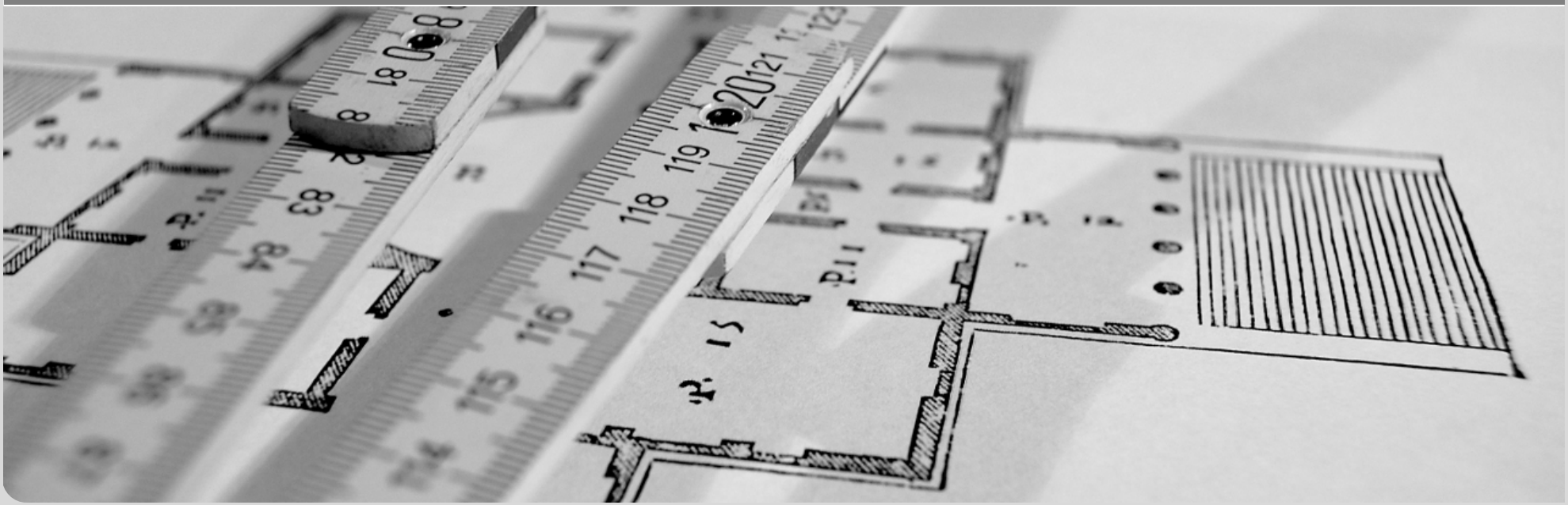
Karlsruher Institut für Technologie

Algorithmische Geometrie

ICPC Praktikum 2012

Huyen Chau Nguyen, Matthias Holoch | 10. Juli 2012

INSTITUT FÜR THEORETISCHE INFORMATIK



Outline/Gliederung

1. Konvexe Hülle
   - Einleitung
   - Gift Wrapping
   - Graham Scan
2. Bereichssuchen
   - 1D Bereichssuche
   - kd-Baum
3. Koordinatenkompression
   - Was ist das?
   - Bei Flächenberechnungen
4. Sweep
   - Line Segment Intersection
   - Allgemeine Eigenschaften
   - Weitere Algorithmen

Konvexe Hülle

Definition

Die **konvexe Hülle** in einem d-dimensionalen Raum mit n-Punkten ist die kleinste konvexe Menge, die alle n Punkte enthält

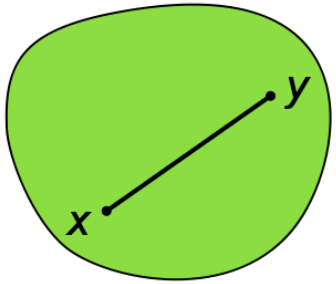
Konvexe Mengen sind geometrische Figuren, die alle Verbindungsstrecken zwischen paarweise verschiedenen Punkten enthalten.

