte eine Datenstruktur für Unterteilungen der Ebene unterstützen?

- Kanten einer Facette ablaufen
- via Kante von Facette zu Facette wechseln
- Nachbarknoten in zyklischer Reihenfolge besuchen

Doppelt verkettete Kantenliste (DCEL)

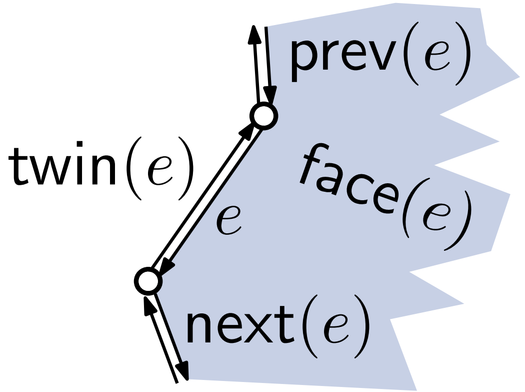
Zutaten:

- Knoten



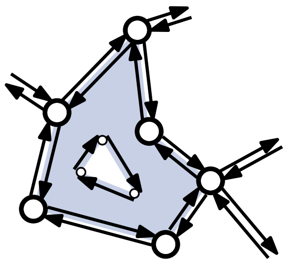
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten = zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - inzidente Facette face(e)

- Facetten

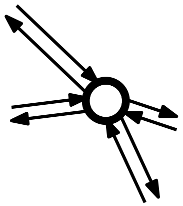


  - Randkante outer(f)
  - Kantenliste inner(f) für evtl. Löcher

Doppelt verkettete Kantenliste (DCEL)

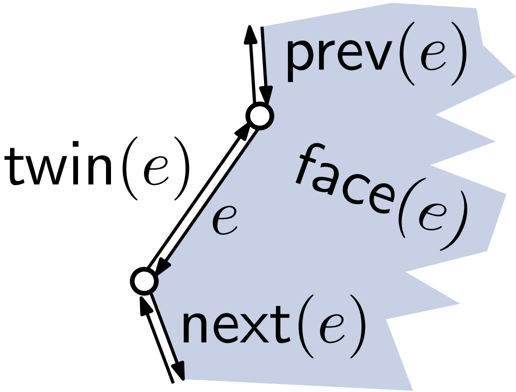
Zutaten:

- Knoten



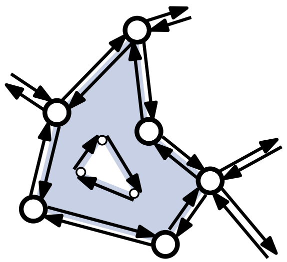
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten = zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - inzidente Facette face(e)

- Facetten

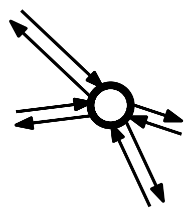


  - Randkante outer(f)
  - Kantenliste inner(f) für evtl. Löcher

Doppelt verkettete Kantenliste (DCEL)

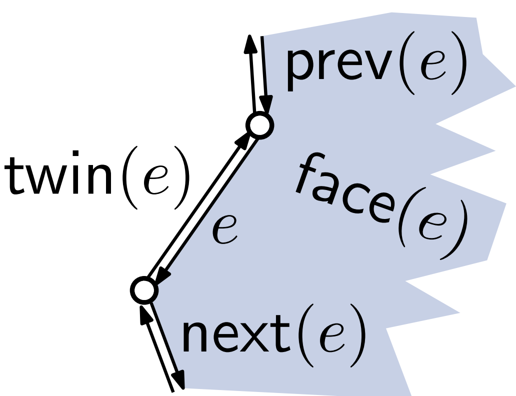
Zutaten:

- Knoten



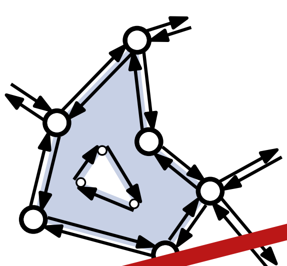
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten $=$ zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - inzidente Facette face(e)

- ~~Facetten~~

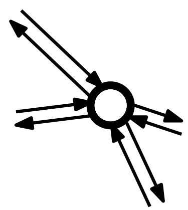


  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Doppelt verkettete Kantenliste (DCEL)

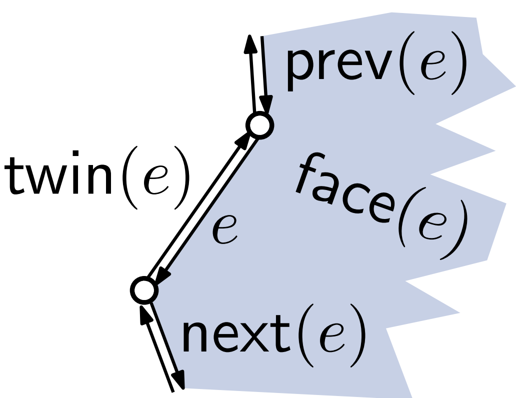
Zutaten:

- Knoten



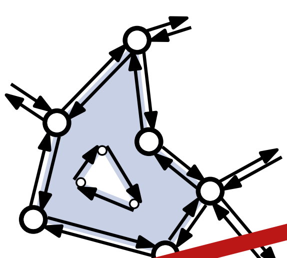
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten $=$ zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - ~~inzidente Facette face(e)~~

- ~~Facetten~~

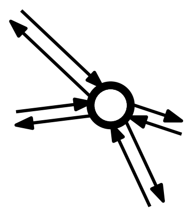


  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Doppelt verkettete Kantenliste (DCEL)

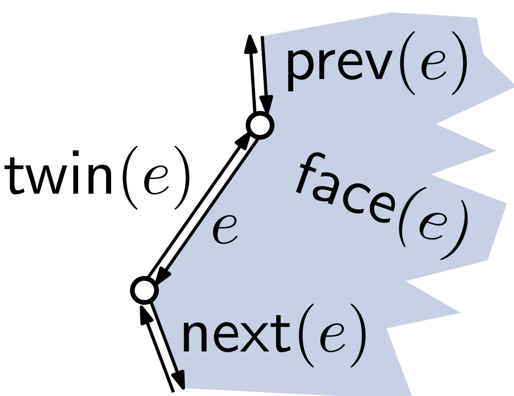
Zutaten:

- Knoten



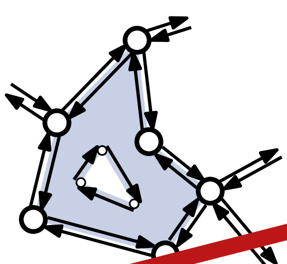
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten $=$ zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - ~~inzidente Facette face(e)~~

- ~~Facetten~~

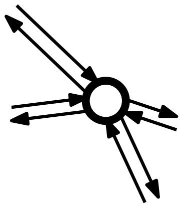


  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Doppelt verkettete Kantenliste (DCEL)

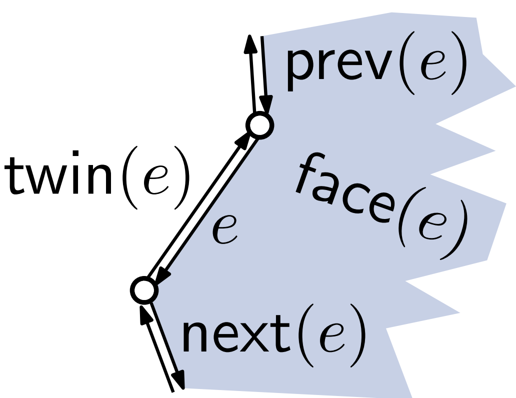
Zutaten:

- Knoten



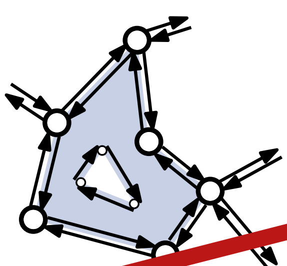
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten $=$ zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - ~~inzidente Facette face(e)~~

- ~~Facetten~~

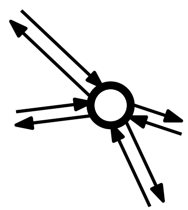


  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Doppelt verkettete Kantenliste (DCEL)

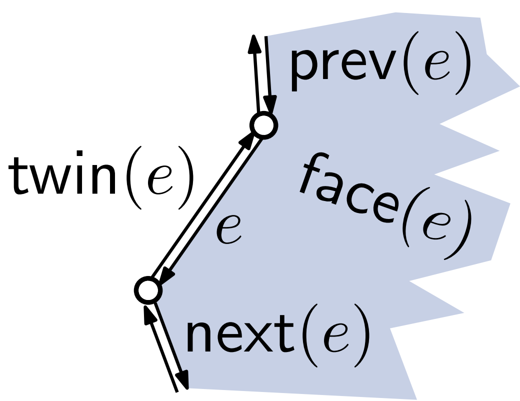
Zutaten:

- Knoten



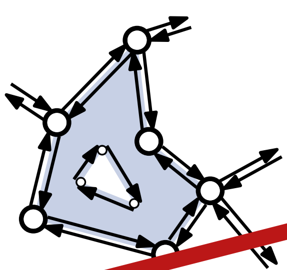
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten = zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - ~~inzidente Facette face(e)~~

- ~~Facetten~~

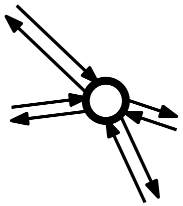


  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Doppelt verkettete Kantenliste (DCEL)

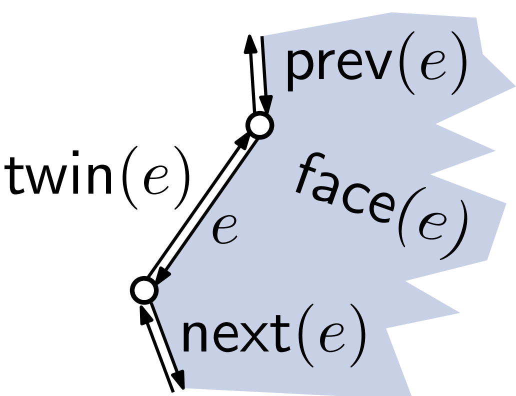
Zutaten:

- Knoten



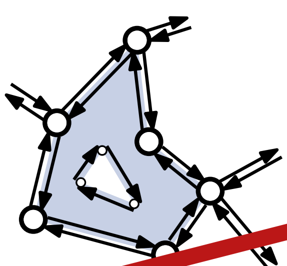
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten $=$ zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - ~~inzidente Facette face(e)~~

- ~~Facetten~~

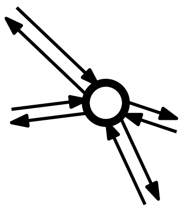


  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Doppelt verkettete Kantenliste (DCEL)

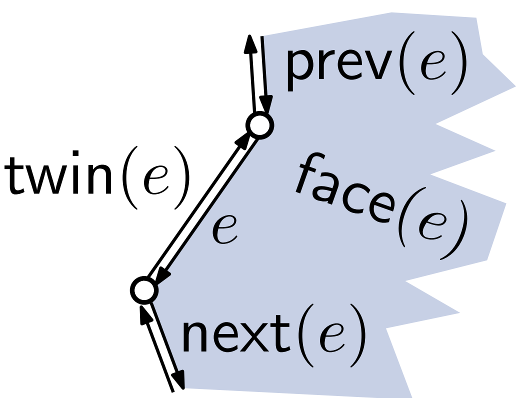
Zutaten:

- Knoten



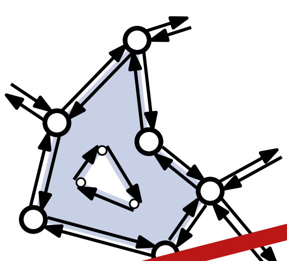
  - Koordinaten $(x(v), y(v))$
  - (erste) ausgehende Kante edge(v)

- Kanten = zwei Halbkanten



  - Knoten origin(v)
  - Gegenkante twin(e)
  - Vorgänger prev(e) & Nachfolger next(e)
  - ~~inzidente Facette face(e)~~

- ~~Facetten~~



  - ~~Randkante outer(f)~~
  - ~~Kantenliste inner(f) für evtl. Löcher~~

Lösungsschritte

1. Wie umgeht man das aktualisieren der *face*(e) Information?

Lösungsschritte

1. Wie umgeht man das aktualisieren der *face*(e) Information?
   - Facetten spielen keine (große) Rolle

Lösungsschritte

1. Wie umgeht man das aktualisieren der *face*(e) Information?

– Facetten spielen keine (große) Rolle

2. Wie wählen wir in $O(1)$ die richtigen Kanten für das anpassen der *next*(e) bzw. *prev*(e) Einträge?

Lösungsschritte

1. Wie umgeht man das aktualisieren der *face*(e) Information?

– Facetten spielen keine (große) Rolle

2. Wie wählen wir in $O(1)$ die richtigen Kanten für das anpassen der *next*(e) bzw. *prev*(e) Einträge?

a) Nur konstant viele Kanten inzident zu jedem Knoten

Lösungsschritte

1. Wie umgeht man das aktualisieren der *face*(e) Information?

– Facetten spielen keine (große) Rolle

2. Wie wählen wir in $O(1)$ die richtigen Kanten für das anpassen der *next*(e) bzw. *prev*(e) Einträge?

a) Nur konstant viele Kanten inzident zu jedem Knoten

b) Durch passende Sortierung können wir die korrekten Kanten finden

Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
  $v \leftarrow \mathcal{Q}$.nextVertex()
  $\mathcal{Q}$.deleteVertex(v)
  handleVertex(v)
return $\mathcal{D}$

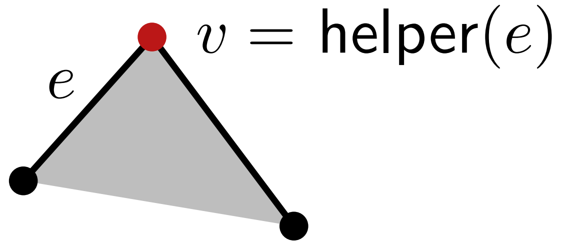
Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
    $v \leftarrow \mathcal{Q}$.nextVertex()
    $\mathcal{Q}$.deleteVertex(v)
    handleVertex(v)
return $\mathcal{D}$

handleStartVertex(vertex v)

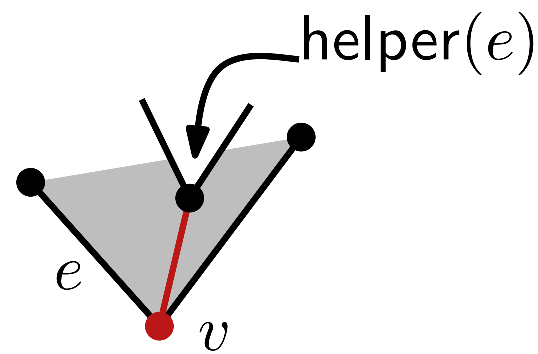
$\mathcal{T} \leftarrow$ füge linke Kante e ein
helper(e) $\leftarrow v$



Algorithmus MakeMonotone(P)

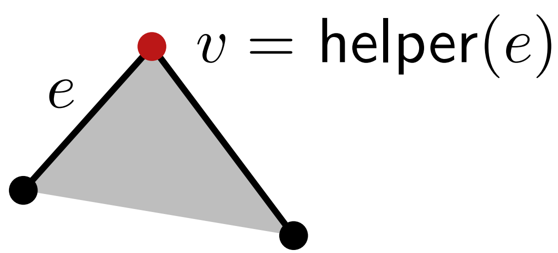
MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
  $v \leftarrow \mathcal{Q}$.nextVertex()
  $\mathcal{Q}$.deleteVertex(v)
  handleVertex(v)
return $\mathcal{D}$



handleStartVertex(vertex v)

$\mathcal{T} \leftarrow$ füge linke Kante e ein
helper(e) $\leftarrow v$



handleEndVertex(vertex v)

$e \leftarrow$ linke Kante
if isMergeVertex(helper(e)) **then**
  $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
lösche e aus $\mathcal{T}$

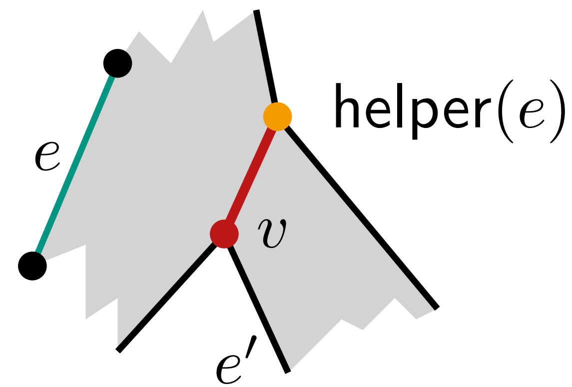
Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
    $v \leftarrow \mathcal{Q}$.nextVertex()
    $\mathcal{Q}$.deleteVertex(v)
    handleVertex(v)
return $\mathcal{D}$

handleSplitVertex(vertex v)

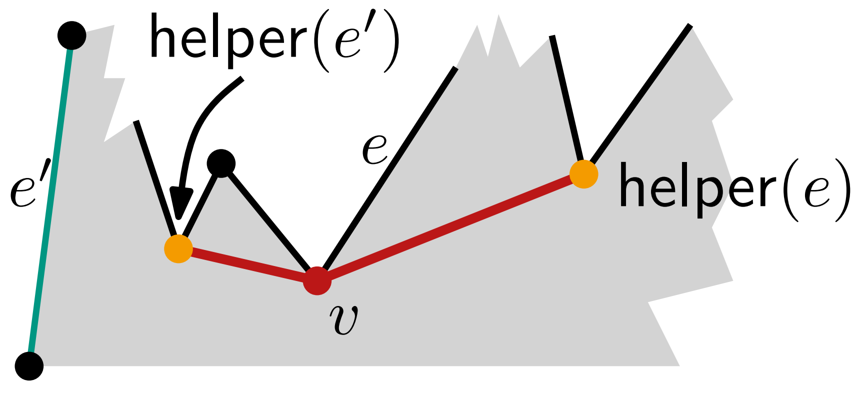
$e \leftarrow$ Kante links von v in $\mathcal{T}$
$\mathcal{D} \leftarrow$ füge (helper$(e), v)$ ein
helper$(e) \leftarrow v$
$\mathcal{T} \leftarrow$ füge rechte Kante e' von v ein
helper$(e') \leftarrow v$



Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
  $v \leftarrow \mathcal{Q}$.nextVertex()
  $\mathcal{Q}$.deleteVertex(v)
  handleVertex(v)
return $\mathcal{D}$



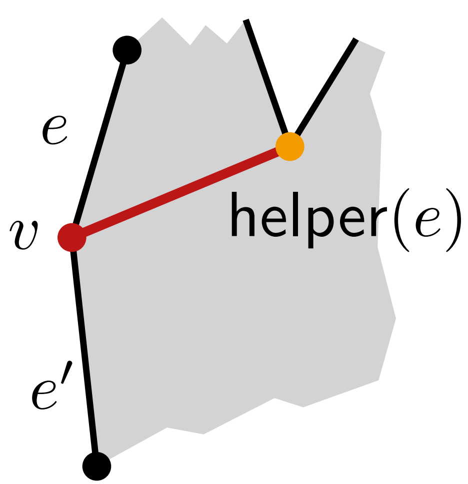
handleMergeVertex(vertex v)

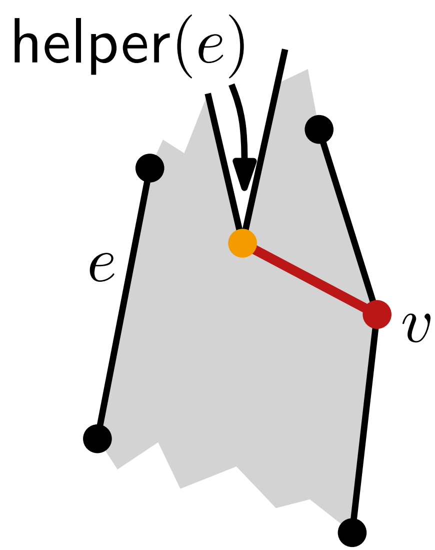
$e \leftarrow$ rechte Kante
if isMergeVertex(helper(e)) **then**
  $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
lösche e aus $\mathcal{T}$
$e' \leftarrow$ Kante links von v in $\mathcal{T}$
if isMergeVertex(helper(e')) **then**
  $\mathcal{D} \leftarrow$ füge (helper(e'), v) ein
helper(e') $\leftarrow v$

Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
    $v \leftarrow \mathcal{Q}$.nextVertex()
    $\mathcal{Q}$.deleteVertex(v)
    handleVertex(v)
return $\mathcal{D}$





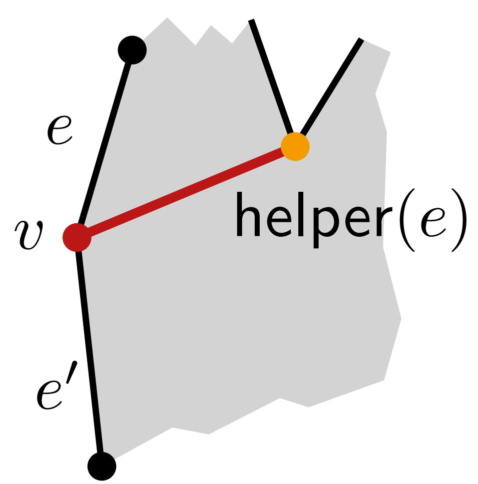
handleRegularVertex(vertex v)

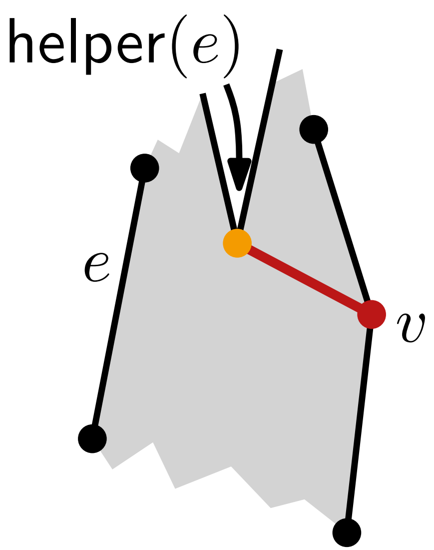
if P liegt lokal rechts von v **then**
    $e, e' \leftarrow$ obere, untere Kante
    if isMergeVertex(helper(e)) **then**
        $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
    lösche e aus $\mathcal{T}$
    $\mathcal{T} \leftarrow$ füge e' ein; helper(e') $\leftarrow v$
else
    $e \leftarrow$ Kante links von v in $\mathcal{T}$
    if isMergeVertex(helper(e)) **then**
        $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
    helper(e) $\leftarrow v$

Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
  $v \leftarrow \mathcal{Q}$.nextVertex()
  $\mathcal{Q}$.deleteVertex(v)
  handleVertex(v)
return $\mathcal{D}$





handleRegularVertex(vertex v)

if P liegt lokal rechts von v **then**
  $e, e' \leftarrow$ obere, untere Kante
  if isMergeVertex(helper(e)) **then**
    $\mathcal{D} \leftarrow$ füge (helper$(e), v$) ein
  lösche e aus $\mathcal{T}$
  $\mathcal{T} \leftarrow$ füge e' ein; helper$(e') \leftarrow v$
else
  $e \leftarrow$ Kante links von v in $\mathcal{T}$
  if isMergeVertex(helper(e)) **then**
    $\mathcal{D} \leftarrow$ füge (helper$(e), v$) ein
  helper$(e) \leftarrow v$