Konvexe Hülle

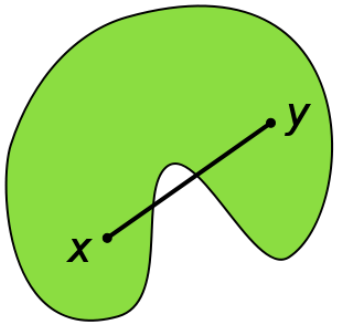
Definition

Die **konvexe Hülle** in einem d-dimensionalen Raum mit n-Punkten ist die kleinste konvexe Menge, die alle n Punkte enthält

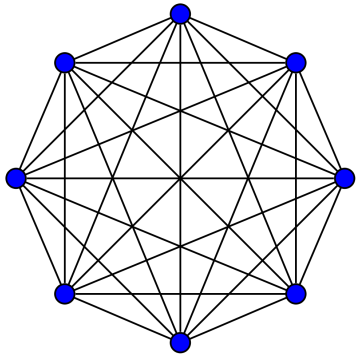
Konvexe Mengen sind geometrische Figuren, die alle Verbindungsstrecken zwischen paarweise verschiedenen Punkten enthalten.



(d) konvex

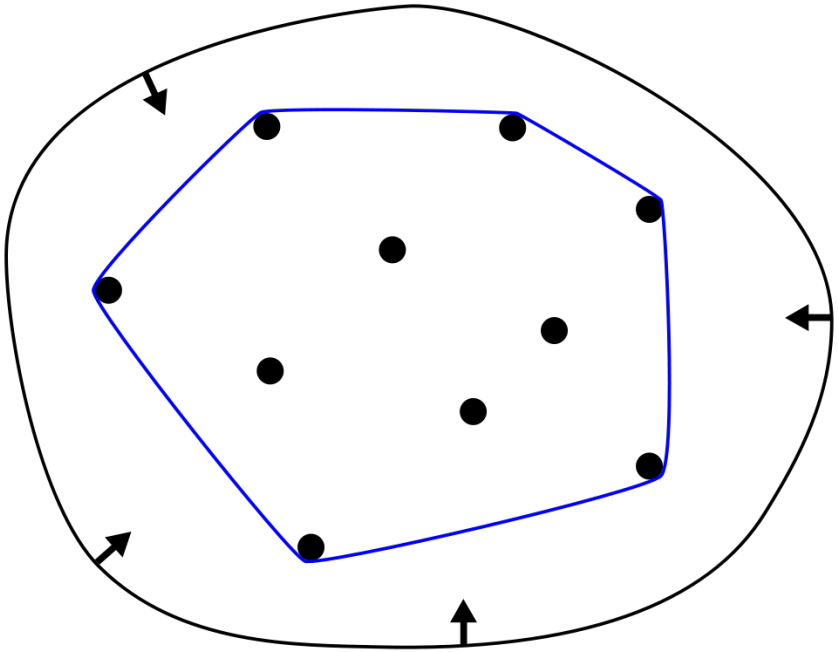


(e) nicht konvex



(f) vollständiger Graph

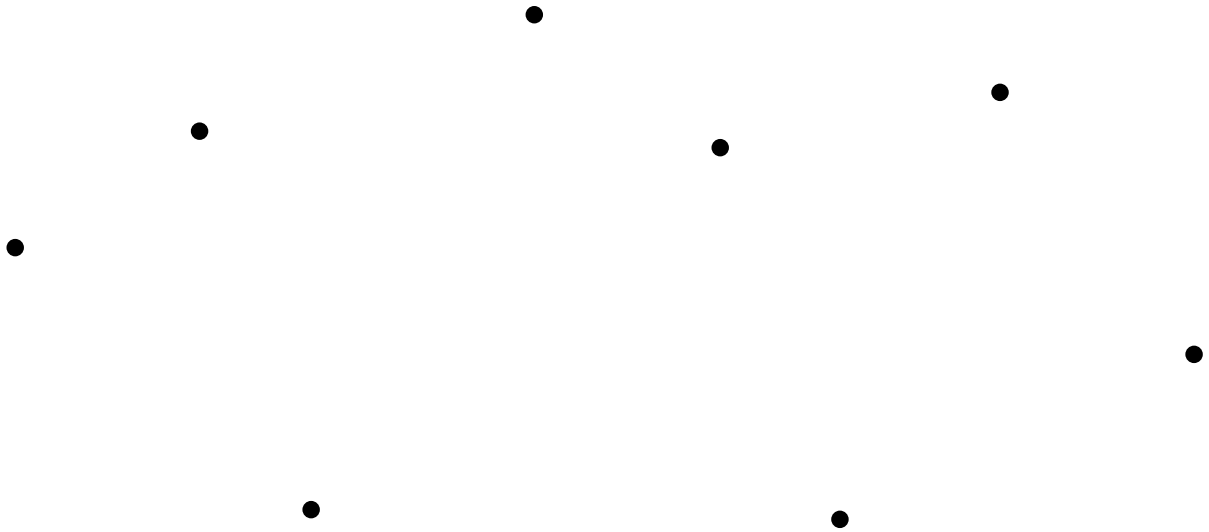
Konvexe Hülle



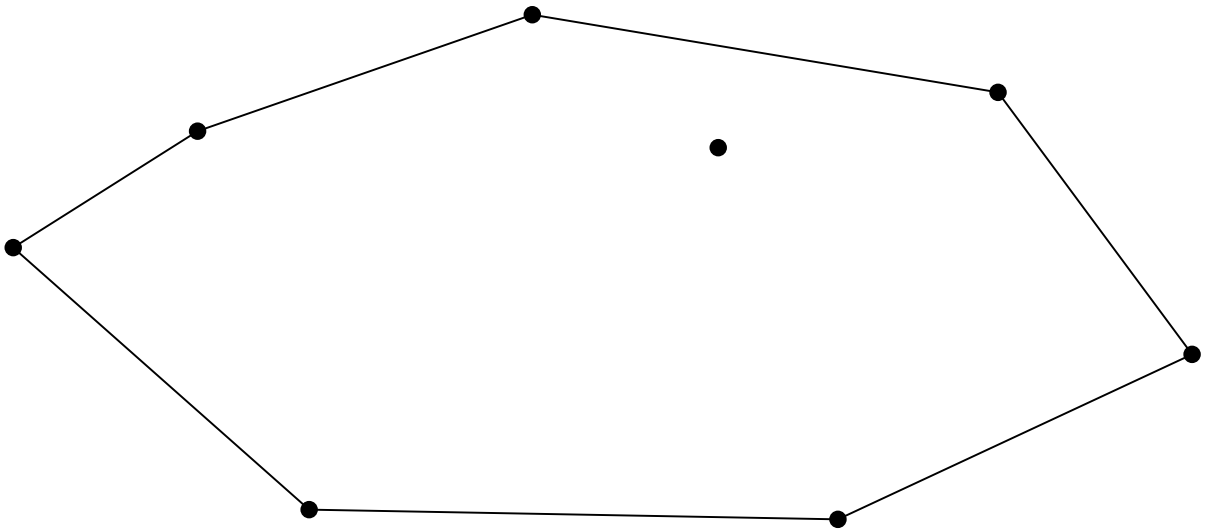
Motivation