Lösung

2. Wie wählen wir in $O(1)$ die richtigen Kanten für das anpassen der *next*(e) bzw. *prev*(e) Einträge?

   a) Nur konstant viele Kanten inzident zu jedem Knoten

   - Initial hat jeder Knoten Grad 2
   - Jeder Knoten ist höchstens einmal helper: $+1$
   - Jeder Knoten ist höchstens einmal 'Sonderknoten': $+2$

   b) Durch passende Sortierung können wir die korrekten Kanten finden

Übersicht

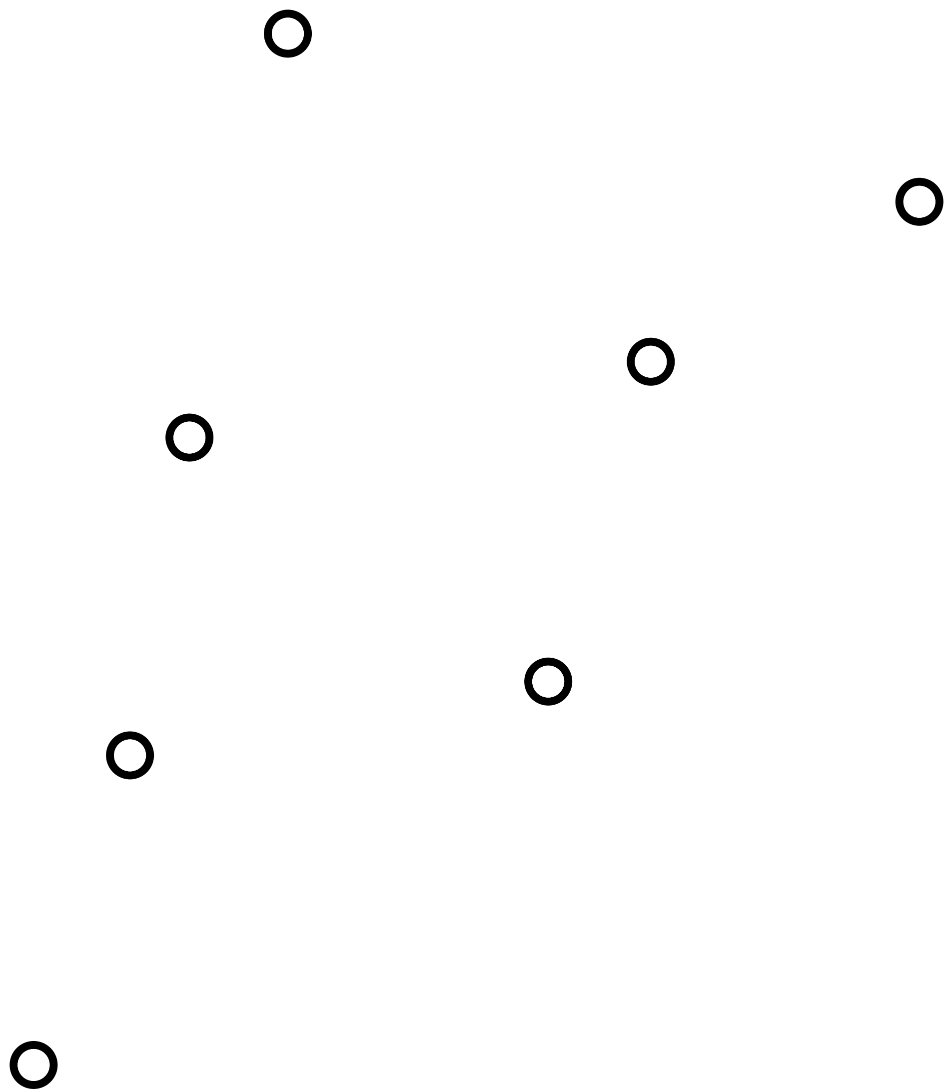
Übungsblatt 4

Nachtrag zu Übungsblatt 3

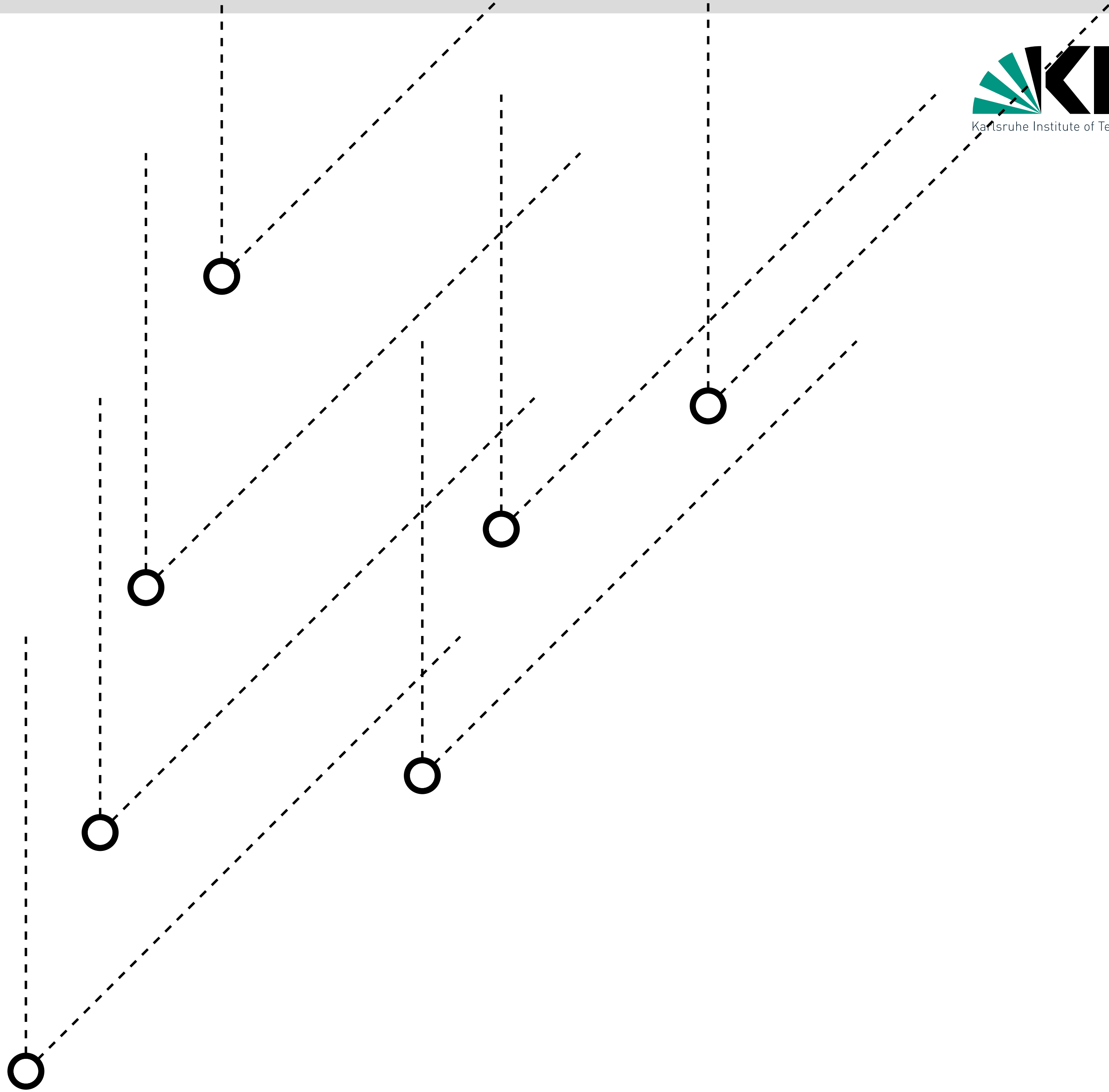
Übungsblatt 5

Werbung

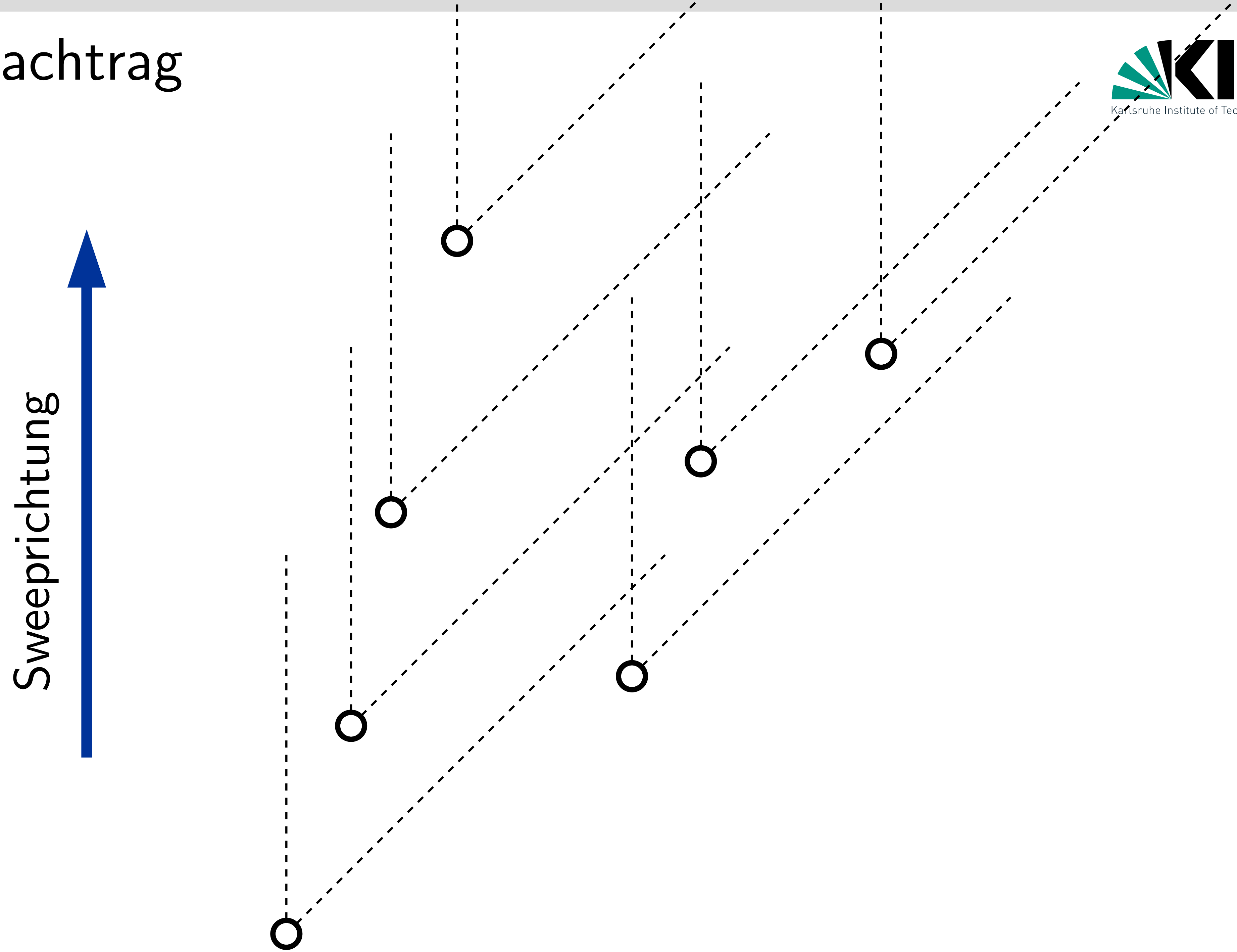
Nachtrag



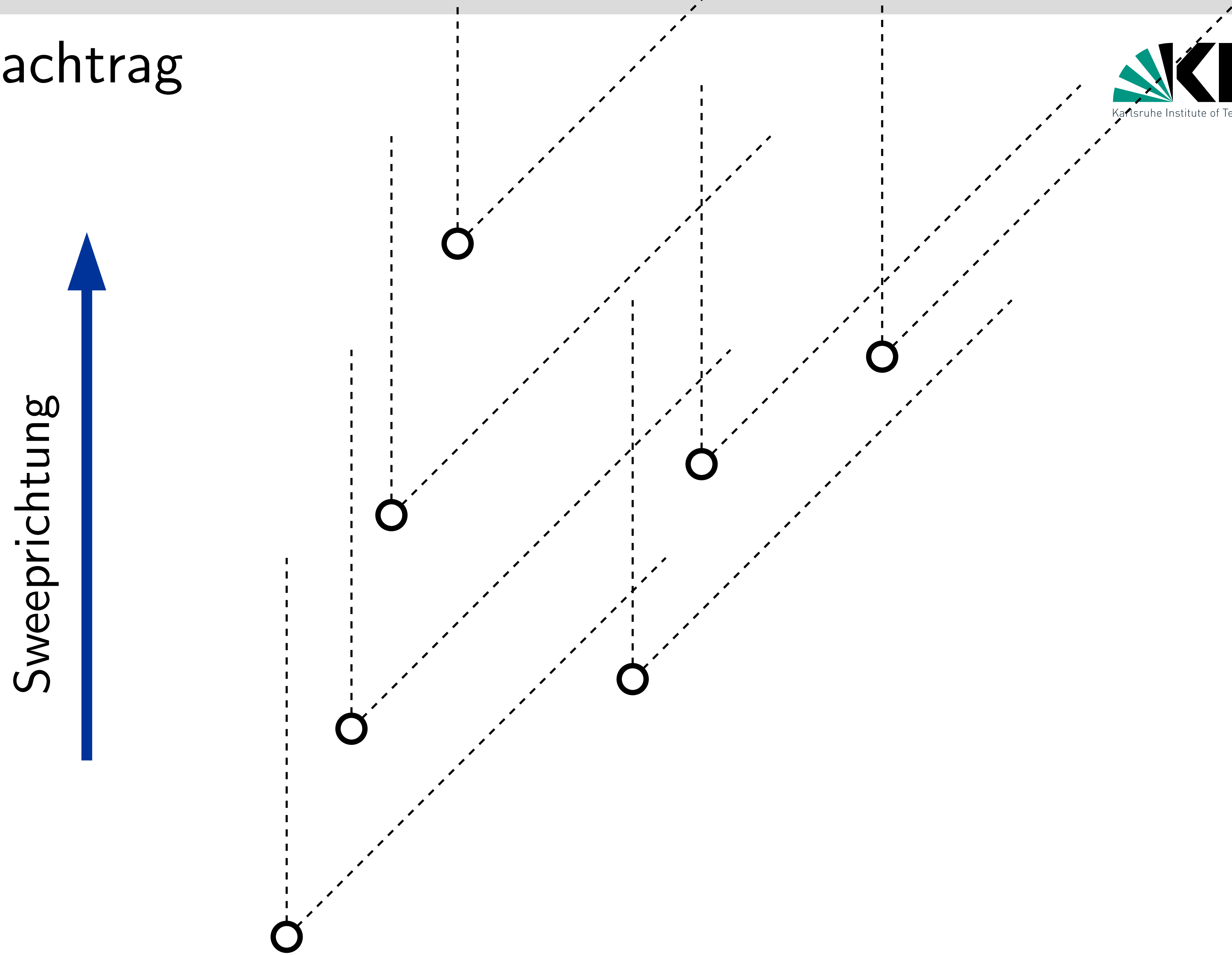
Nachtrag



Nachtrag

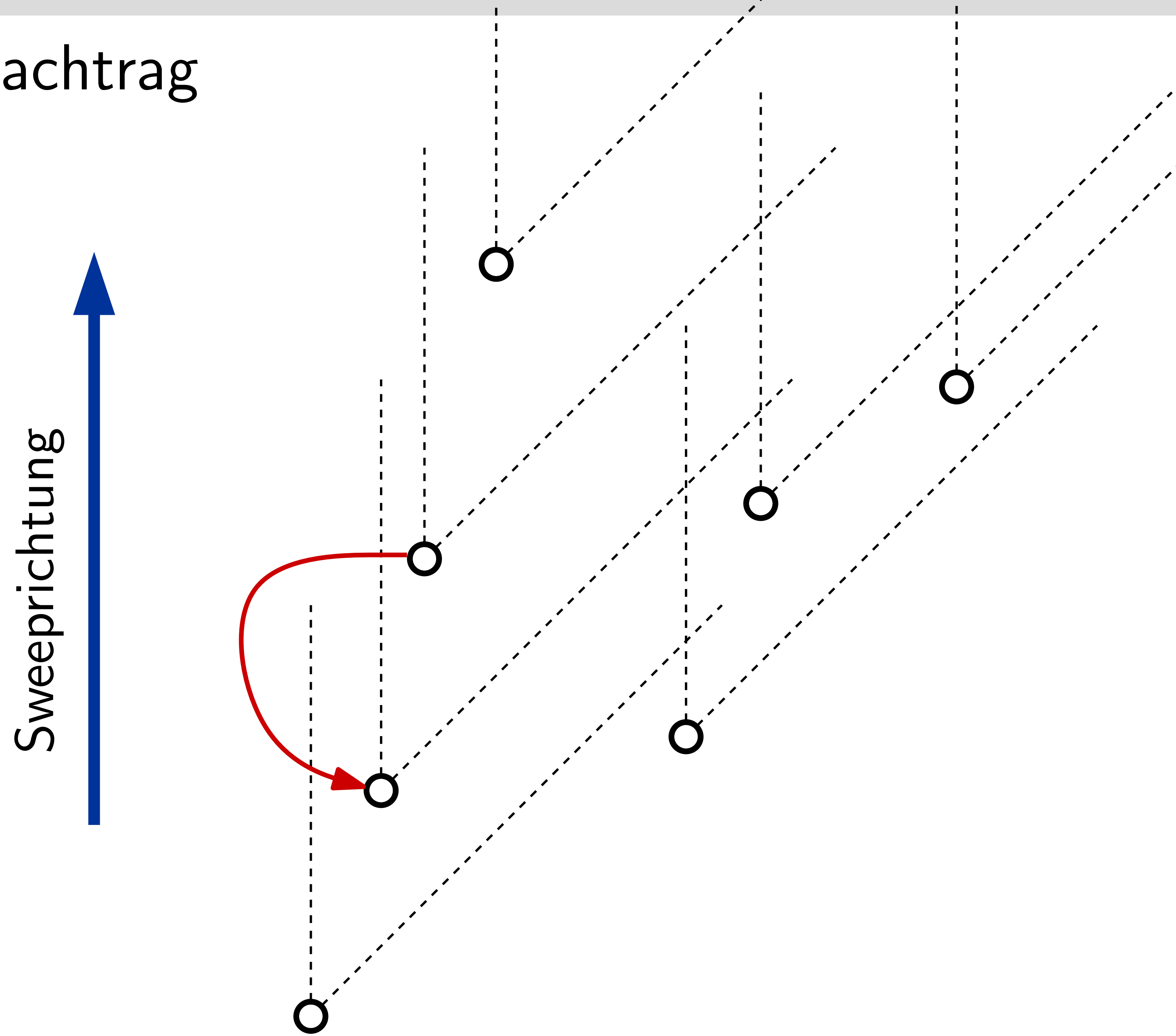


Nachtrag



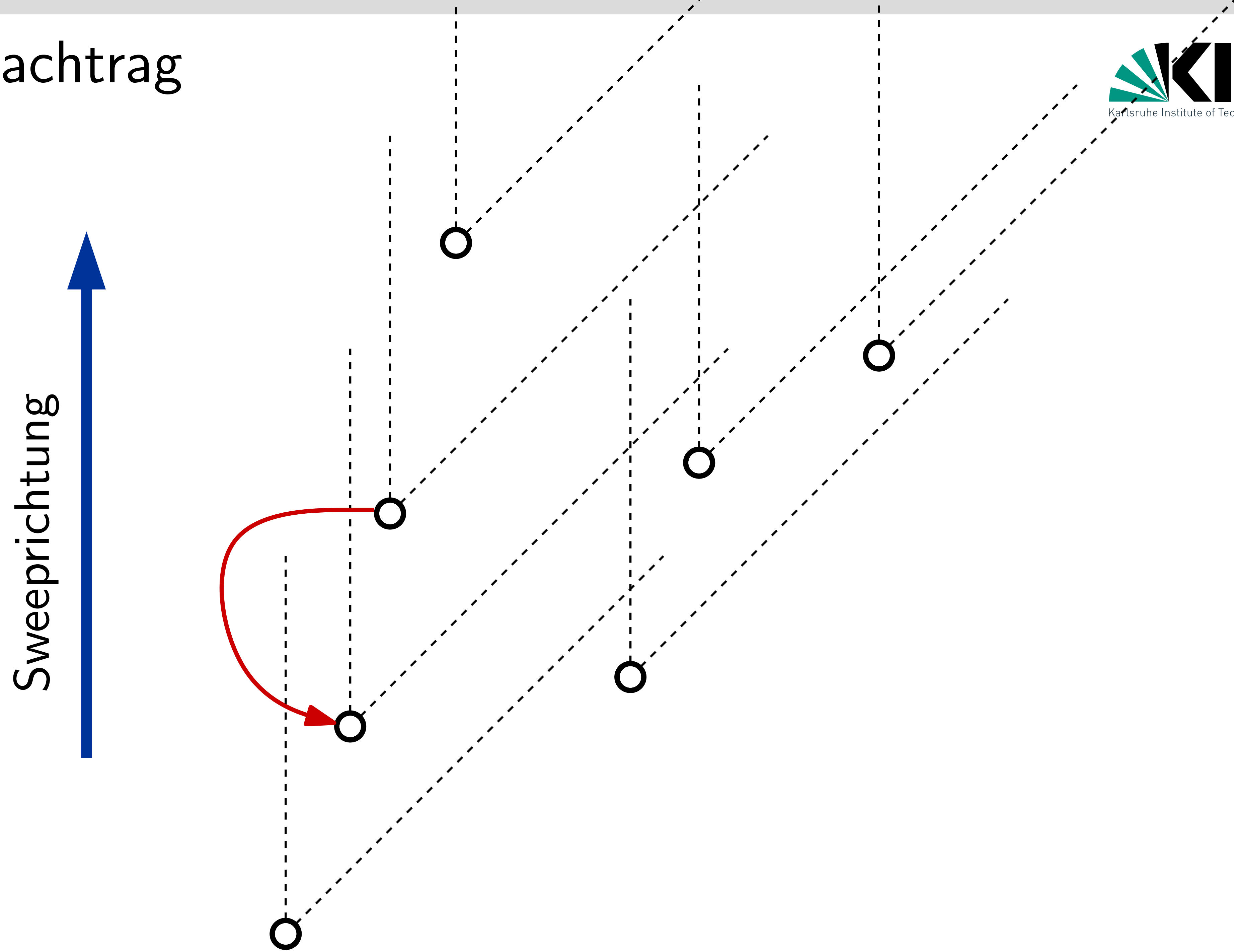
Idee: balancierter binärer Suchbaum T

Nachtrag



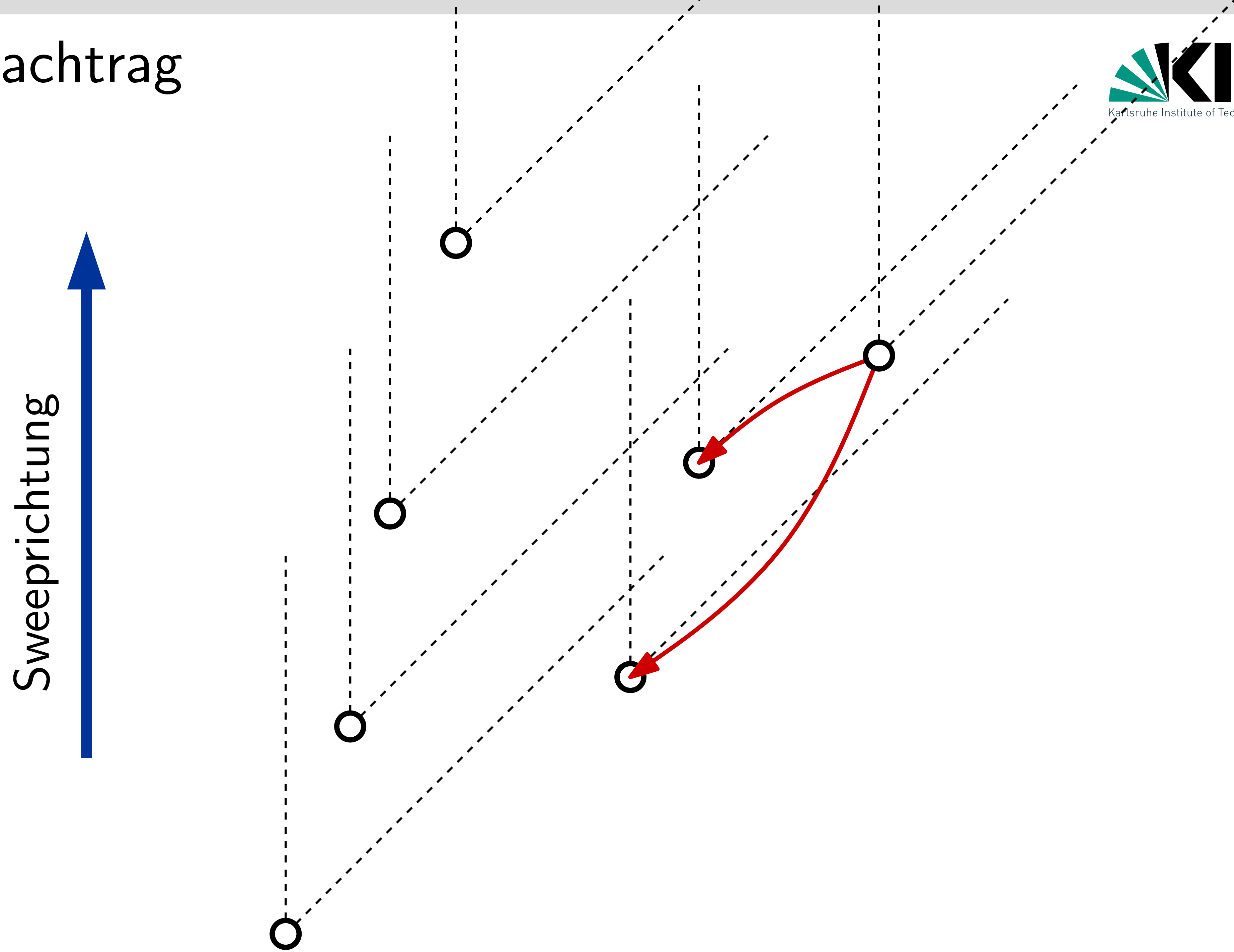
Idee: balancierter binärer Suchbaum T

Nachtrag



Idee: balancierter binärer Suchbaum T

Nachtrag



Idee: balancierter binärer Suchbaum T

Übersicht

Übungsblatt 4

Nachtrag zu Übungsblatt 3

Übungsblatt 5

Werbung

Aufgabe 1

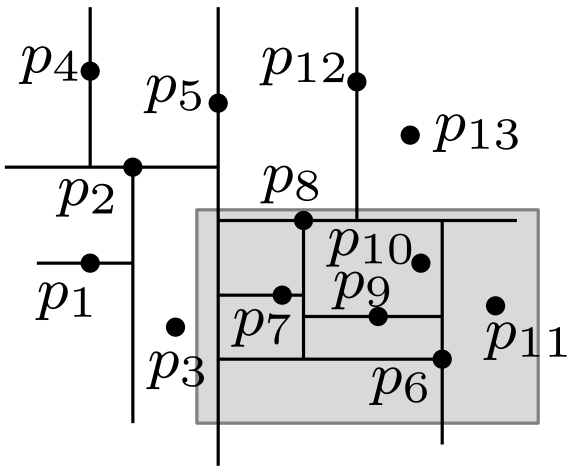
kd-Trees – w-c Laufzeit

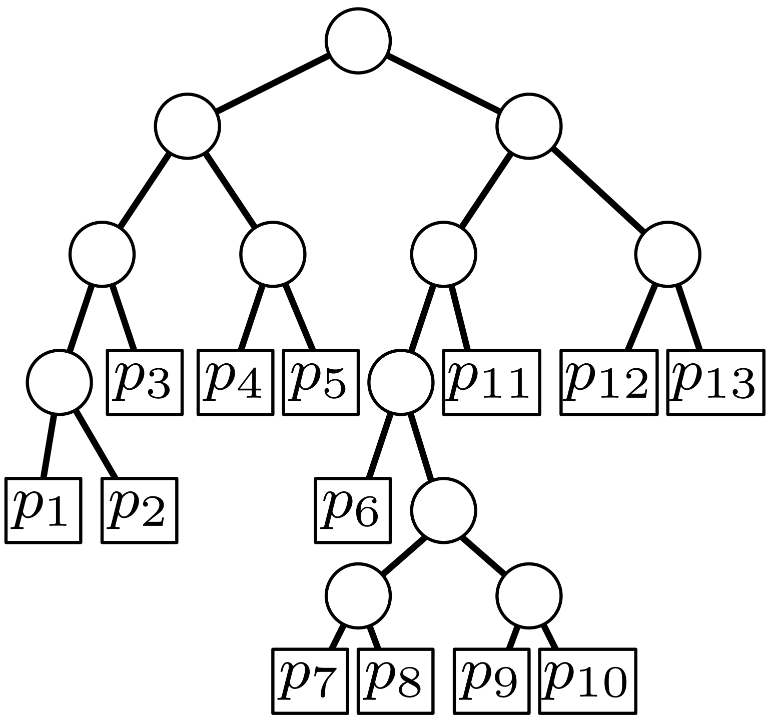
Anfragen

a) warum?

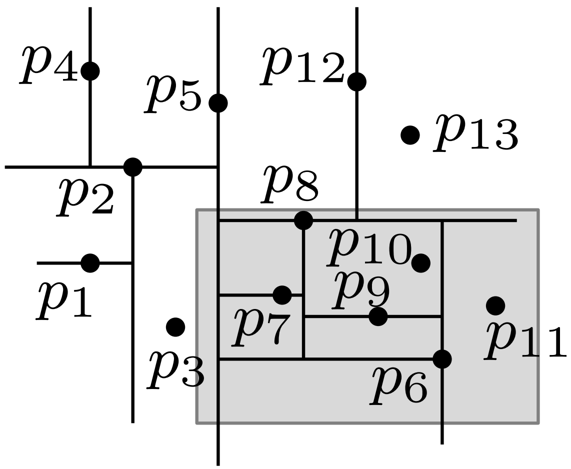
$$Q(n) = \begin{cases} \mathcal{O}(1) & \text{, für } n = 1 \\ 2 + 2Q(n/4) & \text{, für } n > 1 \end{cases}$$

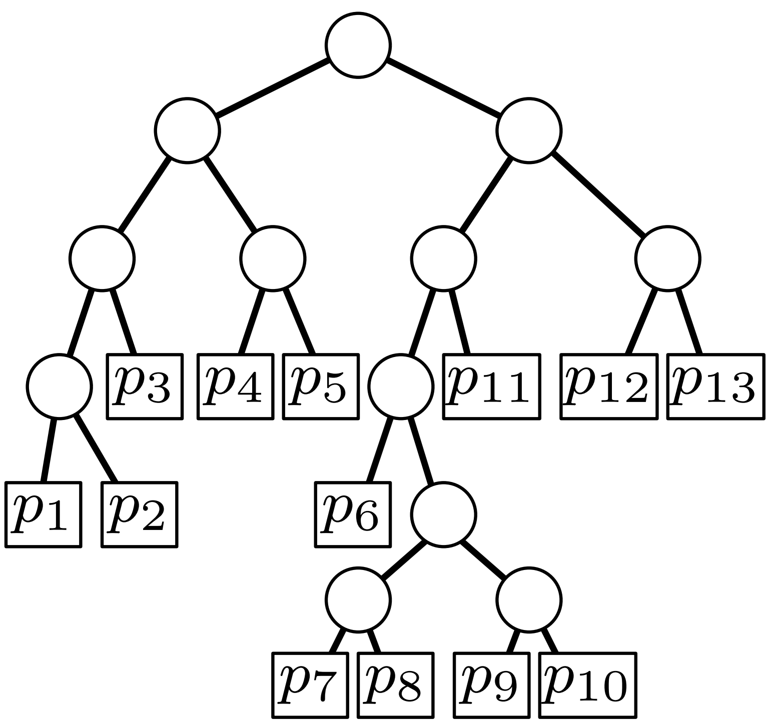
Bereichsabfrage in einem kd-Tree





Bereichsabfrage in einem kd-Tree

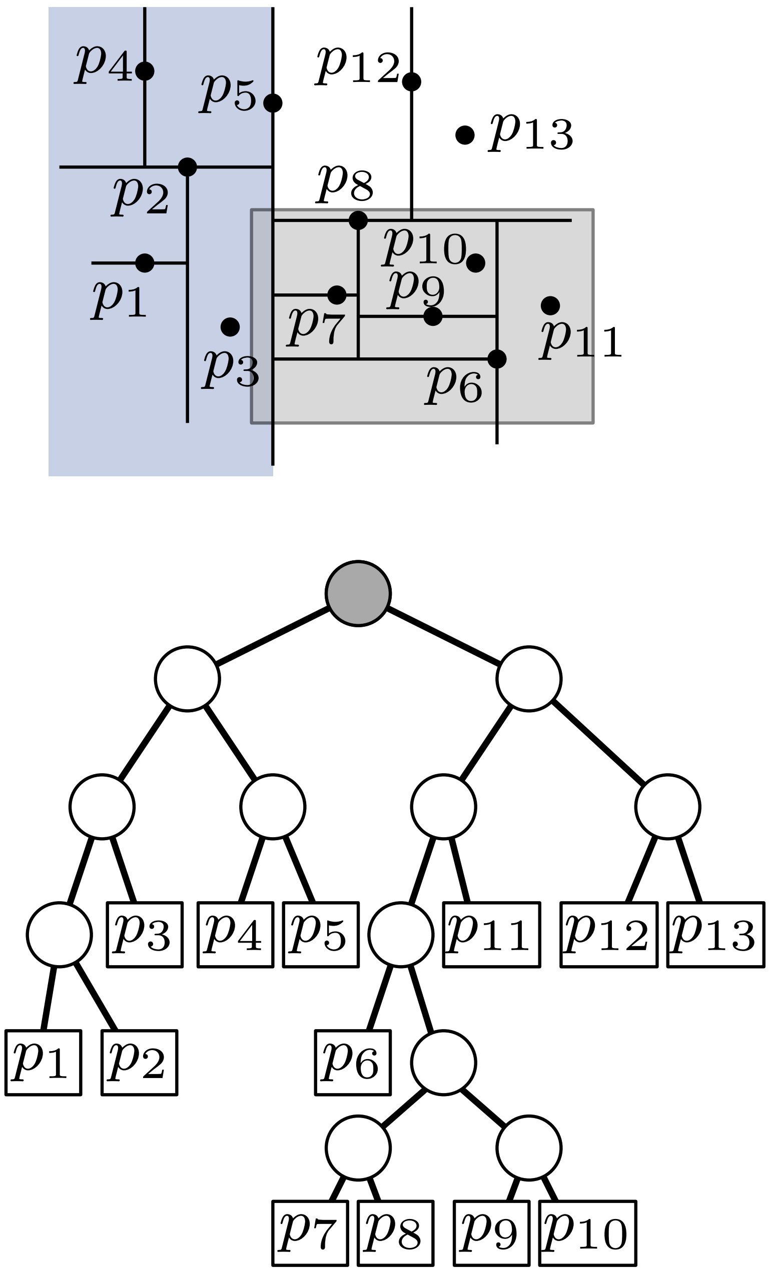




SearchKdTree(v, R)

  if v Blatt **then**
    prüfe Punkt p in v auf $p \in R$
  else
    if region(lc(v)) $\subseteq R$ **then**
      ReportSubtree(lc(v))
    else
      if region(lc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(lc(v), R)
    if region(rc(v)) $\subseteq R$ **then**
      ReportSubtree(rc(v))
    else
      if region(rc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(rc(v), R)

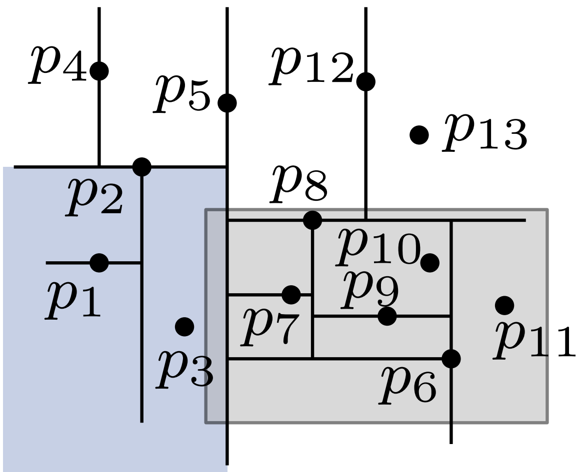
Bereichsabfrage in einem kd-Tree

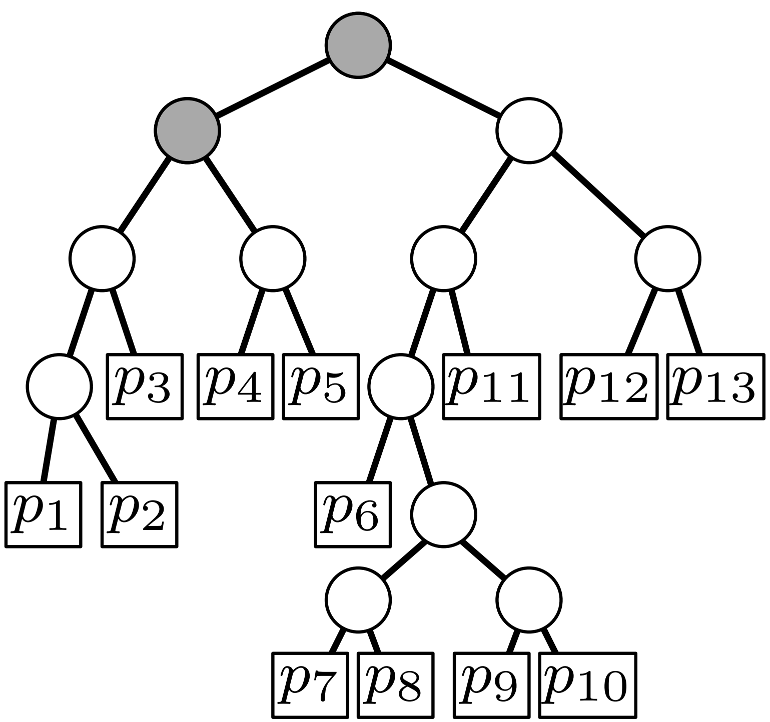


SearchKdTree(v, R)

if v Blatt **then**
  prüfe Punkt p in v auf $p \in R$
else
  if region(lc(v)) $\subseteq R$ **then**
    ReportSubtree(lc(v))
  else
    if region(lc(v)) $\cap R \neq \emptyset$ **then**
    SearchKdTree(lc(v), R)
  if region(rc(v)) $\subseteq R$ **then**
    ReportSubtree(rc(v))
  else
    if region(rc(v)) $\cap R \neq \emptyset$ **then**
    SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

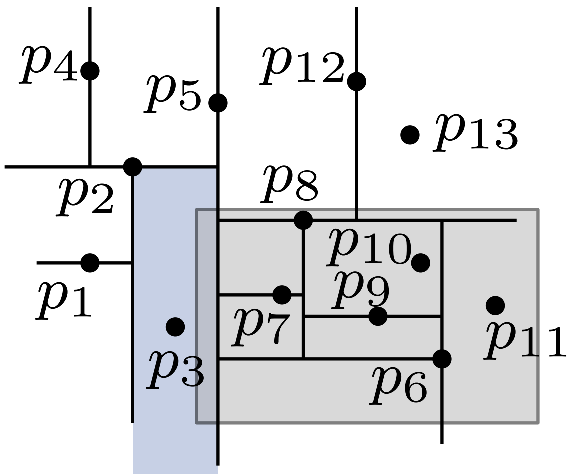


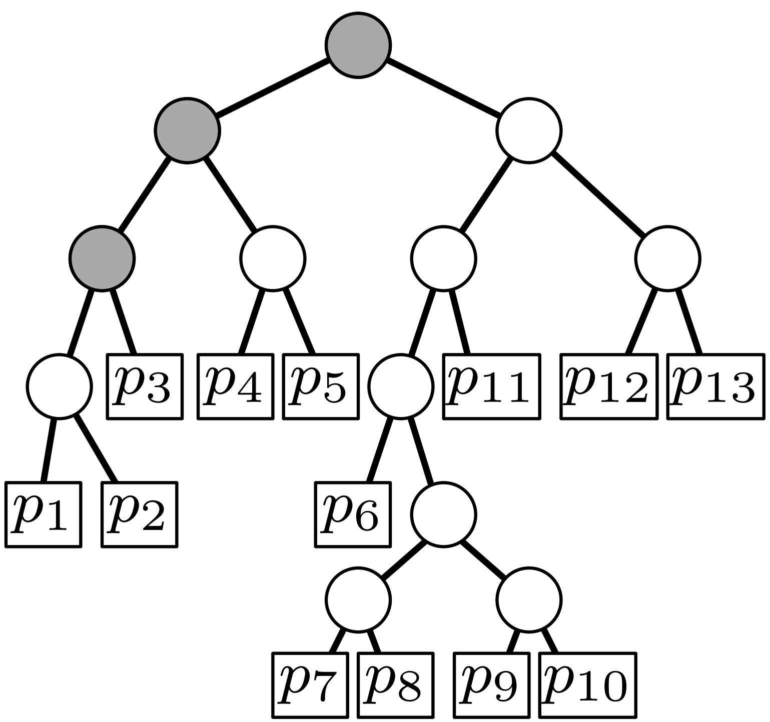


SearchKdTree(v, R)

if v Blatt **then**
    prüfe Punkt p in v auf $p \in R$
else
    if region(lc(v)) $\subseteq R$ **then**
        ReportSubtree(lc(v))
    else
        if region(lc(v)) $\cap R \neq \emptyset$ **then**
        SearchKdTree(lc(v), R)
    if region(rc(v)) $\subseteq R$ **then**
        ReportSubtree(rc(v))
    else
        if region(rc(v)) $\cap R \neq \emptyset$ **then**
        SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

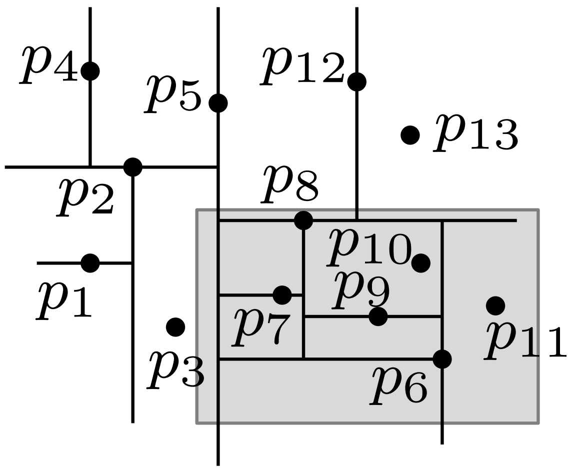


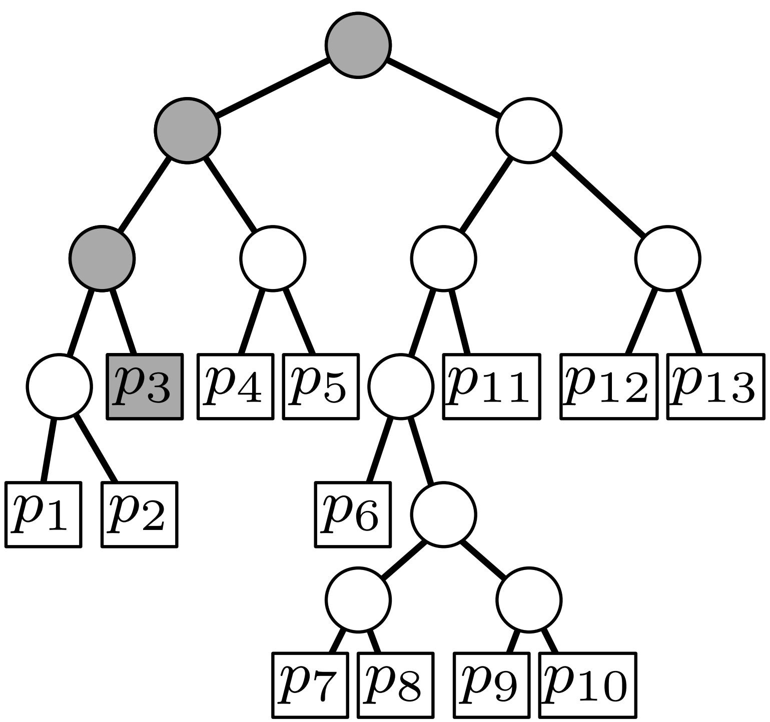


SearchKdTree(v, R)

if v Blatt **then**
    prüfe Punkt p in v auf $p \in R$
else
    if region(lc(v)) $\subseteq R$ **then**
        ReportSubtree(lc(v))
    else
        if region(lc(v)) $\cap R \neq \emptyset$ **then**
        SearchKdTree(lc(v), R)
    if region(rc(v)) $\subseteq R$ **then**
        ReportSubtree(rc(v))
    else
        if region(rc(v)) $\cap R \neq \emptyset$ **then**
        SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