- Kollisionsberechnung
- konvexe Optimierung

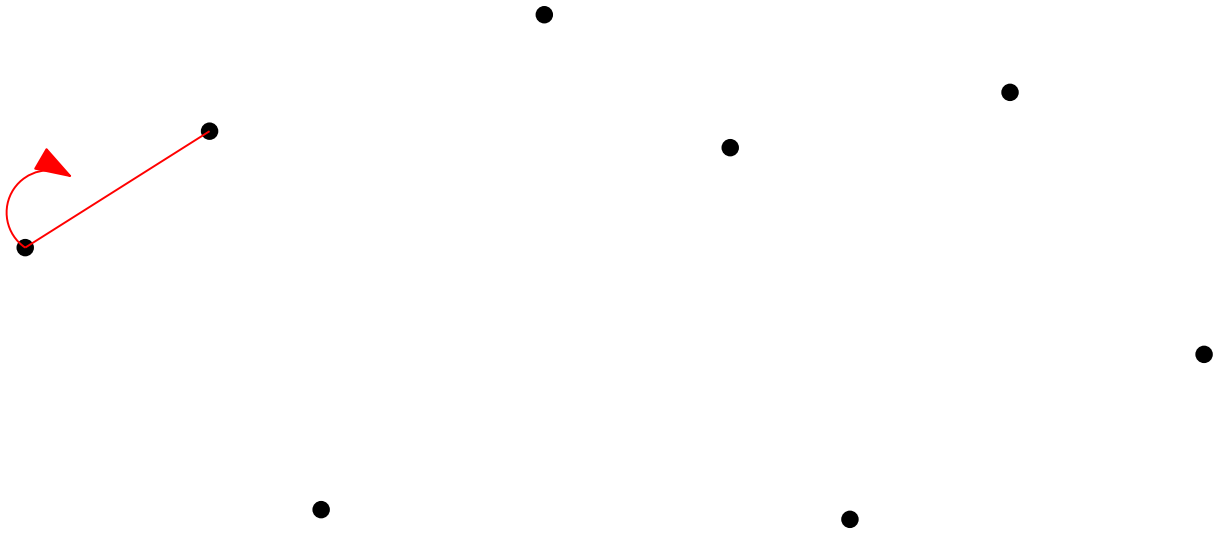
Gift Wrapping



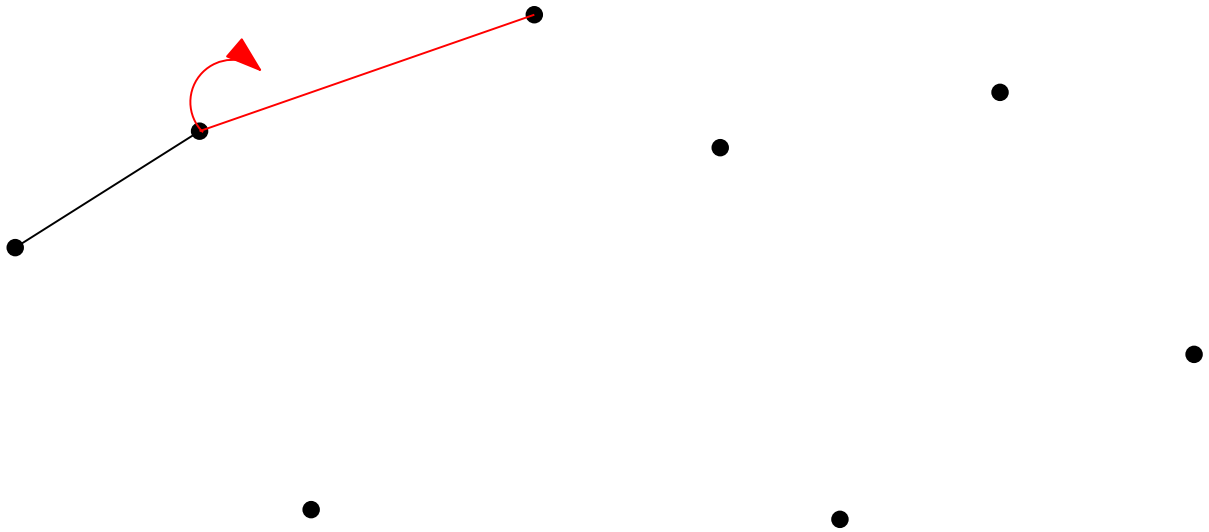
Gift Wrapping



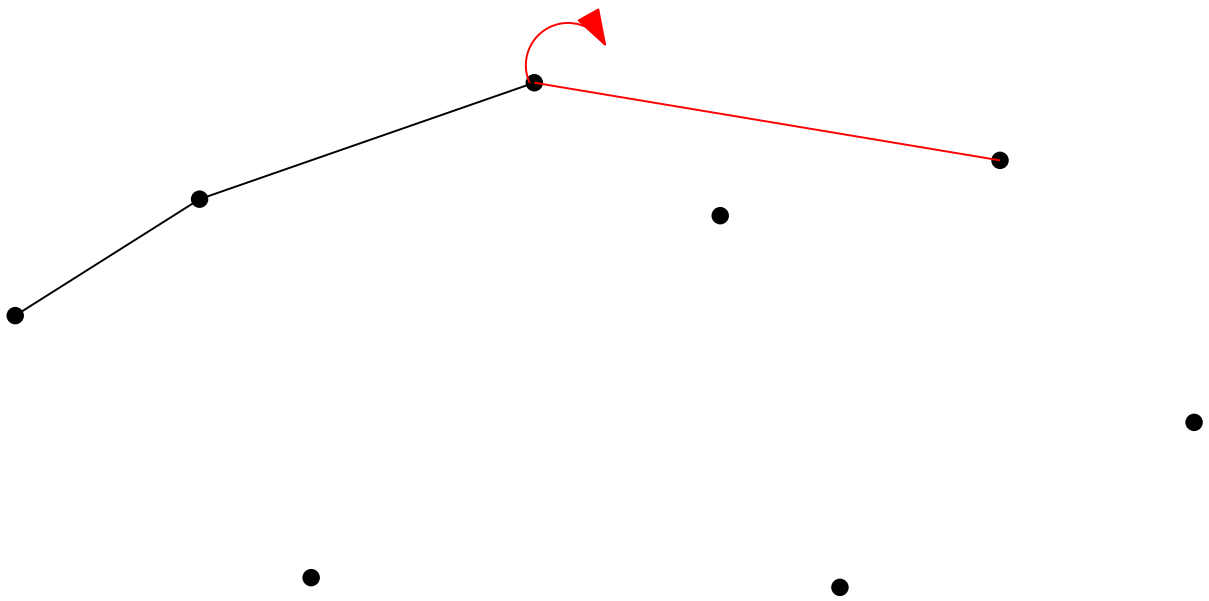