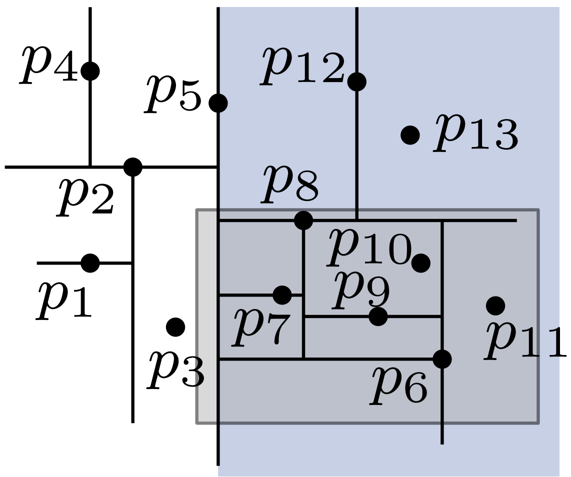


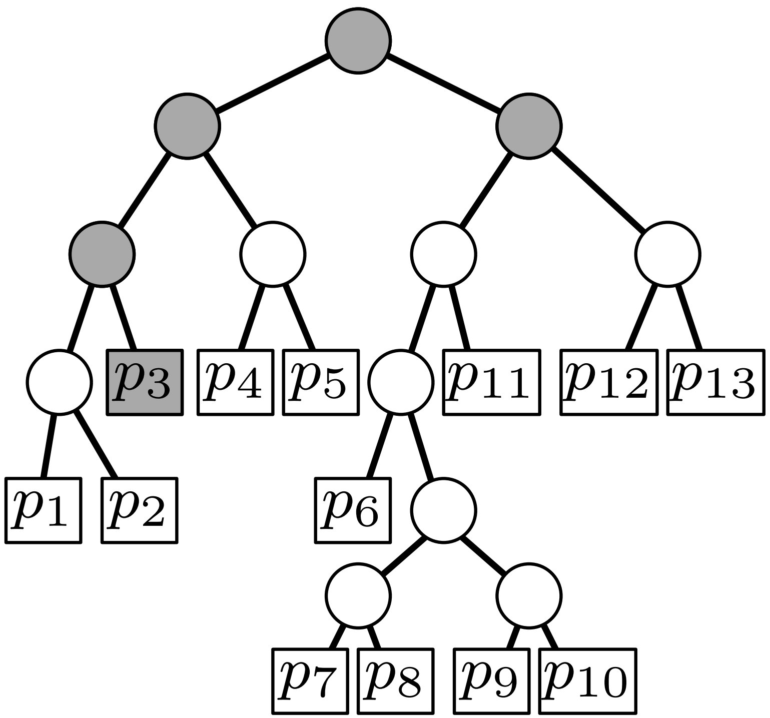


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

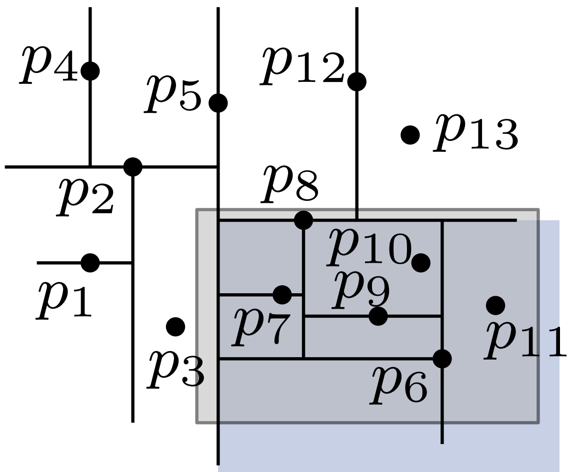


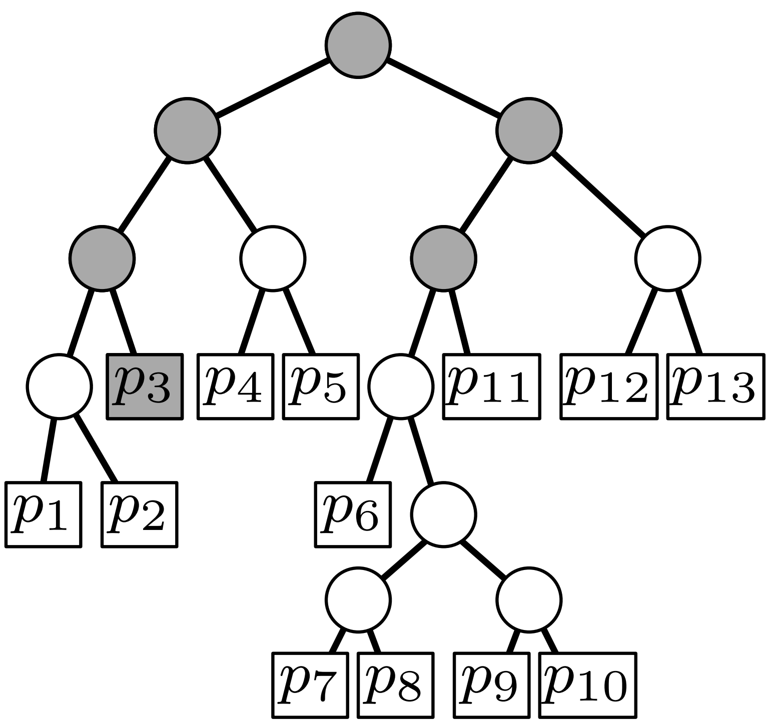


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

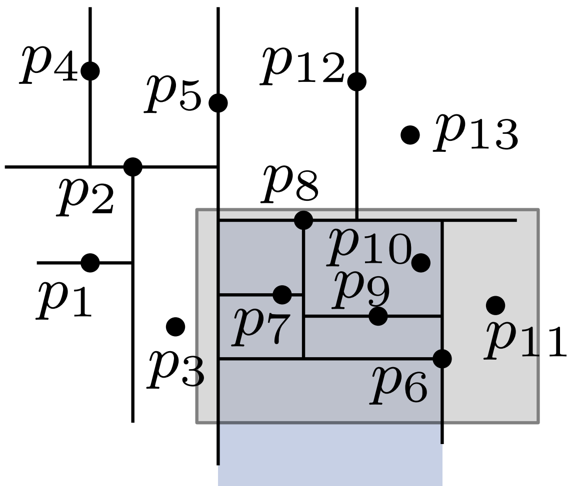


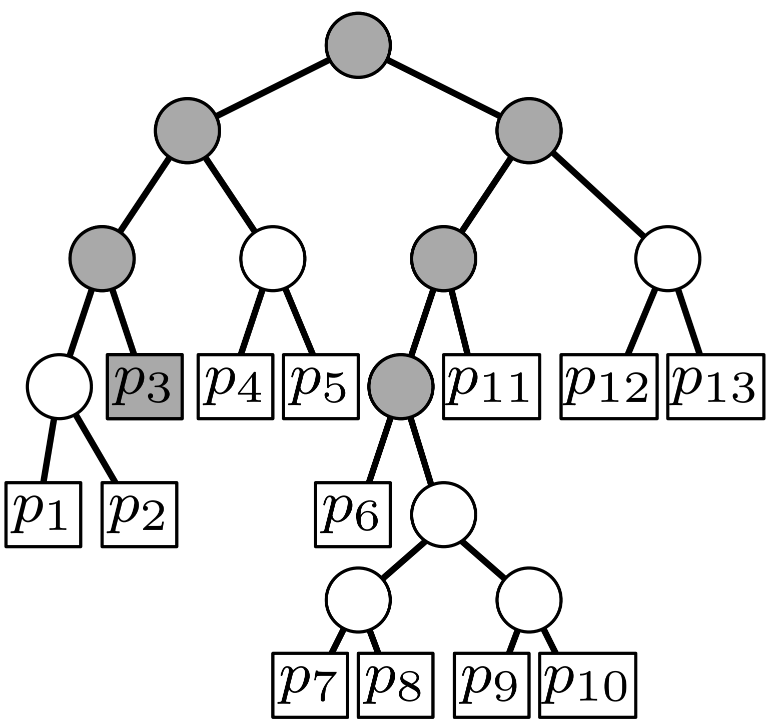


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

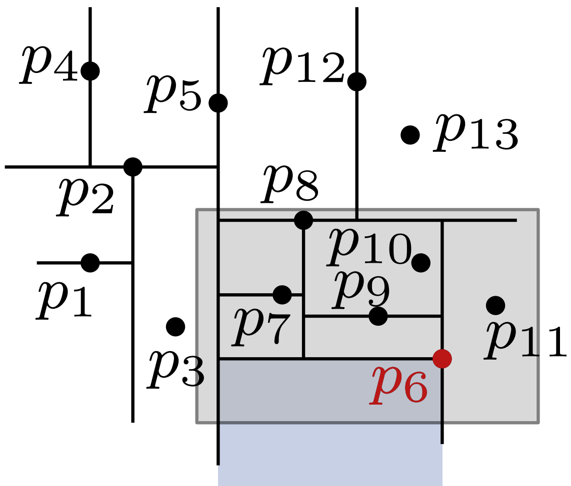


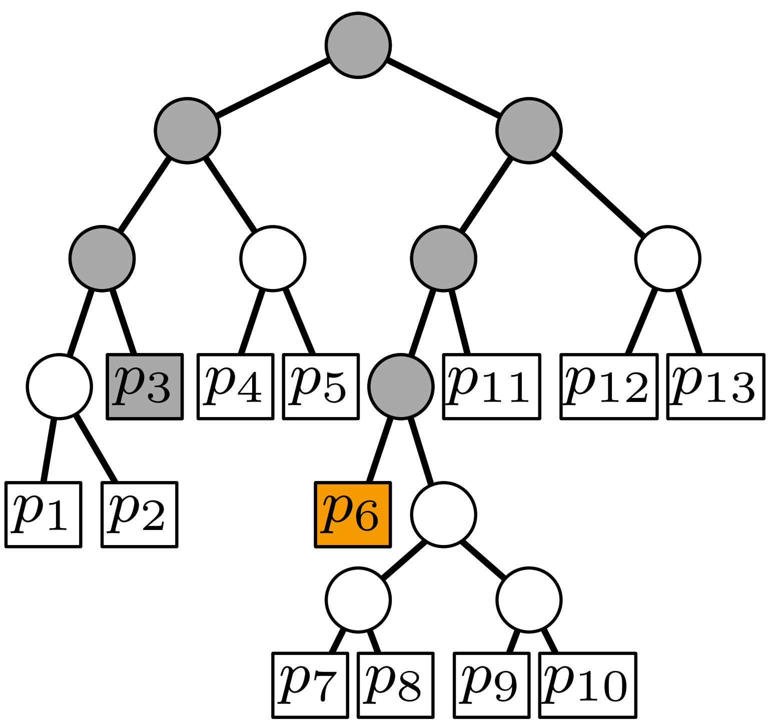


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

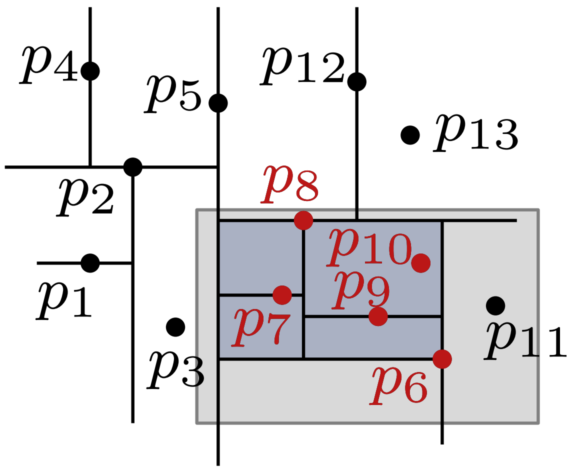


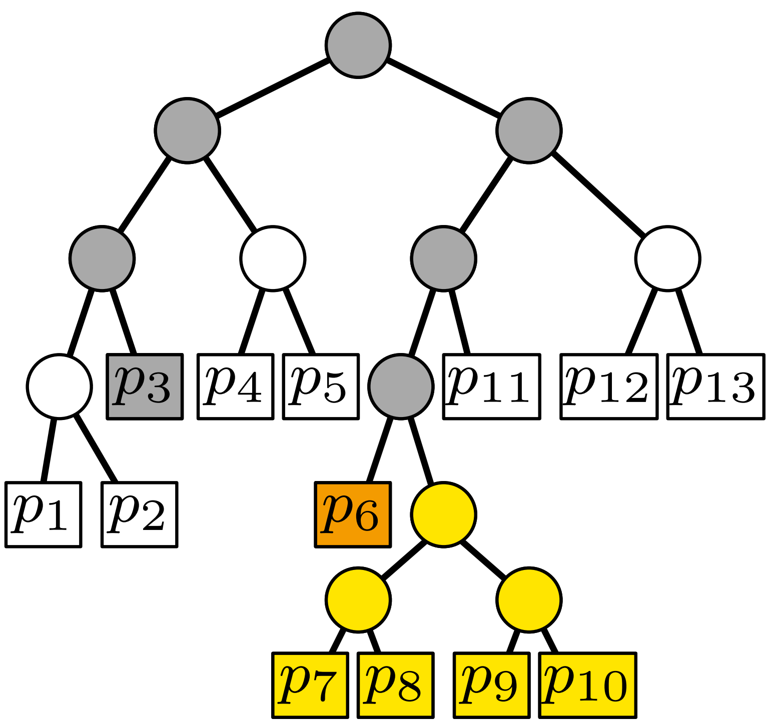


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

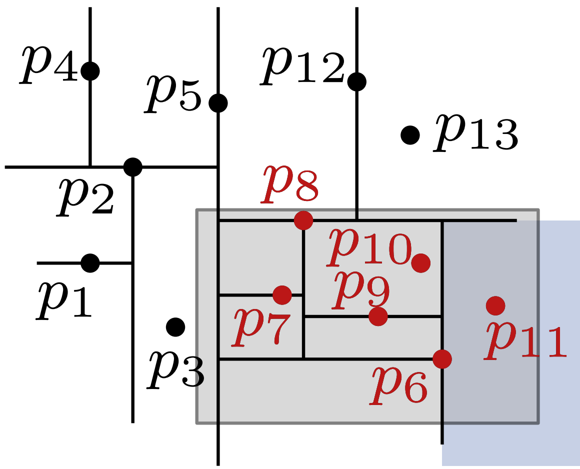


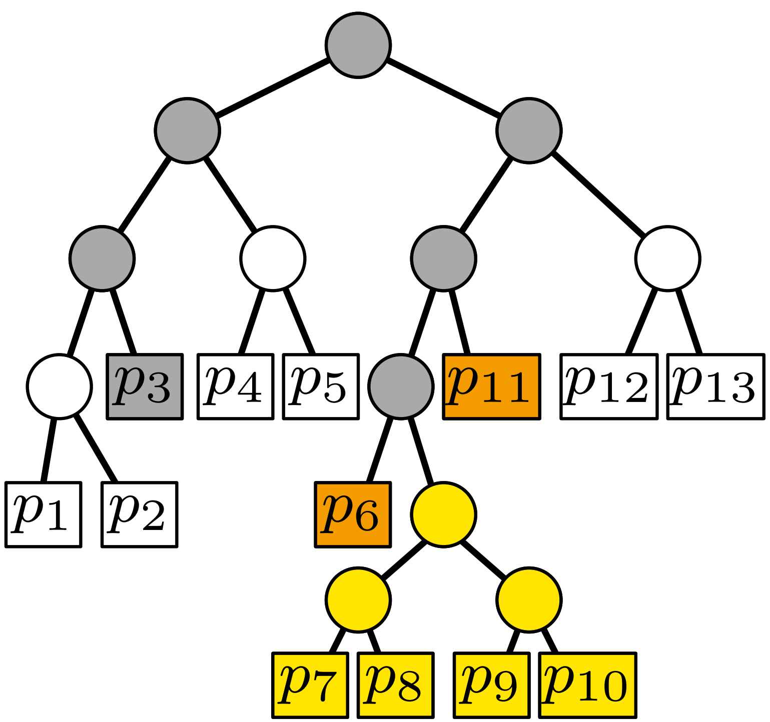


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree





SearchKdTree(v, R)

if v Blatt **then**

  prüfe Punkt p in v auf $p \in R$

else

  if region(lc(v)) $\subseteq R$ **then**

    ReportSubtree(lc(v))

  else

    if region(lc(v)) $\cap R \neq \emptyset$ **then**

    SearchKdTree(lc(v), R)

  if region(rc(v)) $\subseteq R$ **then**

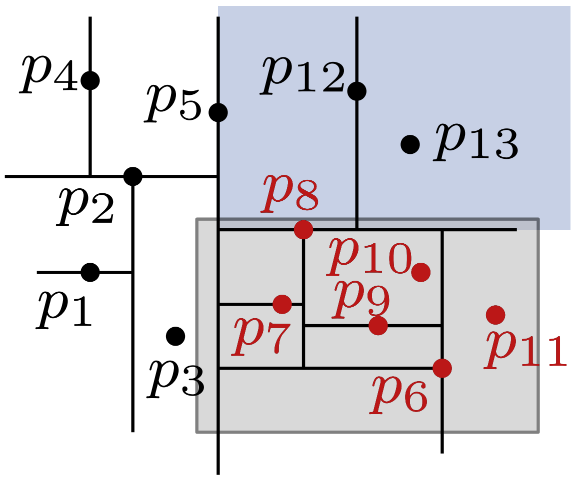
    ReportSubtree(rc(v))

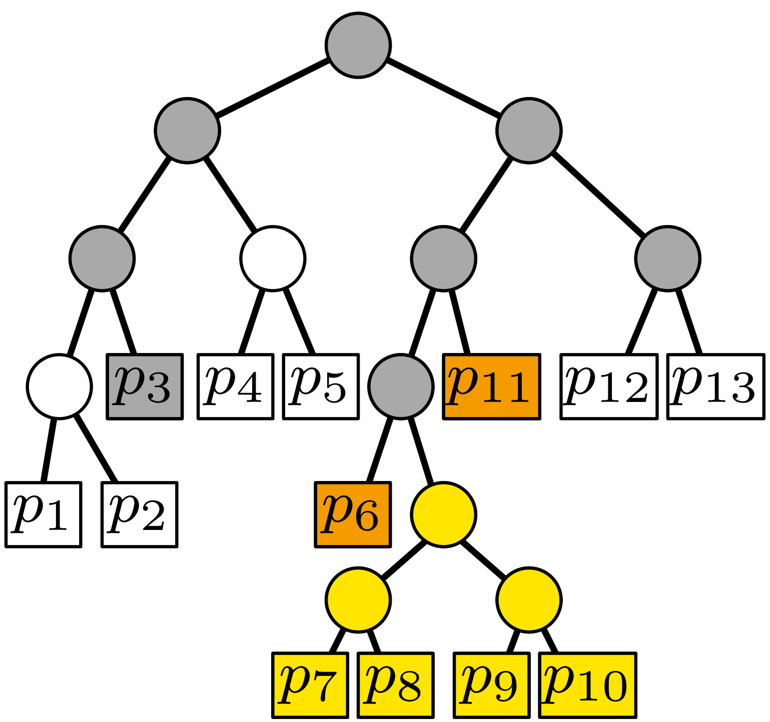
  else

    if region(rc(v)) $\cap R \neq \emptyset$ **then**

    SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

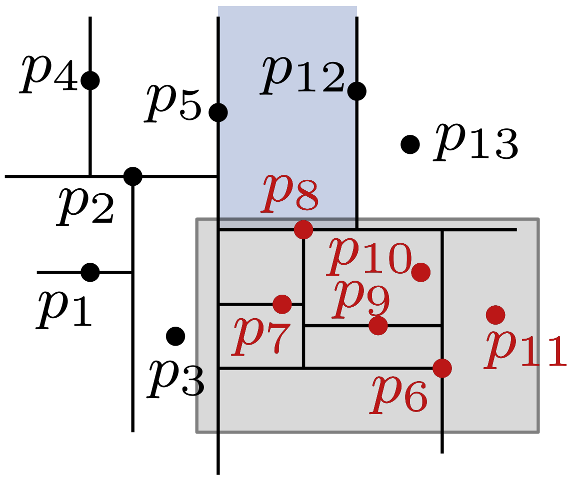


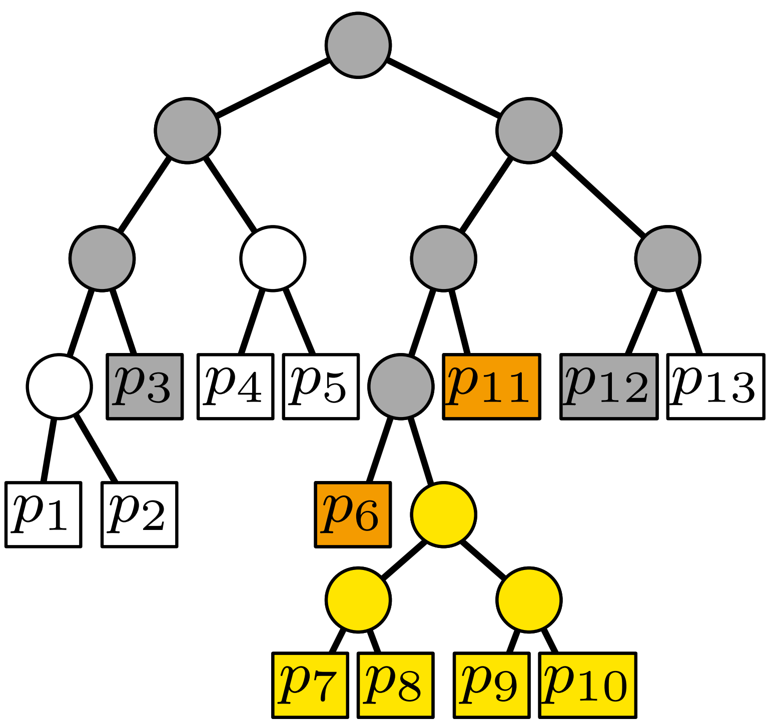


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

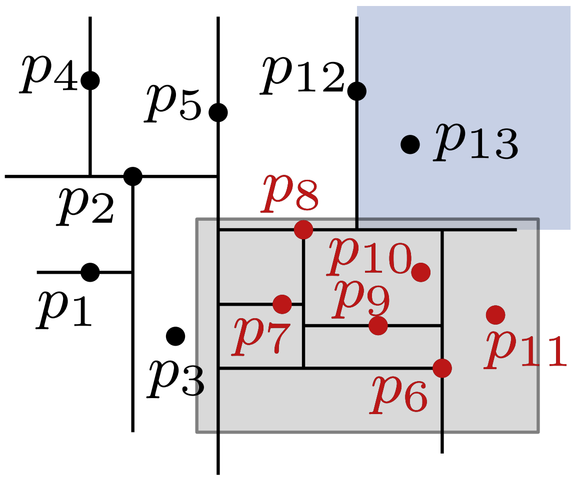


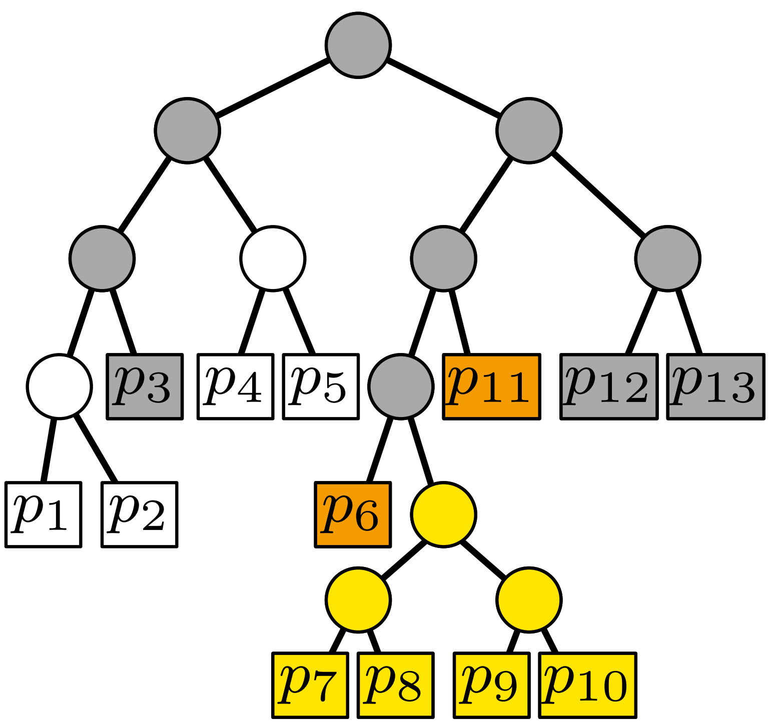


SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

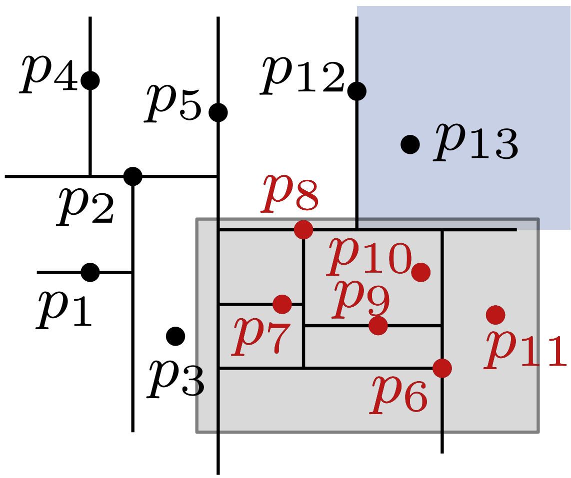


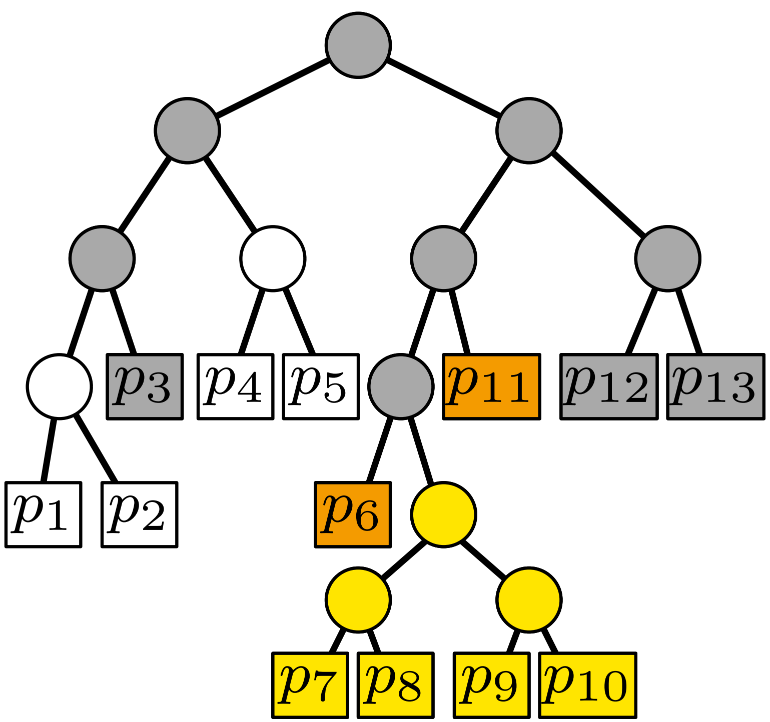


SearchKdTree(v, R)

if v Blatt **then**
  prüfe Punkt p in v auf $p \in R$
else
  if region(lc(v)) $\subseteq R$ **then**
    ReportSubtree(lc(v))
  else
    if region(lc(v)) $\cap R \neq \emptyset$ **then**
    SearchKdTree(lc(v), R)
  if region(rc(v)) $\subseteq R$ **then**
    ReportSubtree(rc(v))
  else
    if region(rc(v)) $\cap R \neq \emptyset$ **then**
    SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

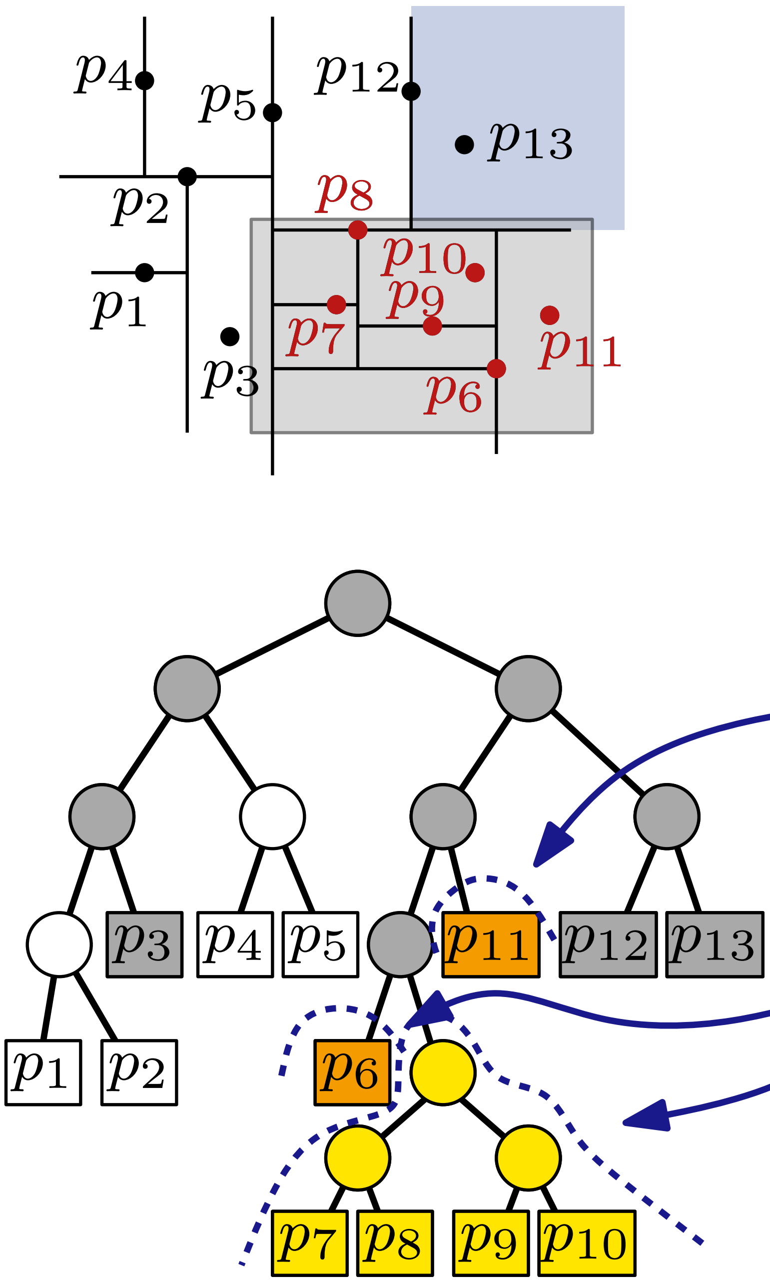




SearchKdTree(v, R)

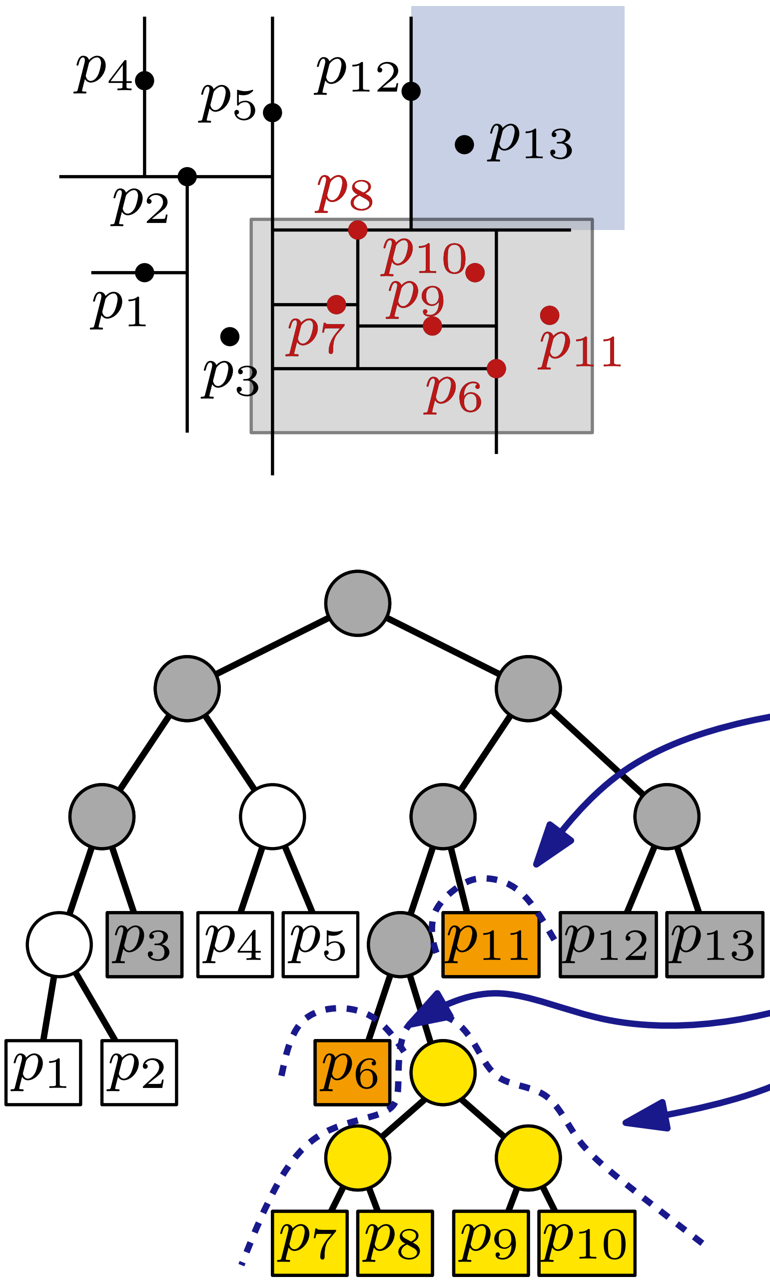
  if v Blatt **then**
    prüfe Punkt p in v auf $p \in R$
  else
    if region(lc(v)) $\subseteq R$ **then**
      ReportSubtree(lc(v))
    else
      if region(lc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(lc(v), R)
    if region(rc(v)) $\subseteq R$ **then**
      ReportSubtree(rc(v))
    else
      if region(rc(v)) $\cap R \neq \emptyset$ **then**
      SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree



ReportSubtree in $\mathcal{O}(k)$

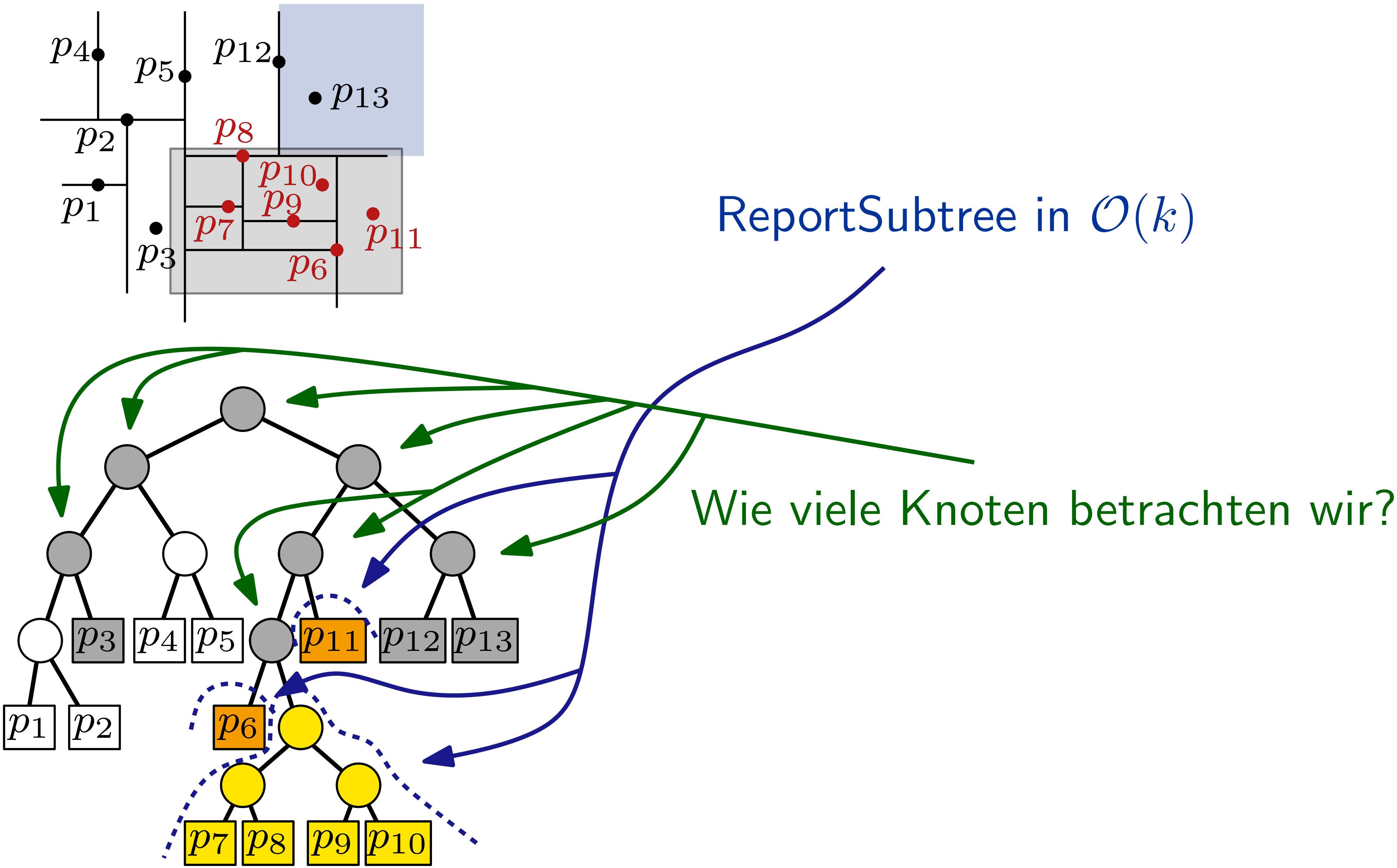
Bereichsabfrage in einem kd-Tree



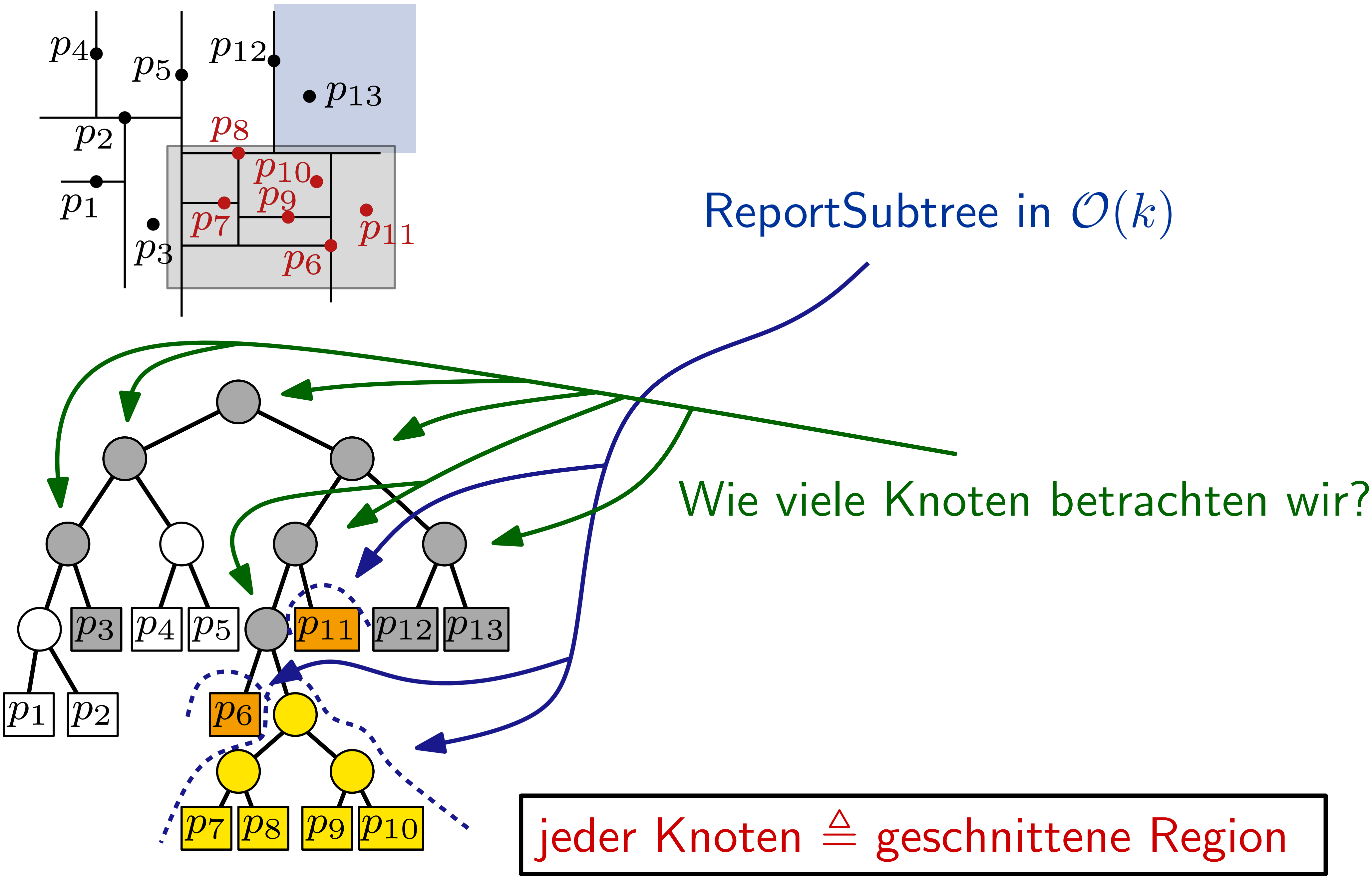
ReportSubtree in $\mathcal{O}(k)$

Wie viele Knoten betrachten wir?

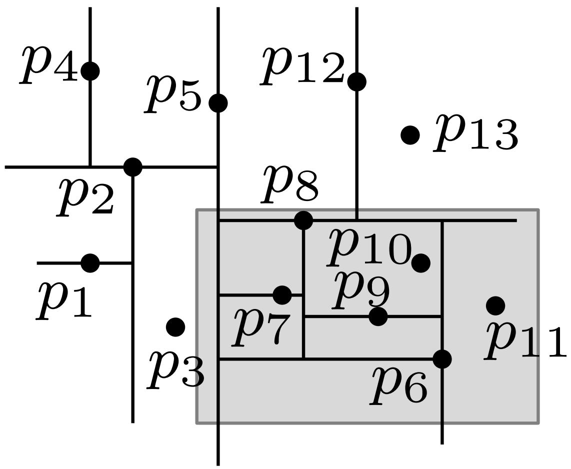
Bereichsabfrage in einem kd-Tree

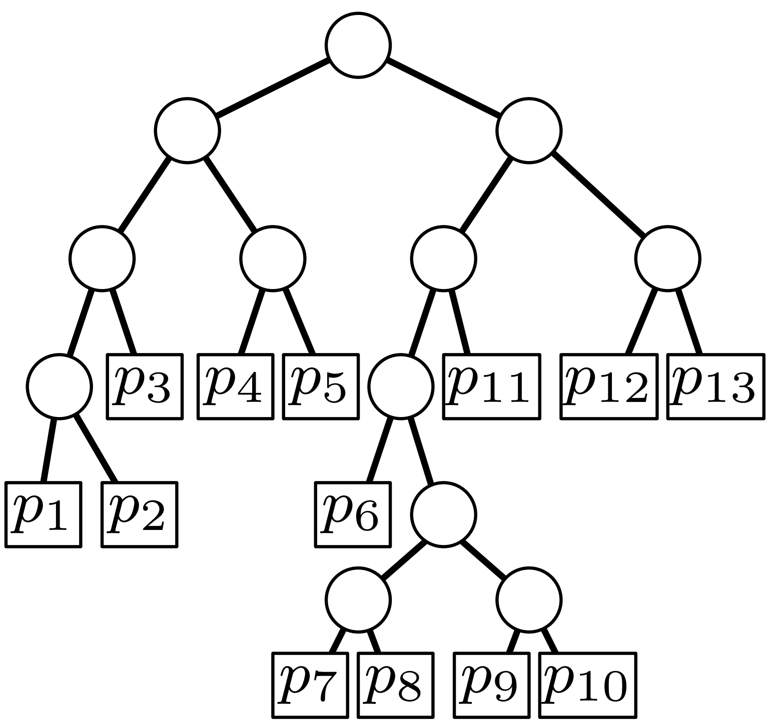


Bereichsabfrage in einem kd-Tree



Bereichsabfrage in einem kd-Tree

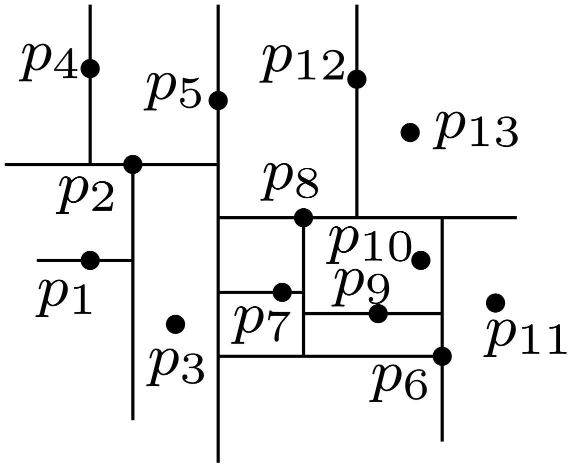




SearchKdTree(v, R)

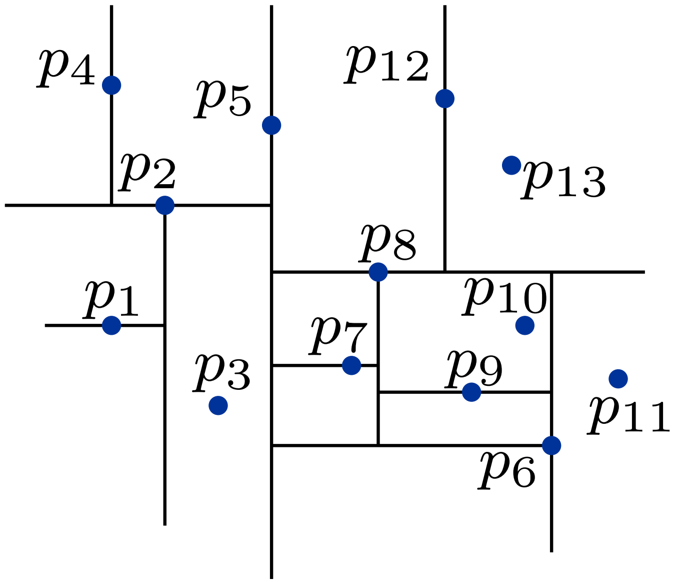
- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc(v), R)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc(v), R)

Bereichsabfrage in einem kd-Tree

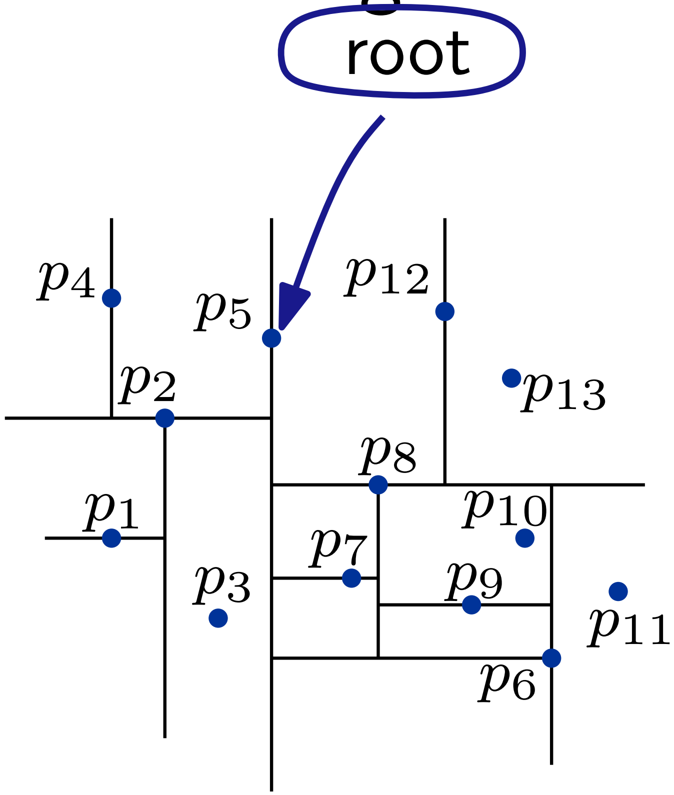


Wie viele Regionen schneidet eine vertikale Linie?

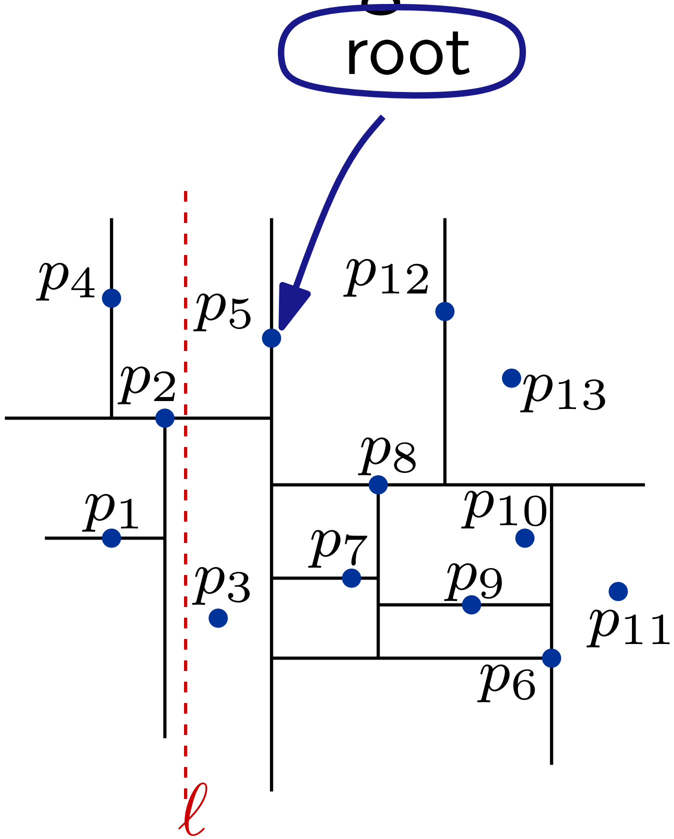
Bereichsabfrage in einem kd-Tree



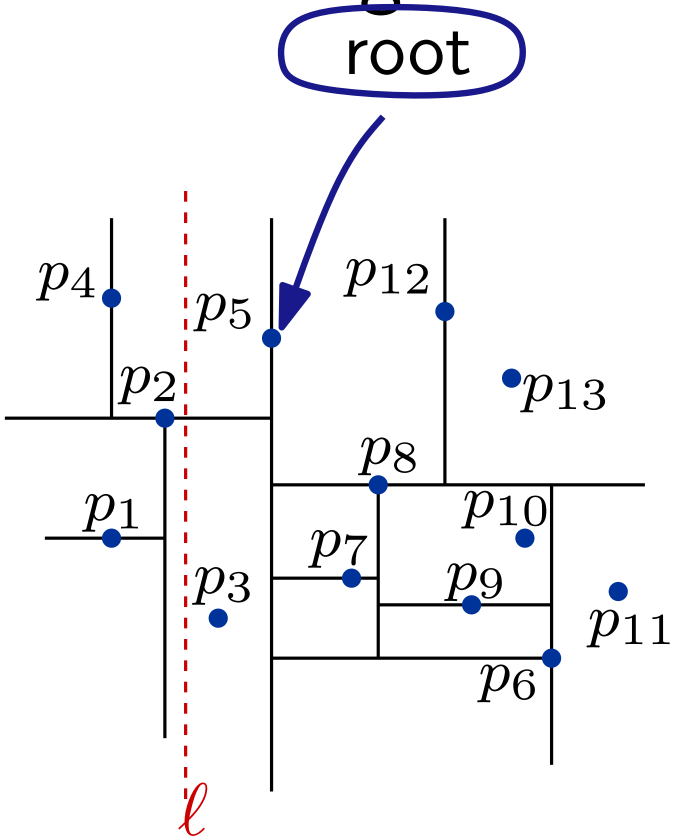
Bereichsabfrage in einem kd-Tree



Bereichsabfrage in einem kd-Tree



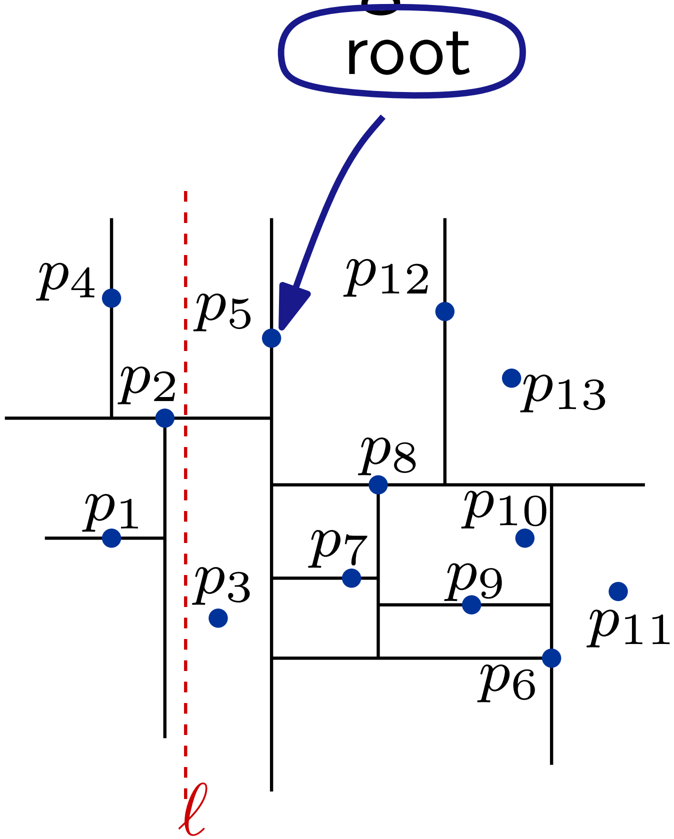
Bereichsabfrage in einem kd-Tree



Vermutung:

$$Q(n) = 1 + Q(n/2)$$

Bereichsabfrage in einem kd-Tree



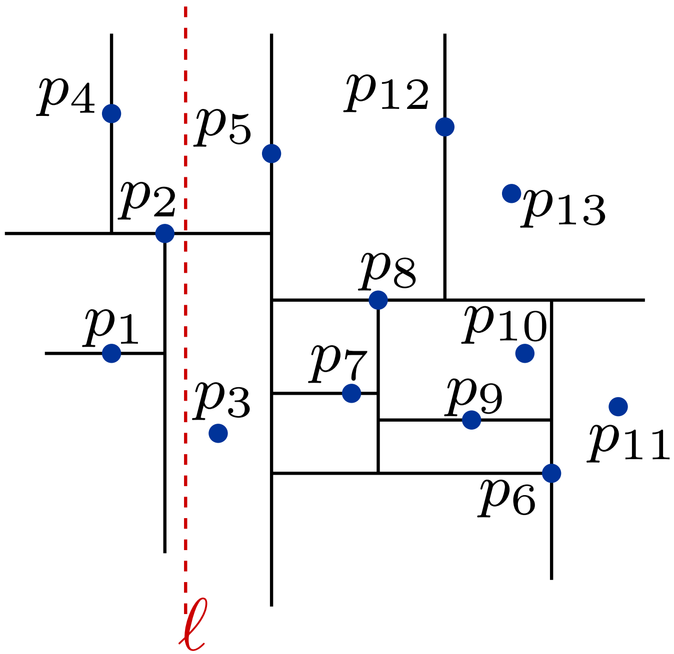
Vermutung:

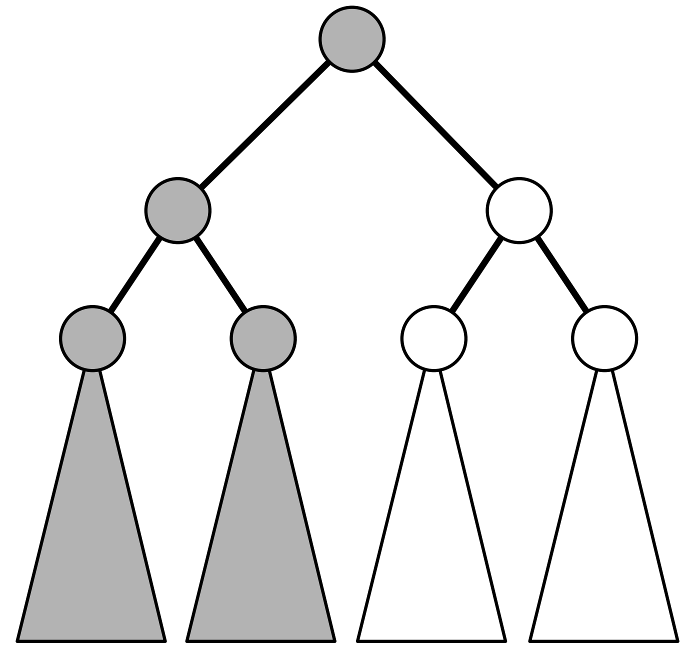
$$Q(n) = 1 + Q(n/2)$$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





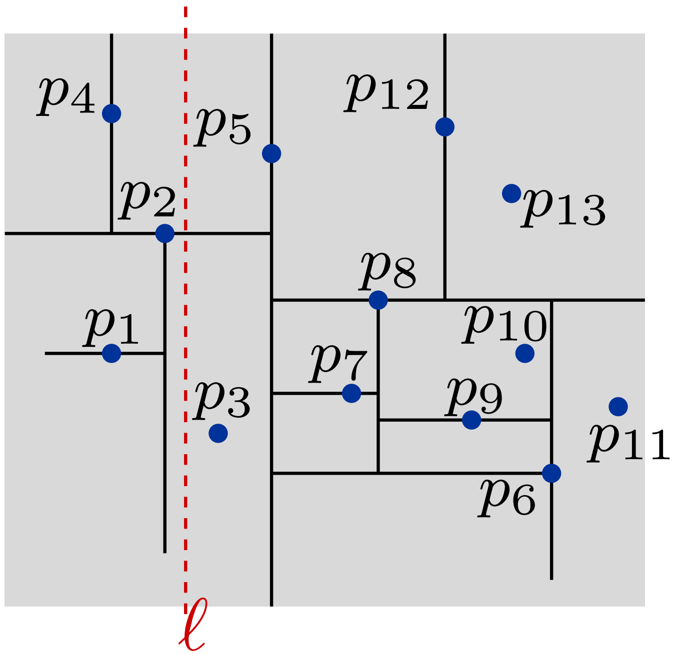
Vermutung:

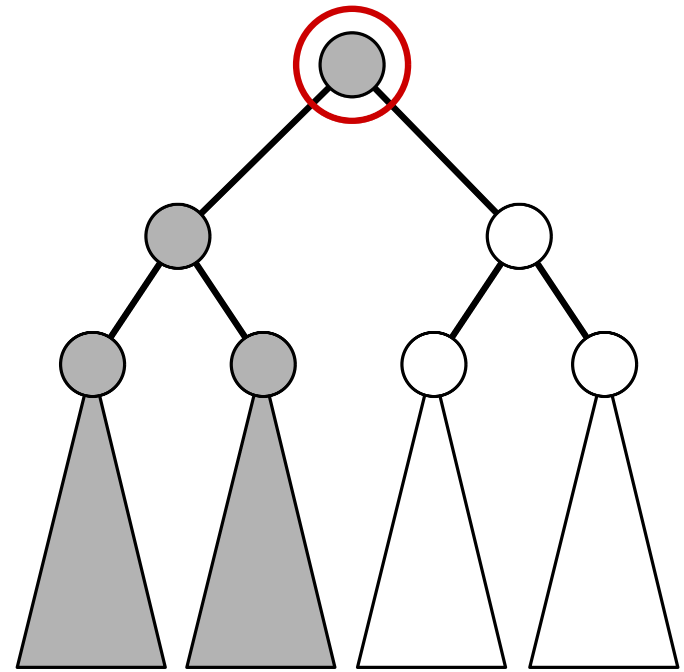
$$Q(n) = 1 + Q(n/2)$$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





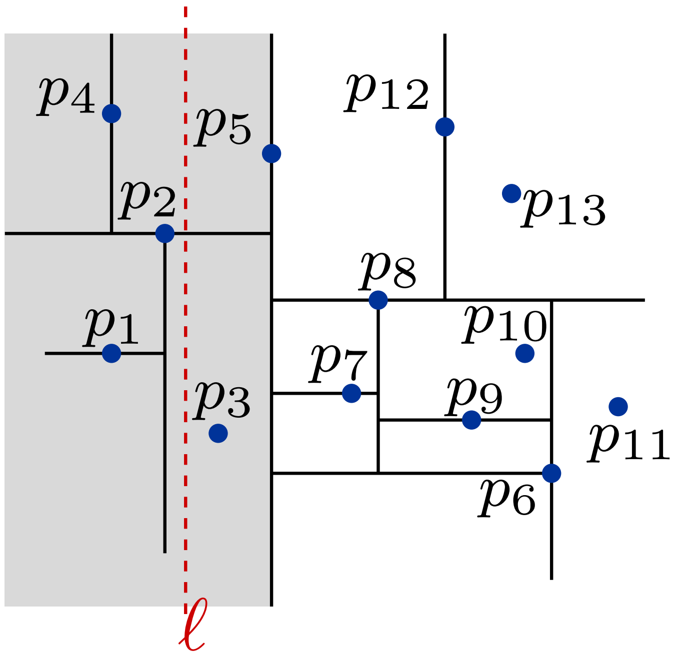
Vermutung:

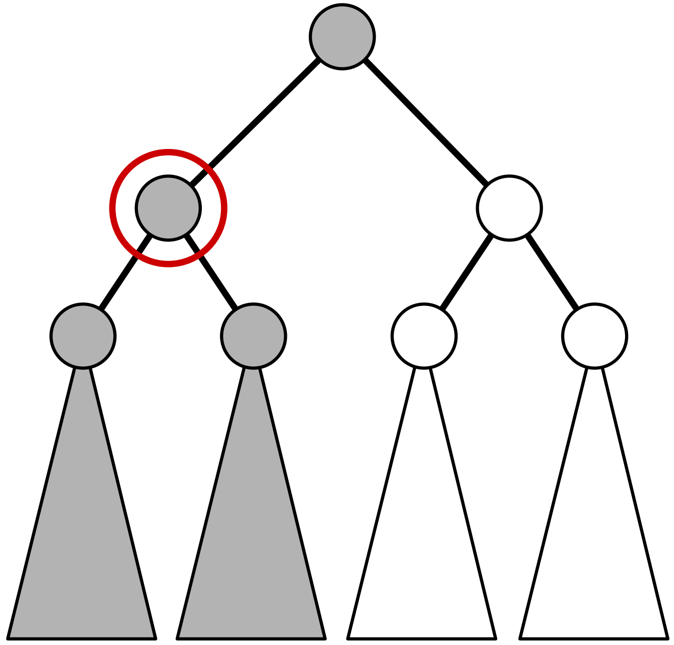
$$Q(n) = 1 + Q(n/2)$$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





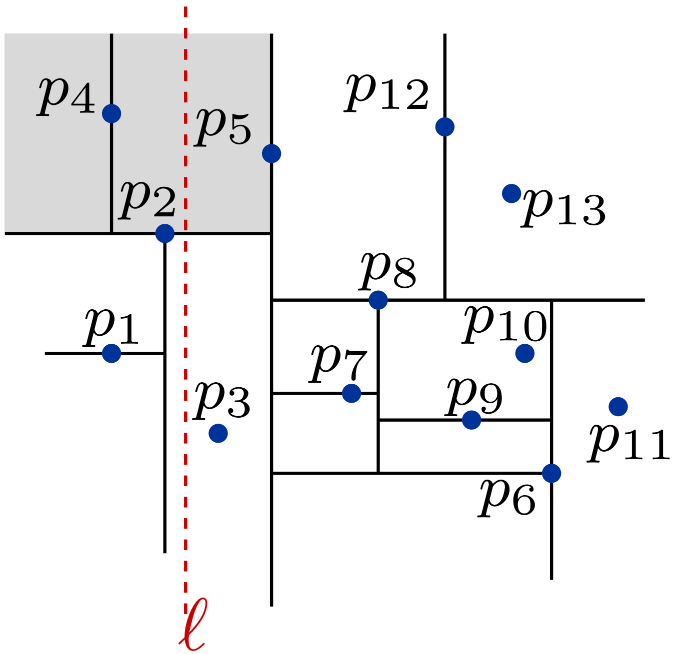
Vermutung:

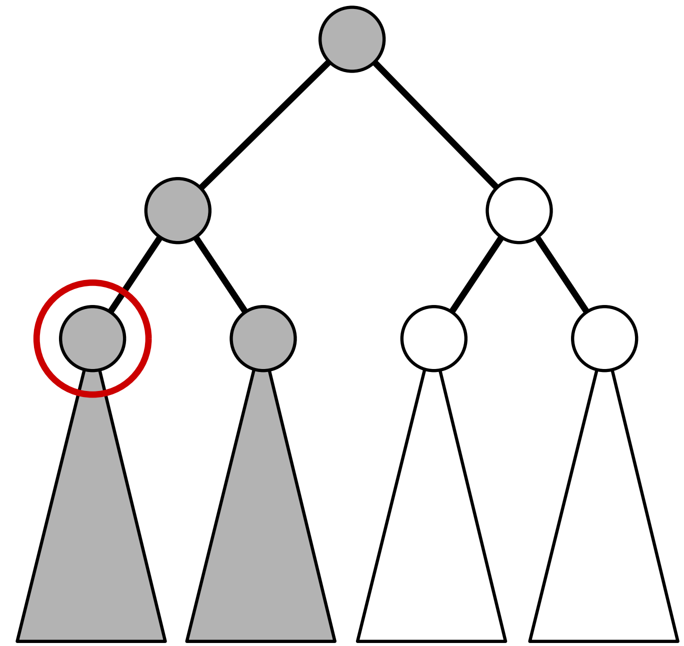
$Q(n) = 1 + Q(n/2)$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





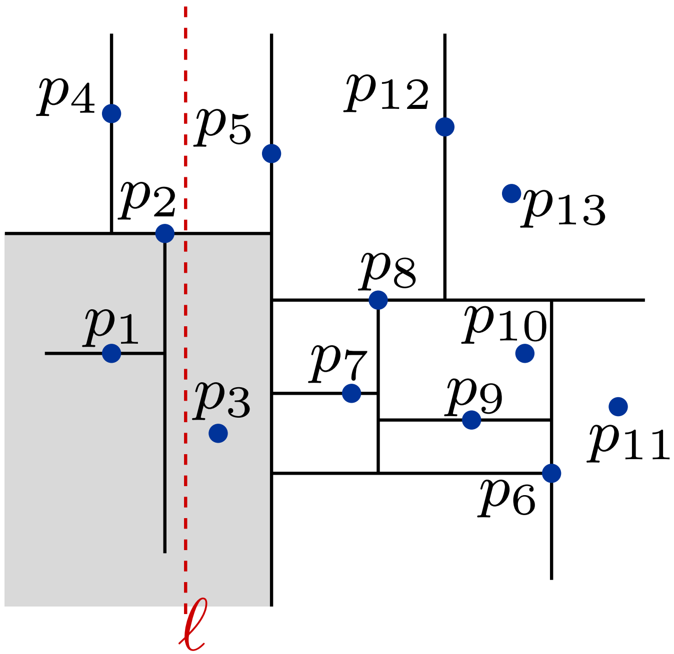
Vermutung:

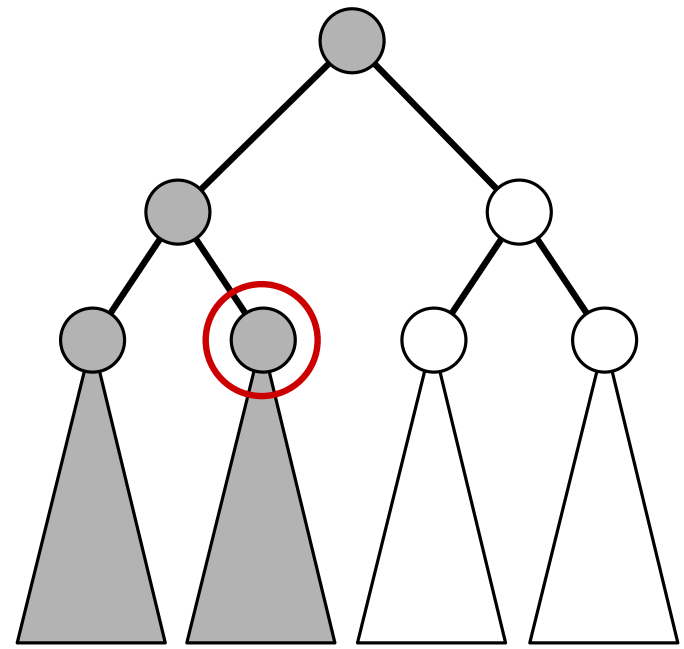
$$Q(n) = 1 + Q(n/2)$$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





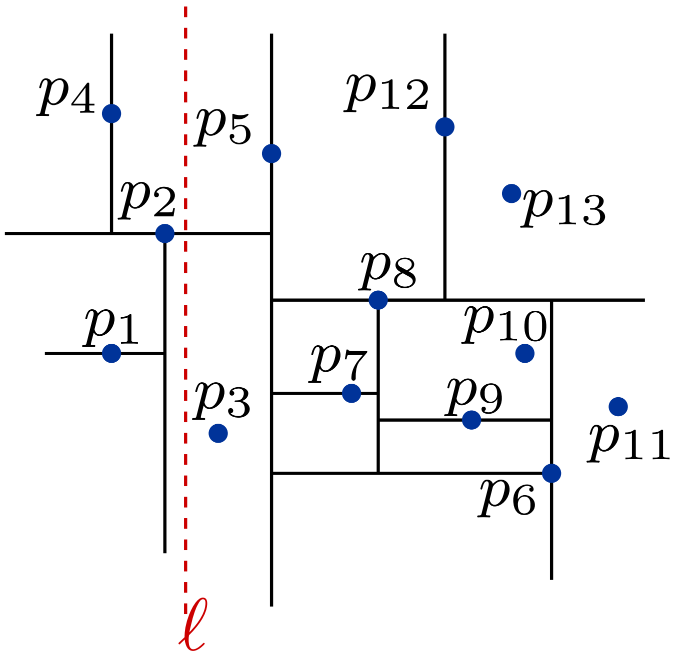
Vermutung:

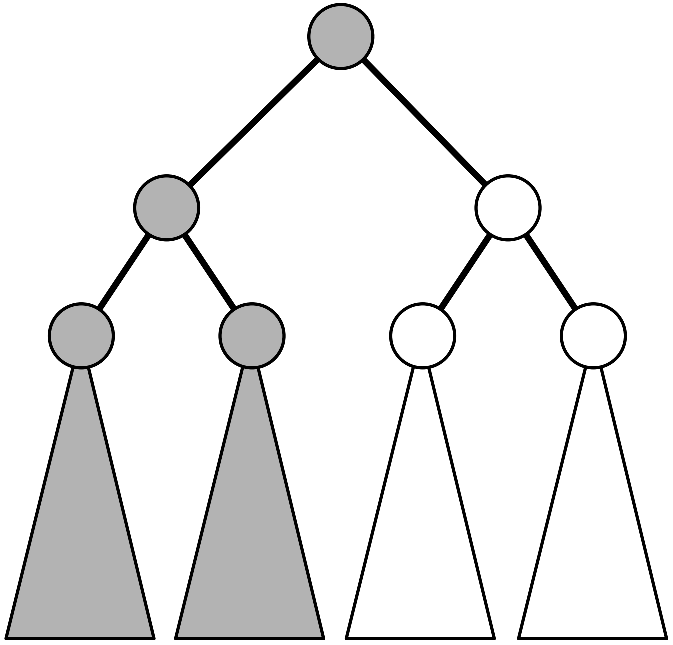
$Q(n) = 1 + Q(n/2)$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





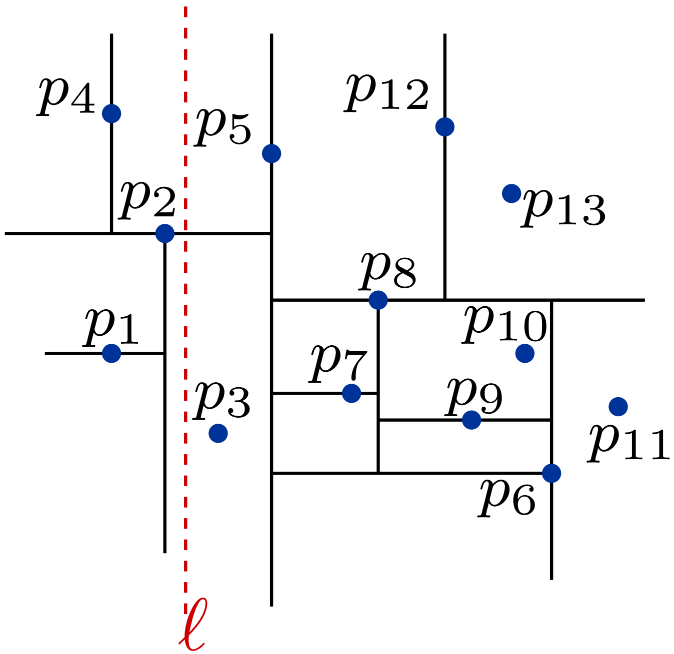
Vermutung:

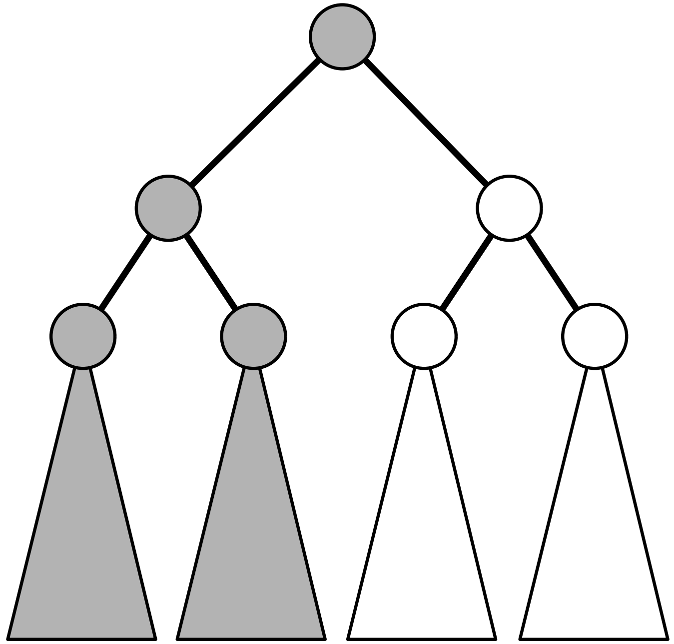
$$Q(n) = 1 + Q(n/2)$$

Problem?

ℓ schneidet beide Kinder des linken Kindes von root

Bereichsabfrage in einem kd-Tree





Vermutung:

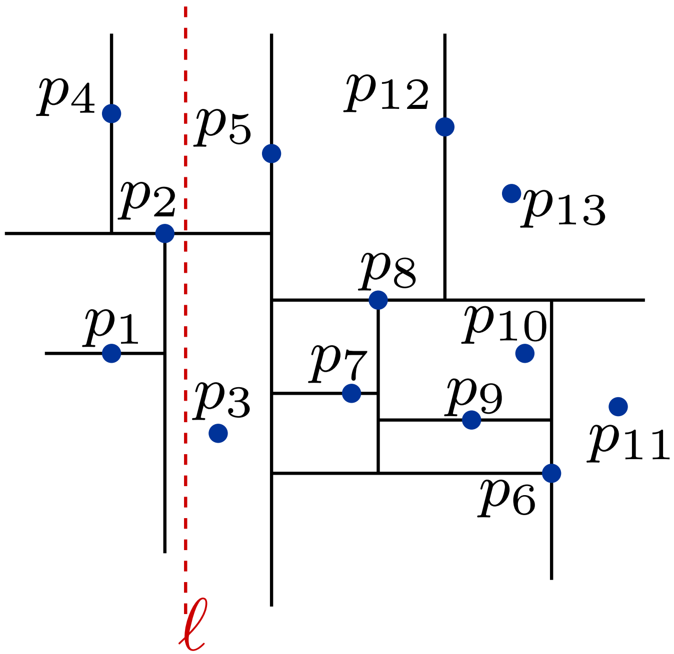
$$Q(n) = 1 + Q(n/2)$$

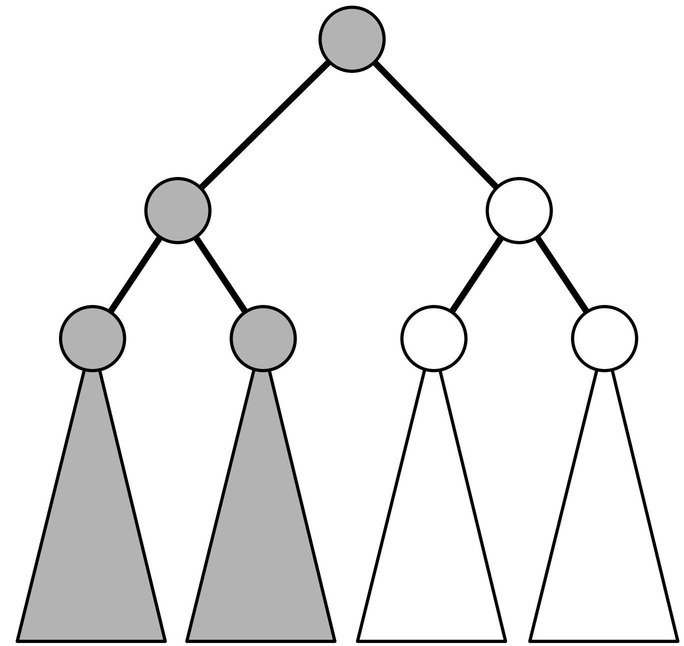
Problem?

ℓ schneidet beide Kinder des linken Kindes von root

$\boxed{\textbf{Gleiche Rekursionssituation}}$

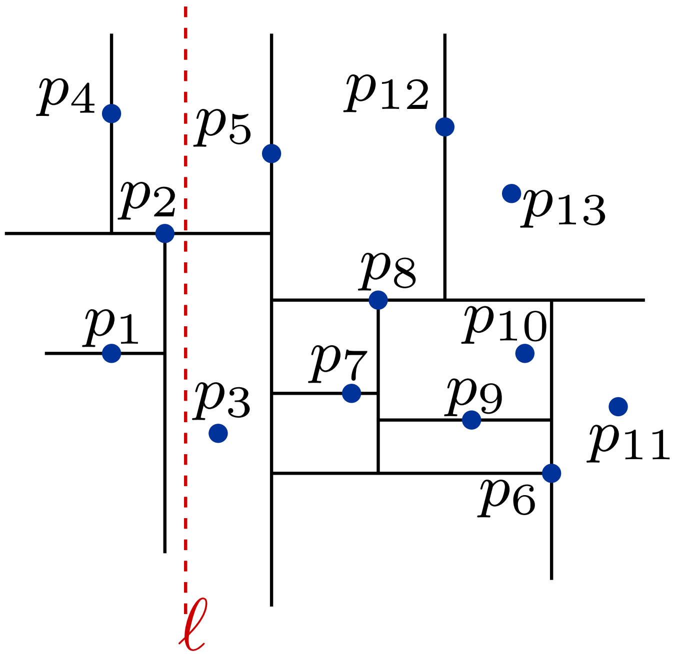
Bereichsabfrage in einem kd-Tree

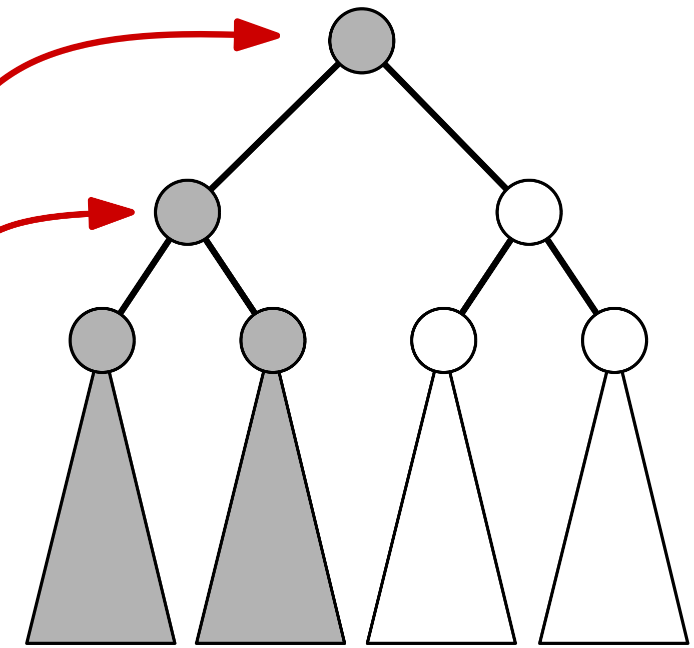




$$Q(n) = \begin{cases} \mathcal{O}(1) & \text{, für } n = 1 \\ 2 + 2Q(n/4) & \text{, für } n > 1 \end{cases}$$

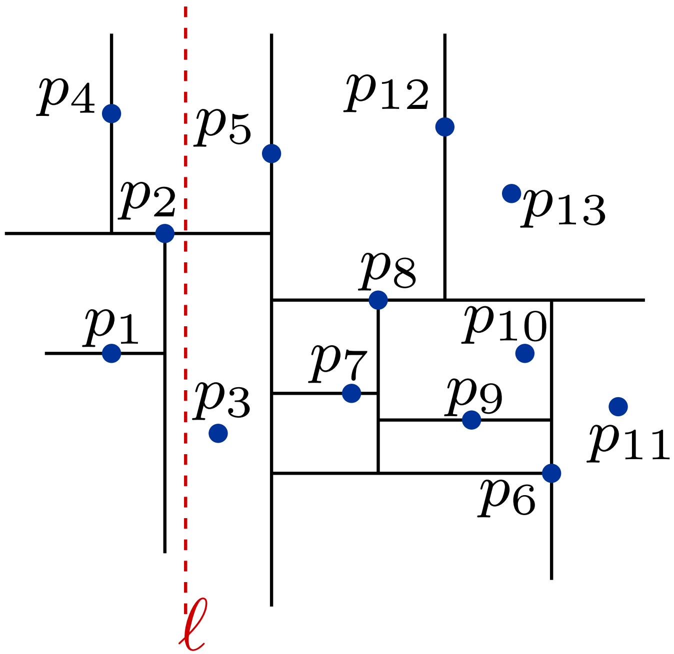
Bereichsabfrage in einem kd-Tree

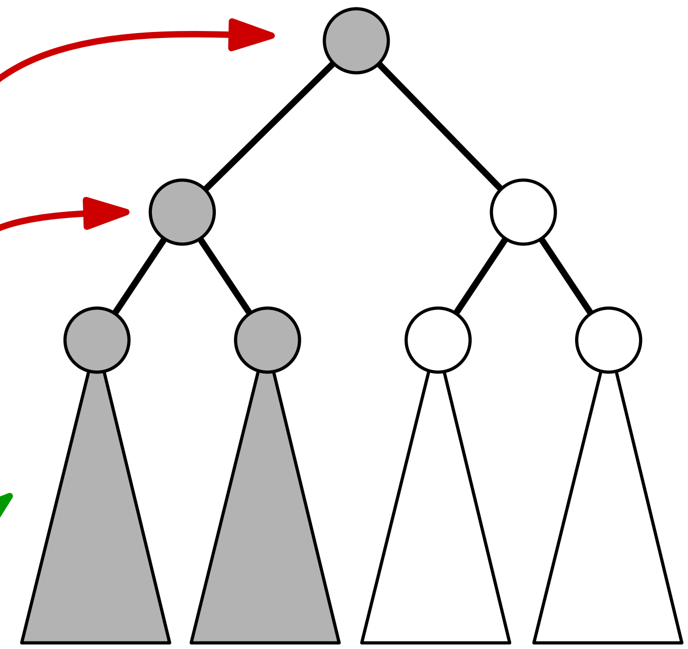




$$Q(n) = \begin{cases} \mathcal{O}(1) & \text{, für } n = 1 \\ 2 + 2Q(n/4) & \text{, für } n > 1 \end{cases}$$

Bereichsabfrage in einem kd-Tree





$$Q(n) = \begin{cases} \mathcal{O}(1) & \text{, für } n = 1 \\ 2 + 2Q(n/4) & \text{, für } n > 1 \end{cases}$$

Aufgabe 1

kd-Trees – w-c Laufzeit

Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & \text{, für } n = 1 \\ 2 + 2Q(n/4) & \text{, für } n > 1 \end{cases}$$

Aufgabe 1

kd-Trees – w-c Laufzeit

Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

b) Rekurrenz auflösen: $Q(n) = \mathcal{O}(\sqrt{n})$.

Aufgabe 1

kd-Trees – w-c Laufzeit

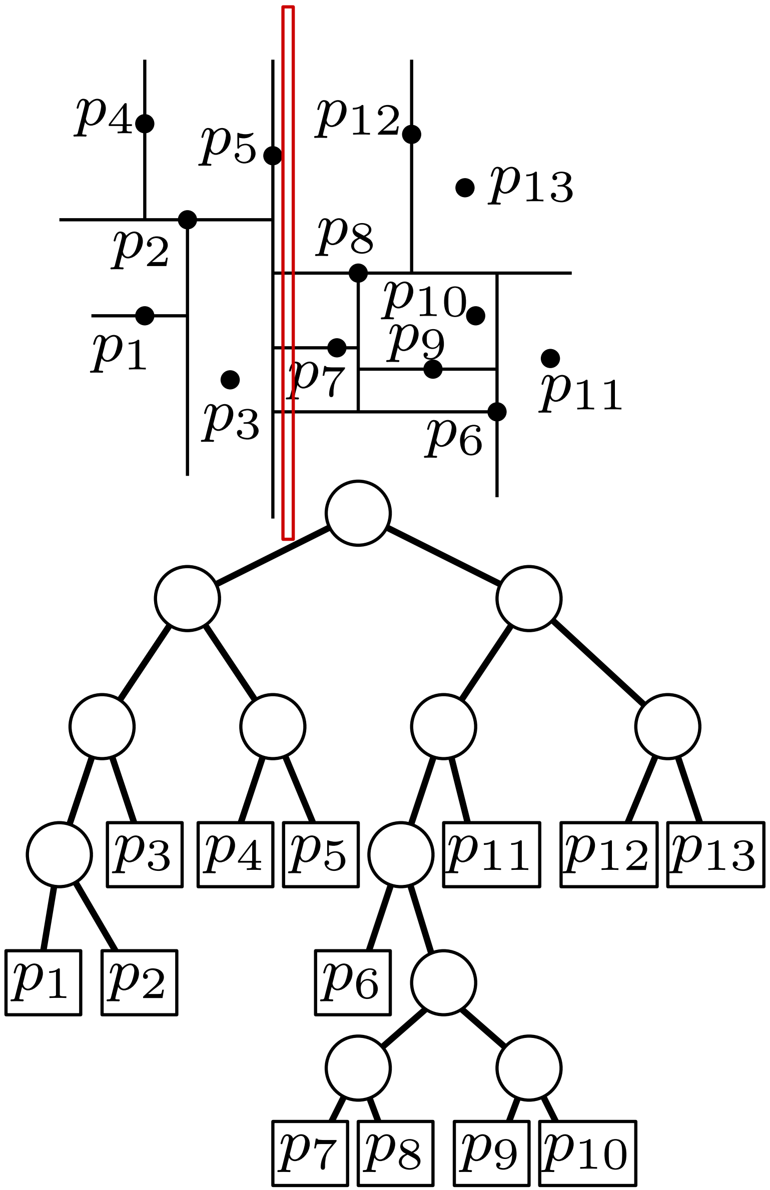
Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

b) Rekurrenz auflösen: $Q(n) = \mathcal{O}(\sqrt{n})$.

c) $\Omega(\sqrt{n})$ untere Schranke für Bereichsanfragen in kd-Trees



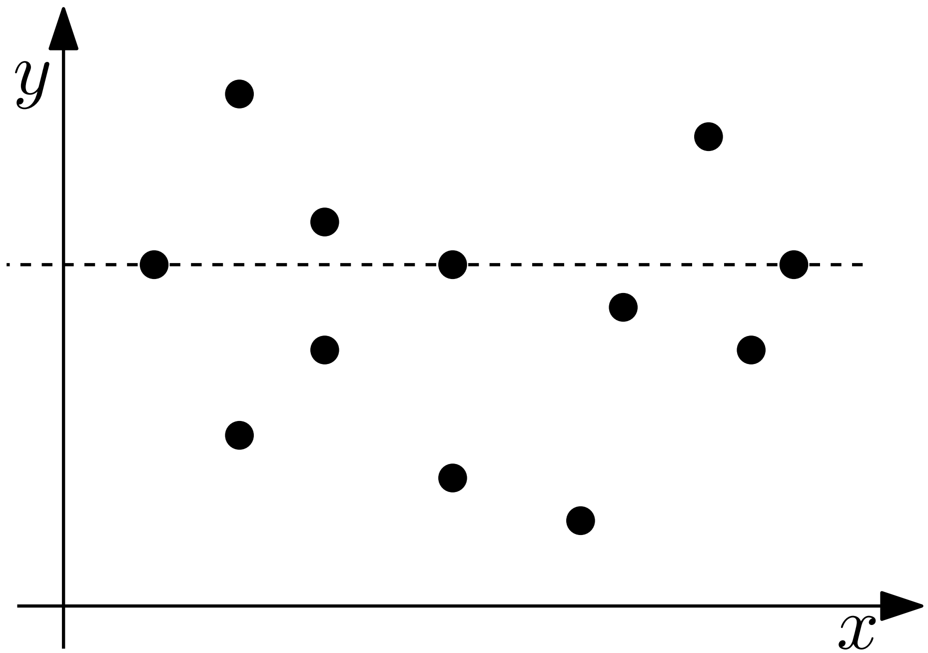
SearchKdTree(v, R)

- **if** v Blatt **then**
  - prüfe Punkt p in v auf $p \in R$
- **else**
  - **if** region(lc(v)) $\subseteq R$ **then**
    - ReportSubtree(lc(v))
  - **else**
    - **if** region(lc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(lc$(v), R$)
  - **if** region(rc(v)) $\subseteq R$ **then**
    - ReportSubtree(rc(v))
  - **else**
    - **if** region(rc(v)) $\cap R \neq \emptyset$ **then**
    - SearchKdTree(rc$(v), R$)

c) $\Omega(\sqrt{n})$ untere Schranke für Bereichsanfragen in kd-Trees

Aufgabe 2

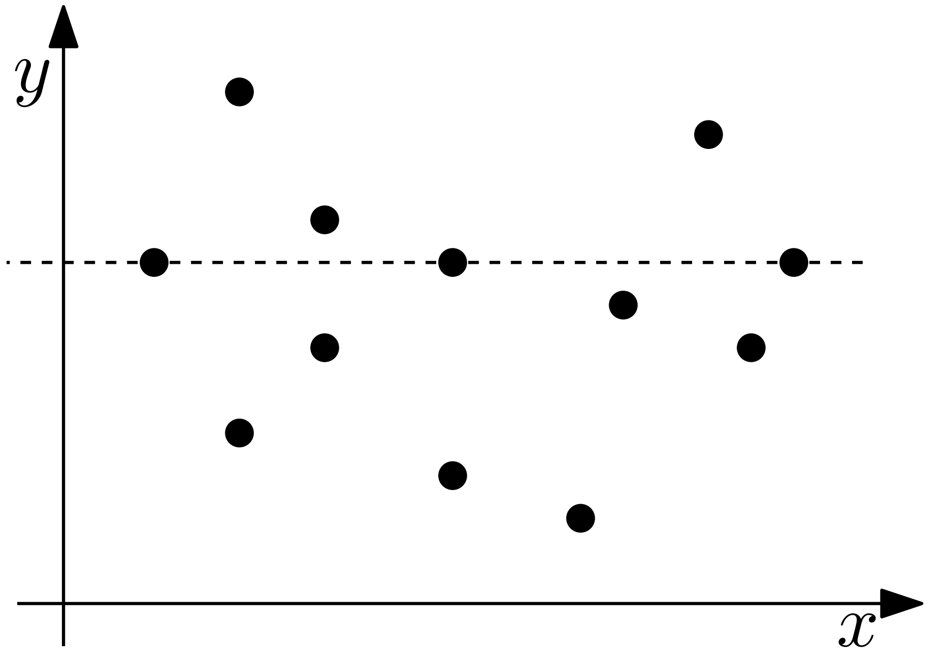
'partial match queries'



Gebe mir alle Punkte mit $y = 7$.

Aufgabe 2

'partial match queries'



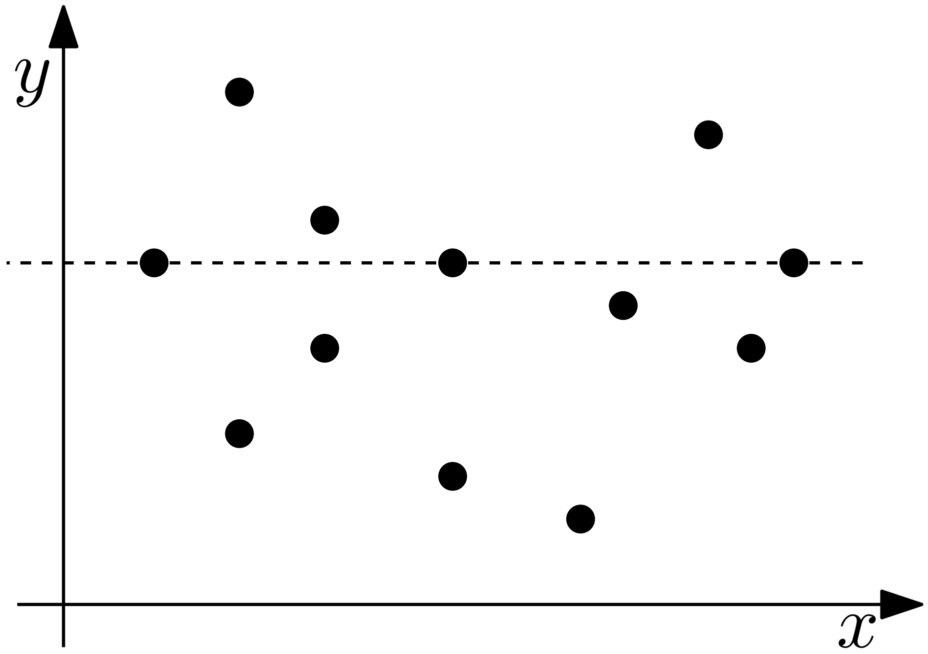
Gebe mir alle Punkte mit $y = 7$.

---

a) Wie kann man kd-Trees dafür nutzen?

Aufgabe 2

'partial match queries'

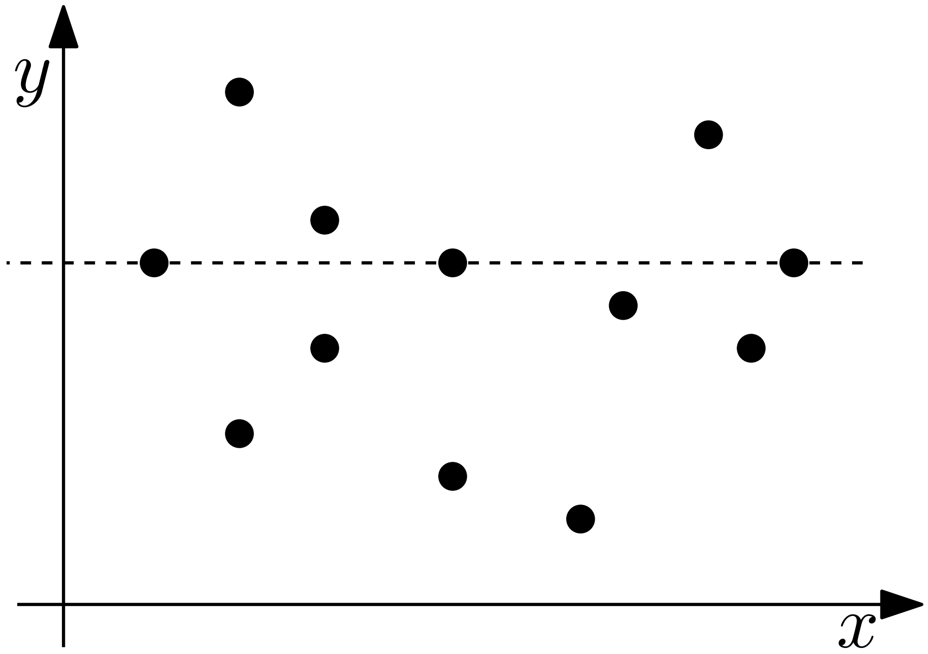


Gebe mir alle Punkte mit $y = 7$.

b) Wie kann man range-Trees dafür nutzen?

Aufgabe 2

'*partial match queries*'



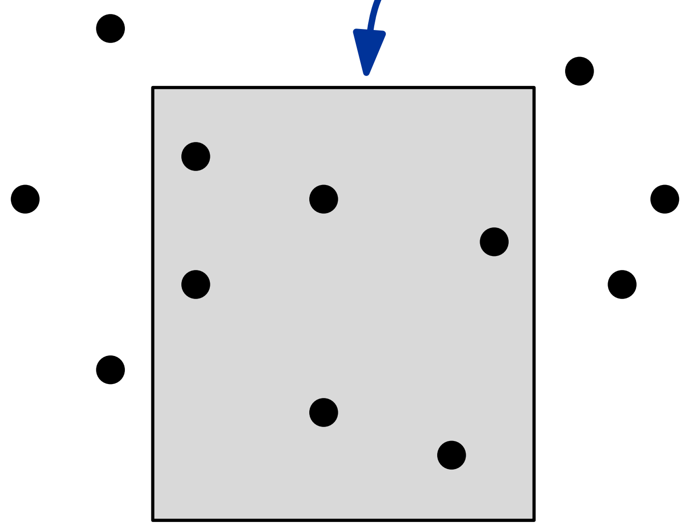
Gebe mir alle Punkte mit $y = 7$.

---

c) Gesucht: Datenstruktur, die das Problem in $\mathcal{O}(\log n + k)$ Zeit und $\mathcal{O}(n)$ Speicher löst.

Aufgabe 3

Wie viele Punkte liegen in diesem Rechteck?

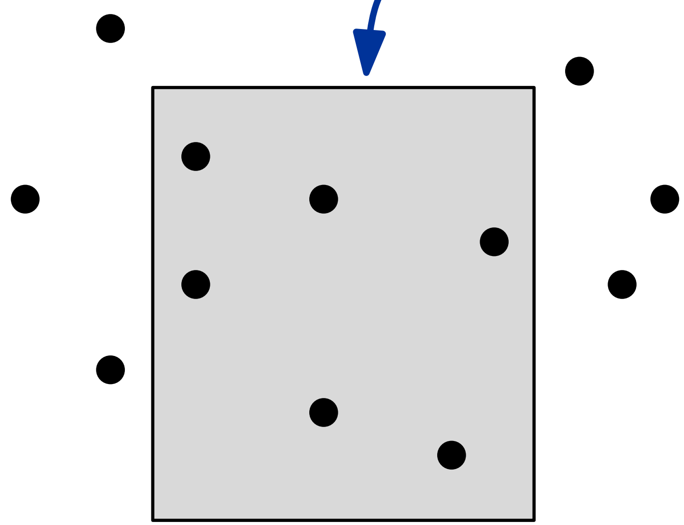


range counting query

Anforderung: Additive Konstante $\mathcal{O}(k)$ in Laufzeit vermeiden

Aufgabe 3

Wie viele Punkte liegen in diesem Rechteck?



range counting query

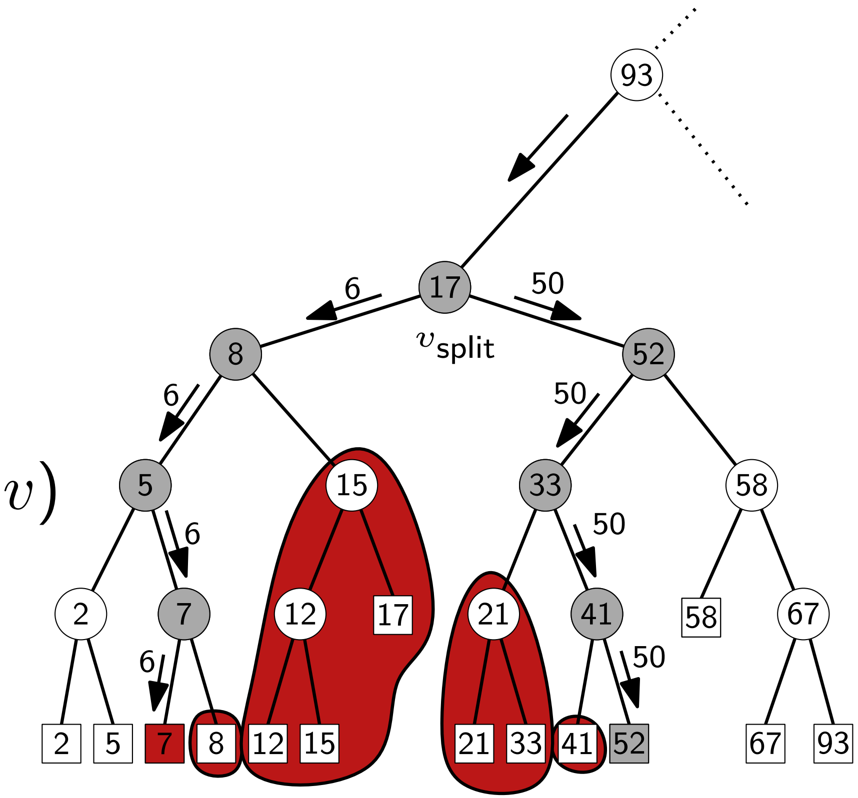
Anforderung: Additive Konstante $\mathcal{O}(k)$ in Laufzeit vermeiden

a) Adaptiere 1-dim Range-Tree um range counting queries in $\mathcal{O}(\log n)$ machbar ist

Aufgabe 3

1dRangeQuery(T, x, x')

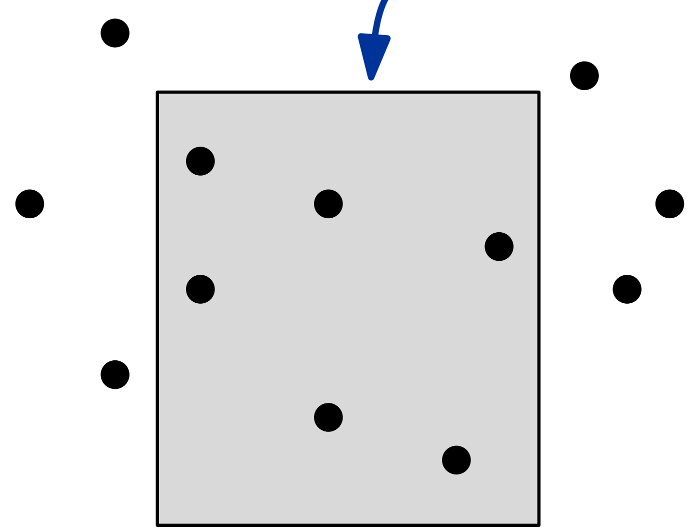
$v_{\text{split}} \leftarrow$ FindSplitNode(T, x, x')
if v_{split} ist Blatt **then** prüfe v_{split}
else
  $v \leftarrow$ lc(v_{split})
  while v kein Blatt **do**
    if $x \leq x_v$ **then**
      ReportSubtree(rc(v)); $v \leftarrow$ lc(v)
    else $v \leftarrow$ rc(v)
  prüfe v
  `// analog für` x' `und rc(`v_{split}`)`



a) Adaptiere 1-dim Range-Tree um range counting queries in $\mathcal{O}(\log n)$ machbar ist

Aufgabe 3

Wie viele Punkte liegen in diesem Rechteck?



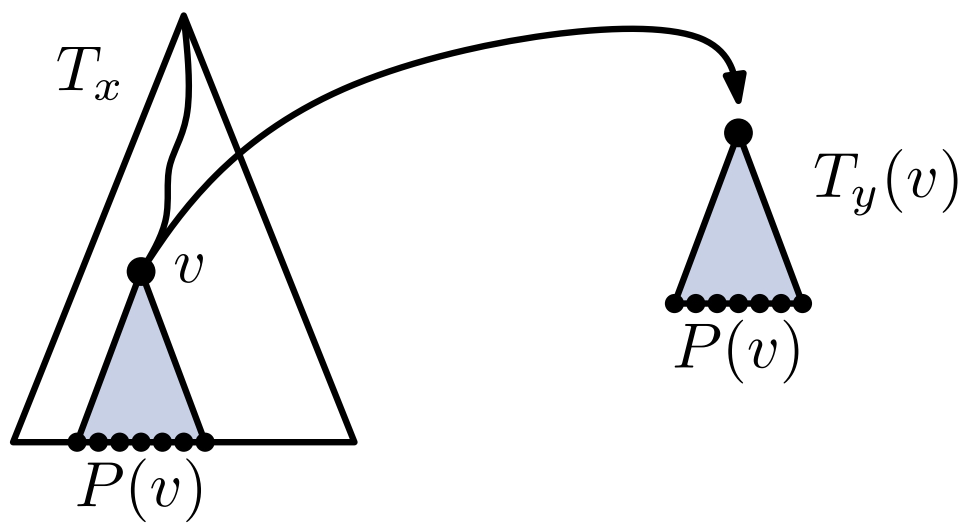
range counting query

Anforderung: Additive Konstante $\mathcal{O}(k)$ in Laufzeit vermeiden

b) Benutze Lösung aus a) um das d-dimensionale Problem zu lösen

Aufgabe 3

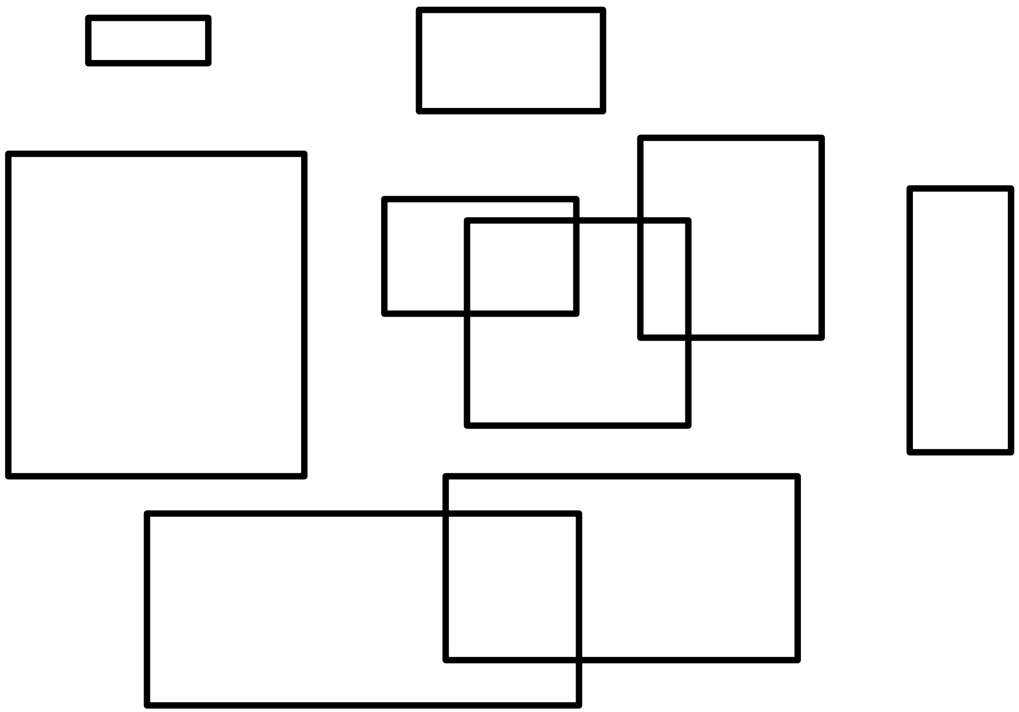
2dim Range-Trees:



Anforderung: Additive Konstante $\mathcal{O}(k)$ in Laufzeit vermeiden

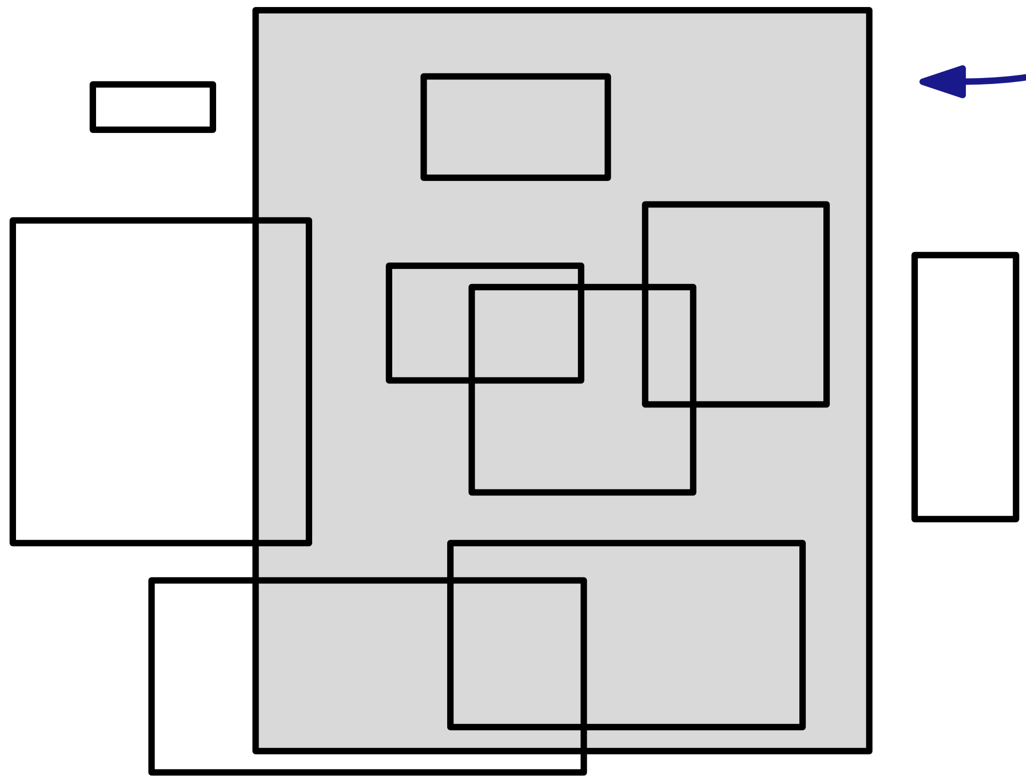
b) Benutze Lösung aus a) um das d-dimensionale Problem zu lösen

Aufgabe 4



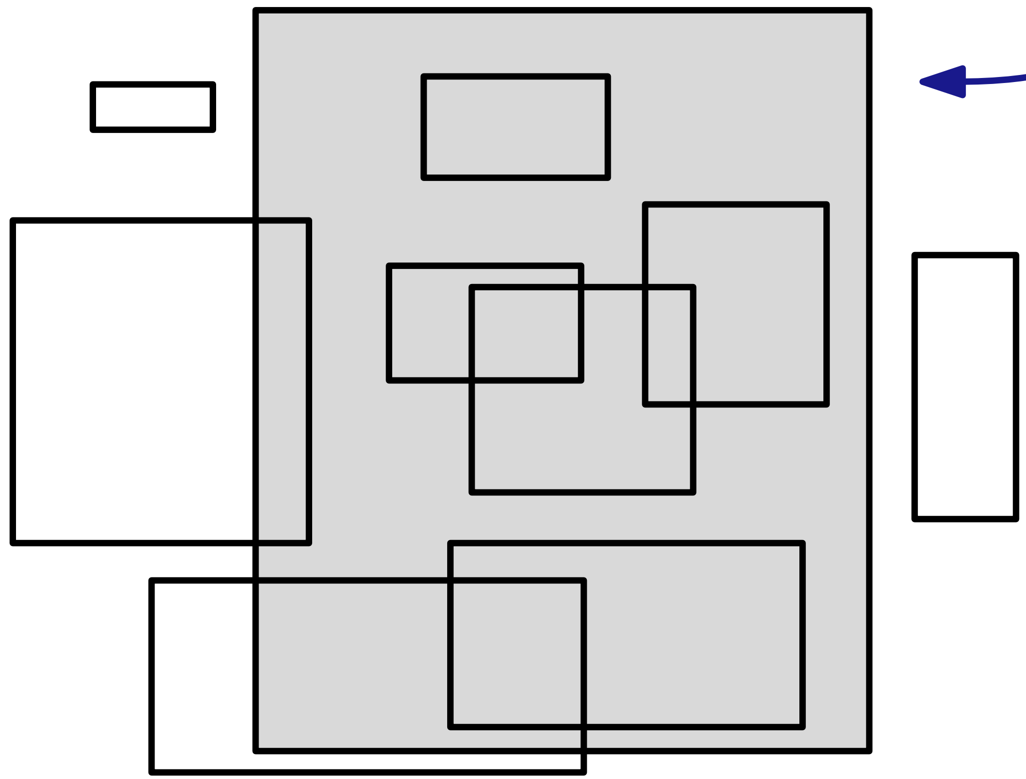
Aufgabe 4

Welche Rechtecke liegen vollständig in diesem Rechteck?



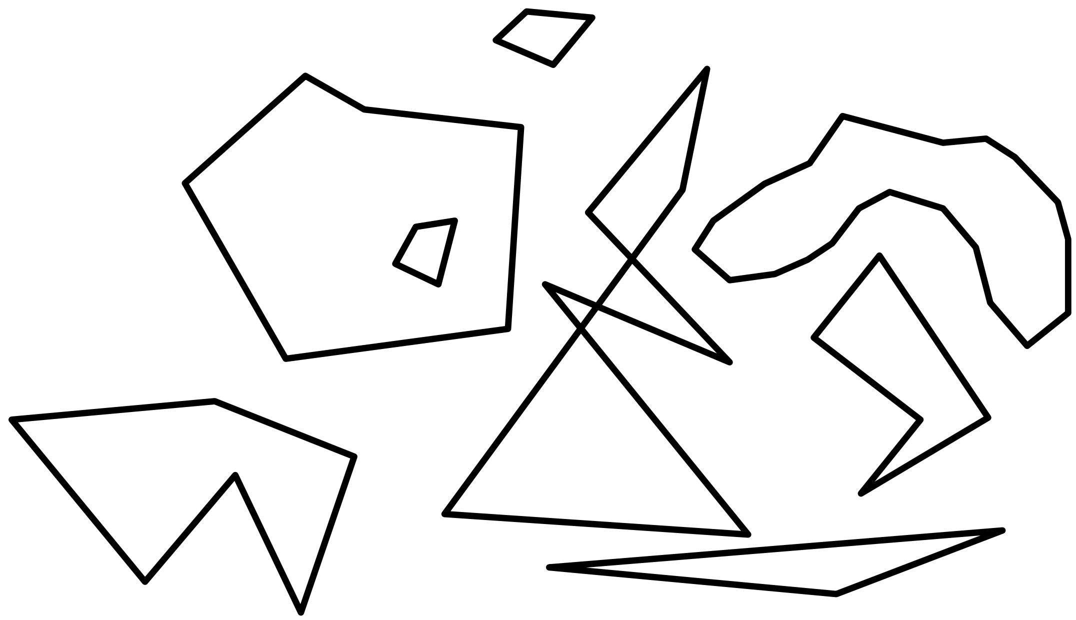
Aufgabe 4

Welche Rechtecke liegen vollständig in diesem Rechteck?



Datenstruktur mit $\mathcal{O}(n \log^3 n)$ Speicher und $\mathcal{O}(\log^4 n + k)$ Anfragezeit.

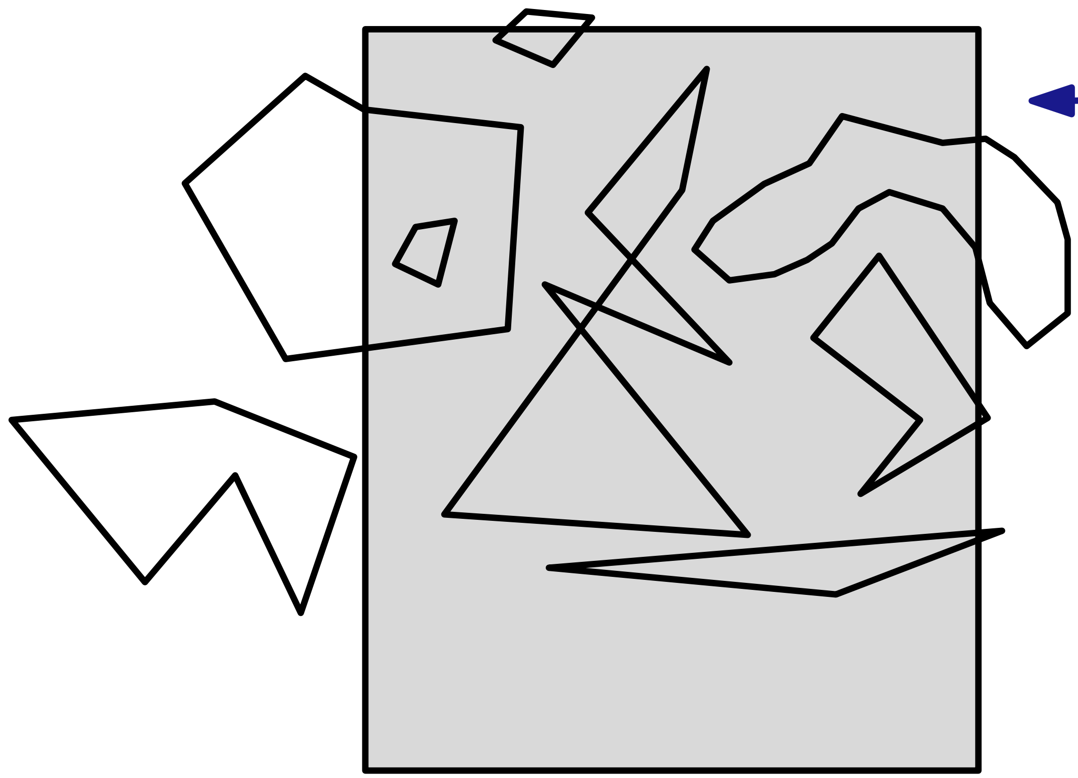
Aufgabe 4



Datenstruktur mit $\mathcal{O}(n \log^3 n)$ Speicher und $\mathcal{O}(\log^4 n + k)$ Anfragezeit.

Aufgabe 4

Welche Polygone liegen vollständig in diesem Rechteck?



Datenstruktur mit $\mathcal{O}(n \log^3 n)$ Speicher und $\mathcal{O}(\log^4 n + k)$ Anfragezeit.

Übersicht

Übungsblatt 4

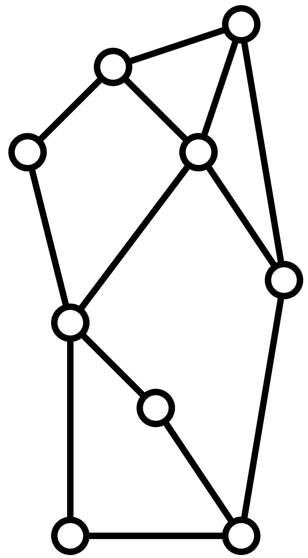
Nachtrag zu Übungsblatt 3

Übungsblatt 5

Werbung

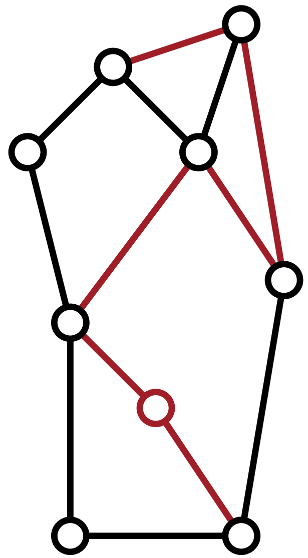
Visualisierung dynamischer Netzwerke

dynamisches Netzwerk



Visualisierung dynamischer Netzwerke

dynamisches Netzwerk



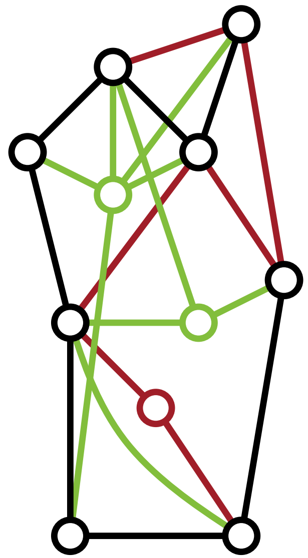
- alte Teile entfallen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Visualisierung dynamischer Netzwerke

Karlsruher Institut für Technologie

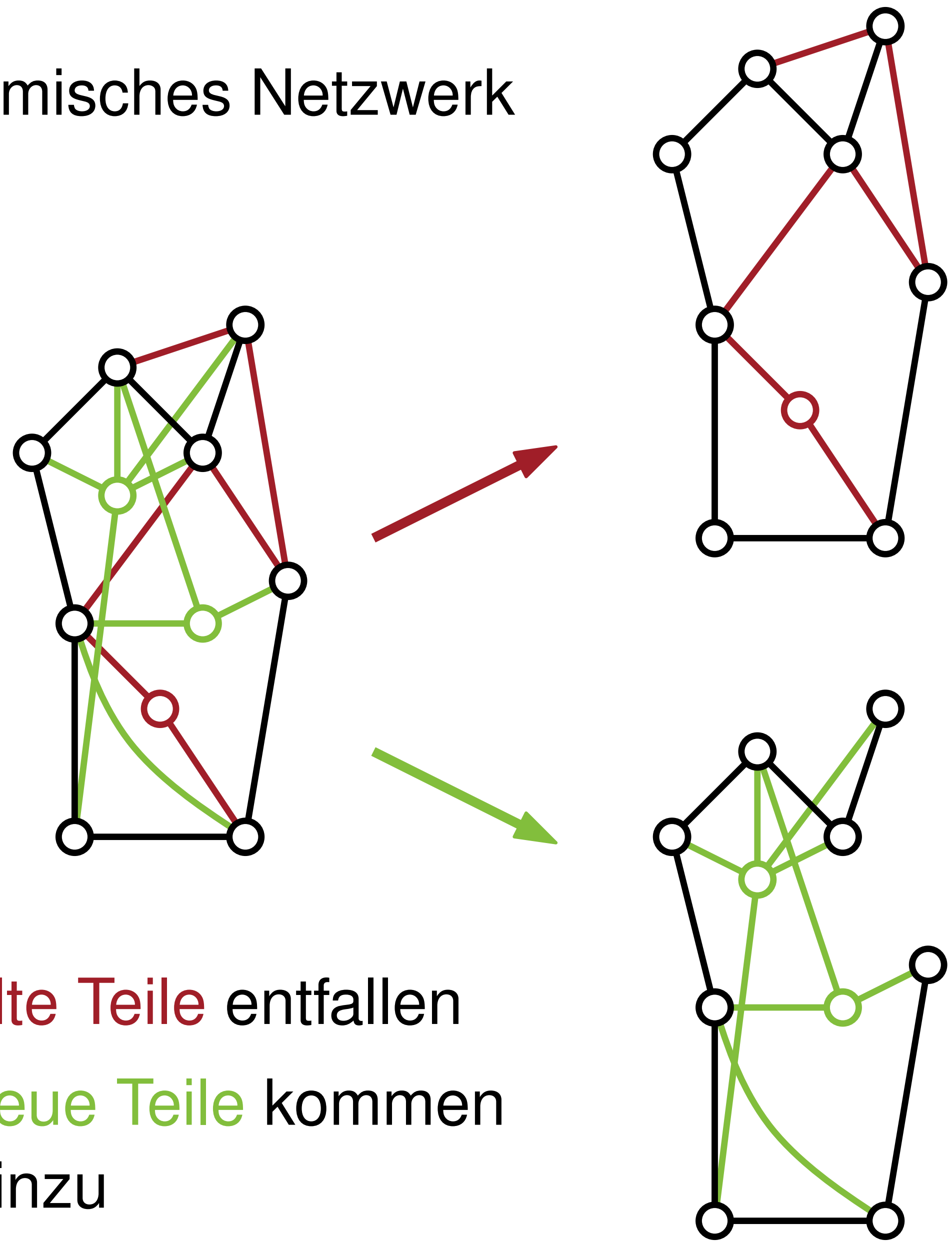
dynamisches Netzwerk



- alte Teile entfallen
- neue Teile kommen hinzu

Visualisierung dynamischer Netzwerke

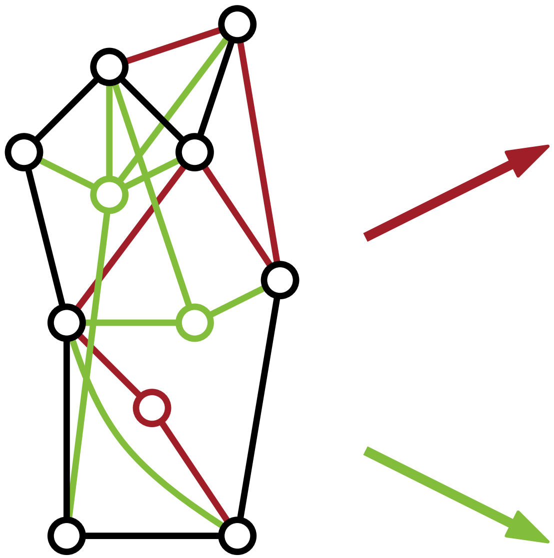
dynamisches Netzwerk

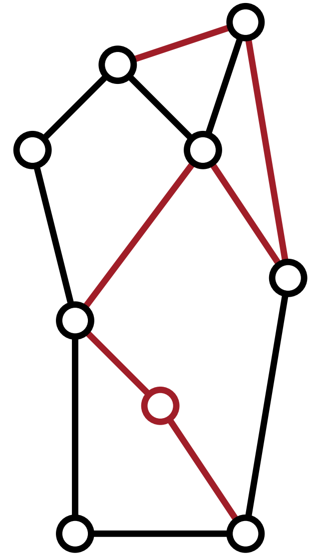


- alte Teile entfallen
- neue Teile kommen hinzu

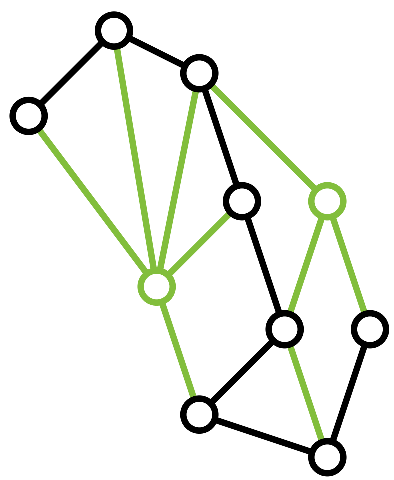
Visualisierung dynamischer Netzwerke

dynamisches Netzwerk





individuelle Zeichnungen

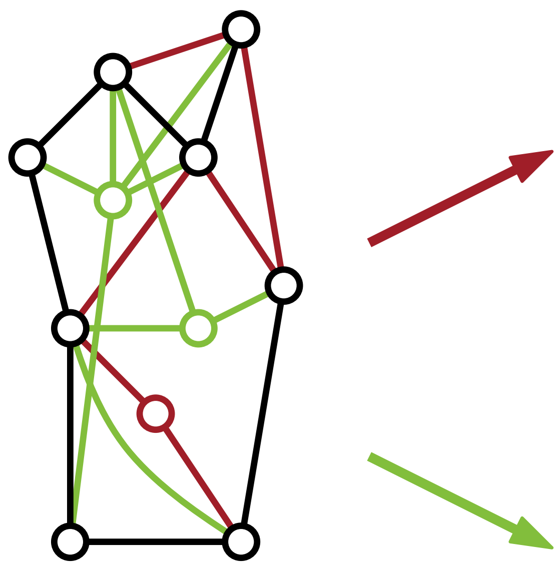


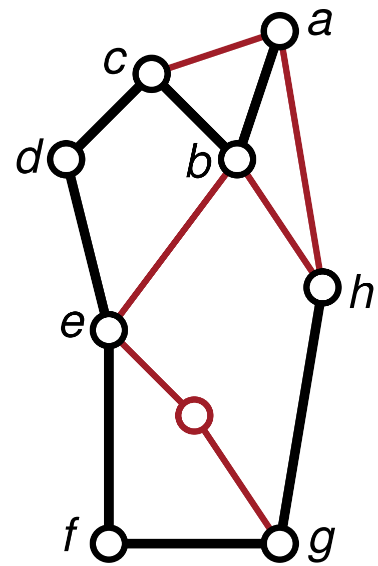
- alte Teile entfallen
- neue Teile kommen hinzu

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

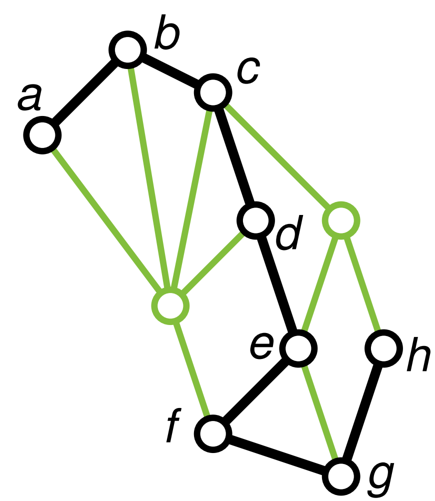
Visualisierung dynamischer Netzwerke

dynamisches Netzwerk





individuelle Zeichnungen

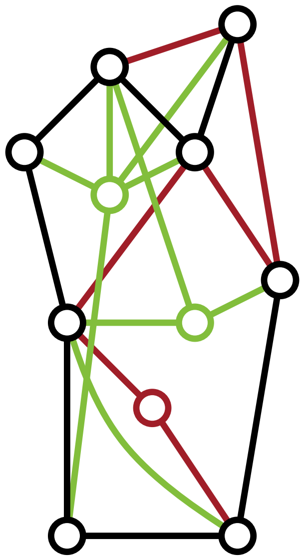


- alte Teile entfallen
- neue Teile kommen hinzu

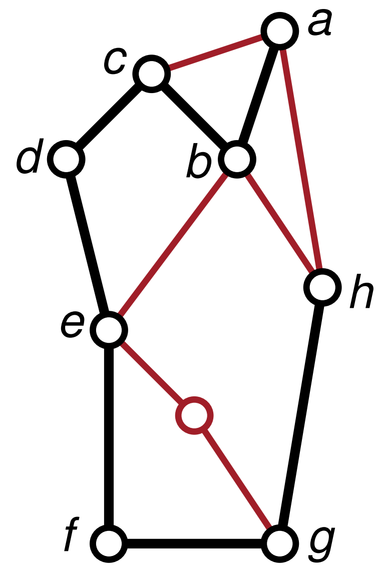
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Visualisierung dynamischer Netzwerke

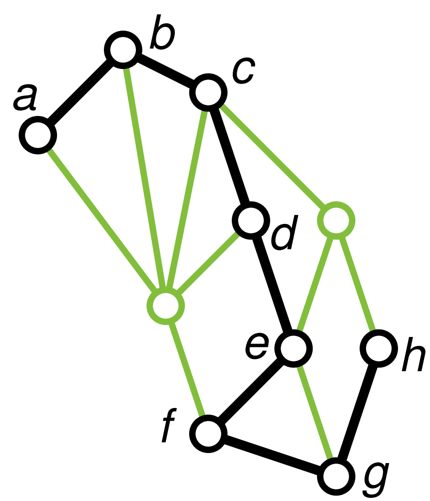
dynamisches Netzwerk



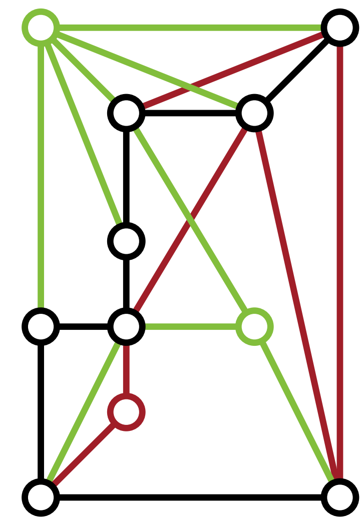
- alte Teile entfallen
- neue Teile kommen hinzu



individuelle Zeichnungen



simultane Zeichnung



Kompromiss zwischen

- guter Lesbarkeit und
- Ähnlichkeit der Zeichnungen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Hiwi gesucht

Aufgabe

Algorithmen zur Visualisierung dynamischer Graphen sollen

- entwickelt,
- implementiert,
- und evaluiert werden.

Hiwi gesucht

Aufgabe

Algorithmen zur Visualisierung dynamischer Graphen sollen

- entwickelt,
- implementiert,
- und evaluiert werden.

Voraussetzungen

- Kenntnisse in C++ oder Java
- Interesse an algorithmischen Fragestellungen

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Hiwi gesucht

Aufgabe

Algorithmen zur Visualisierung dynamischer Graphen sollen

- entwickelt,
- implementiert,
- und evaluiert werden.

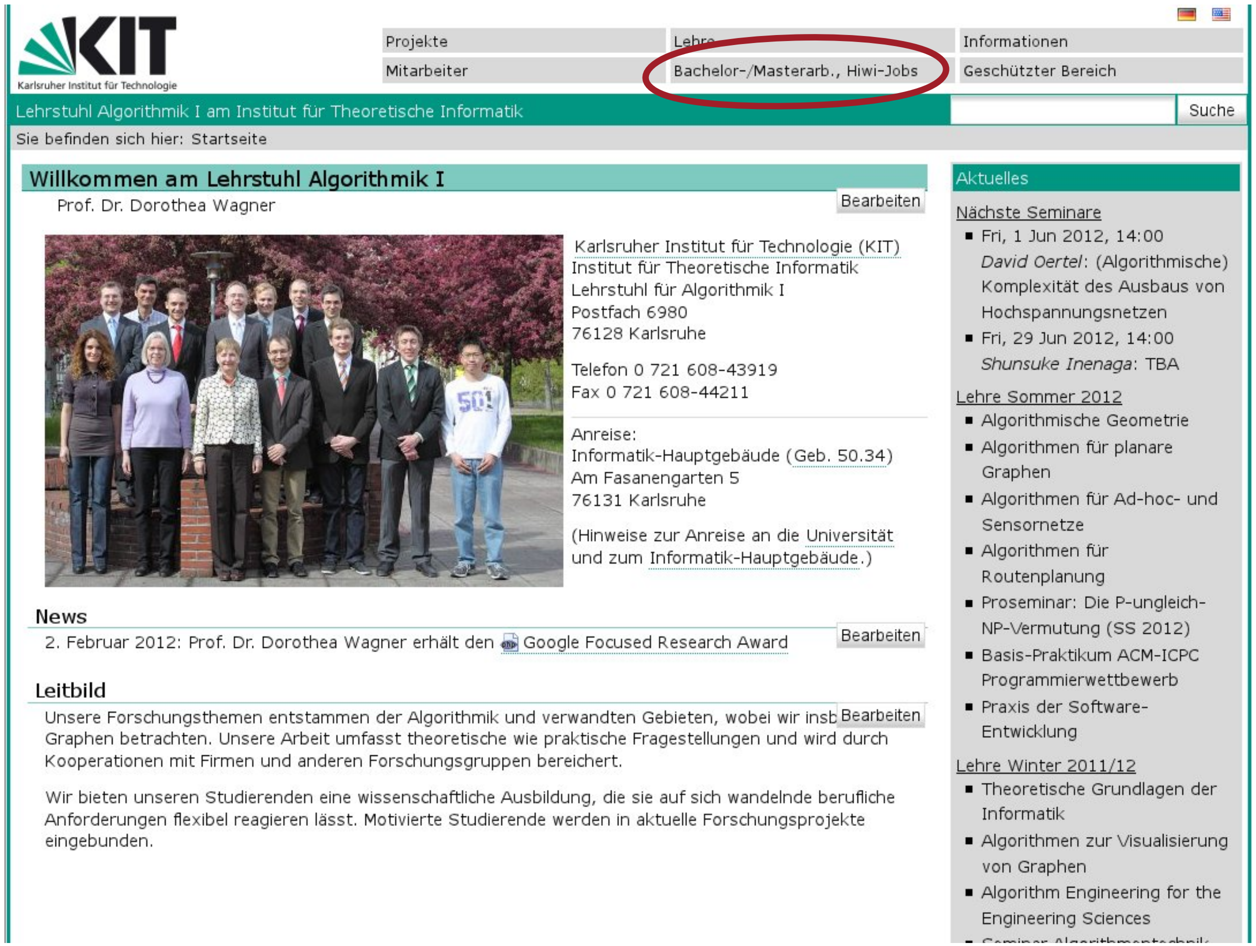
Voraussetzungen

- Kenntnisse in C++ oder Java
- Interesse an algorithmischen Fragestellungen

Wie? Wo? Was?

- Thomas Bläsius · Raum 316 · `thomas.blaesius@kit.edu`

Hiwi gesucht



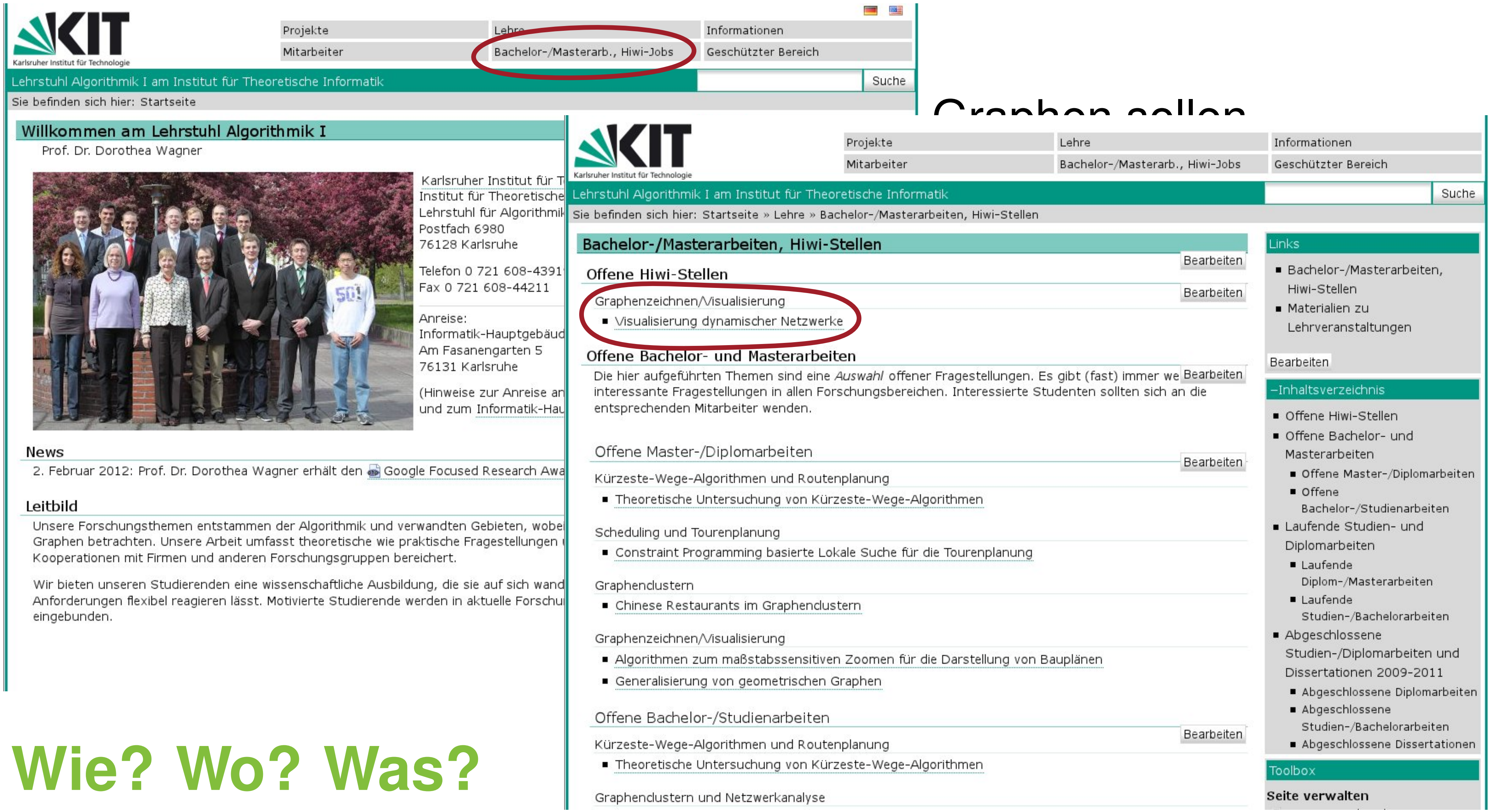
Graphen sollen

ngen

Wie? Wo? Was?

- Thomas Bläsius · Raum 316 · `thomas.blaesius@kit.edu`

Hiwi gesucht



Wie? Wo? Was?

- Thomas Bläsius · Raum 316 · `thomas.blaesius@kit.edu`

Das war's!

Nächster Termin:

Donnertag, 31.05, 10:15 Uhr

Raum 131, Gebäude 50.34