Gift Wrapping



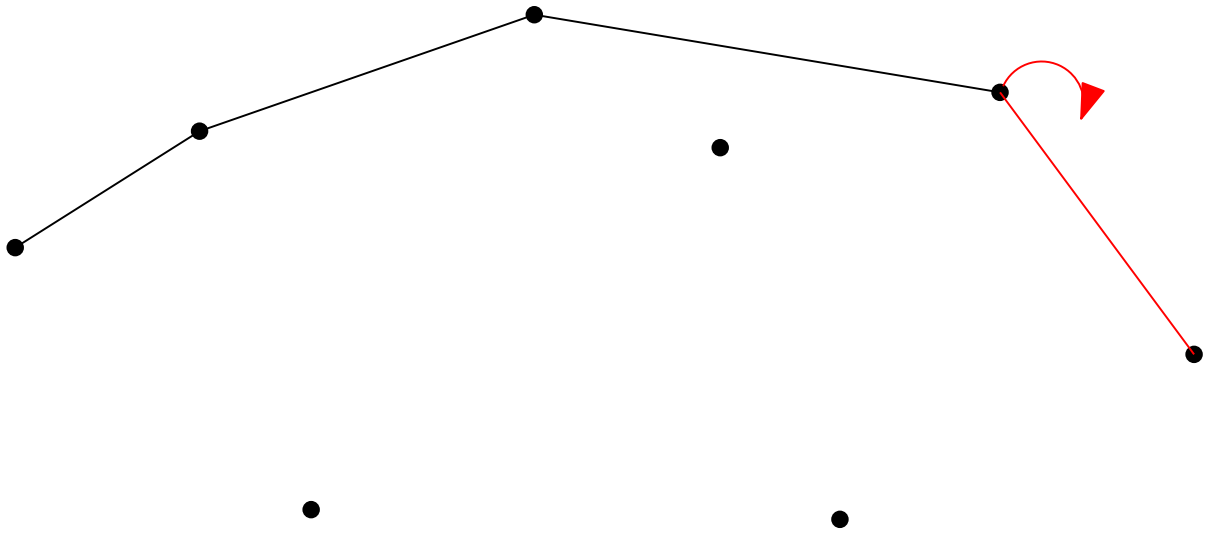
Gift Wrapping



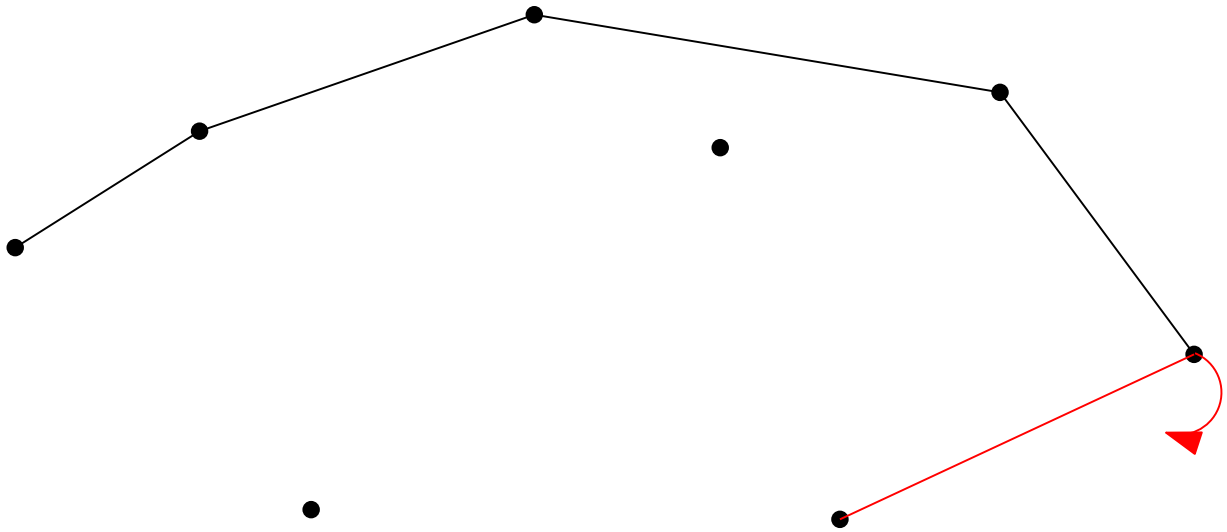
Gift Wrapping



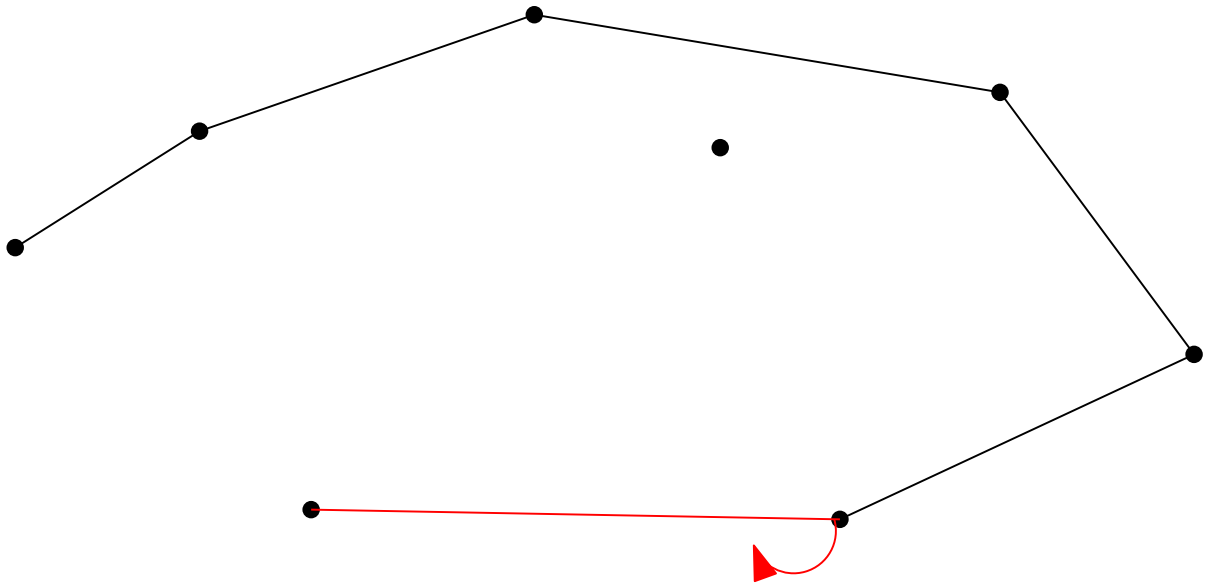
Gift Wrapping



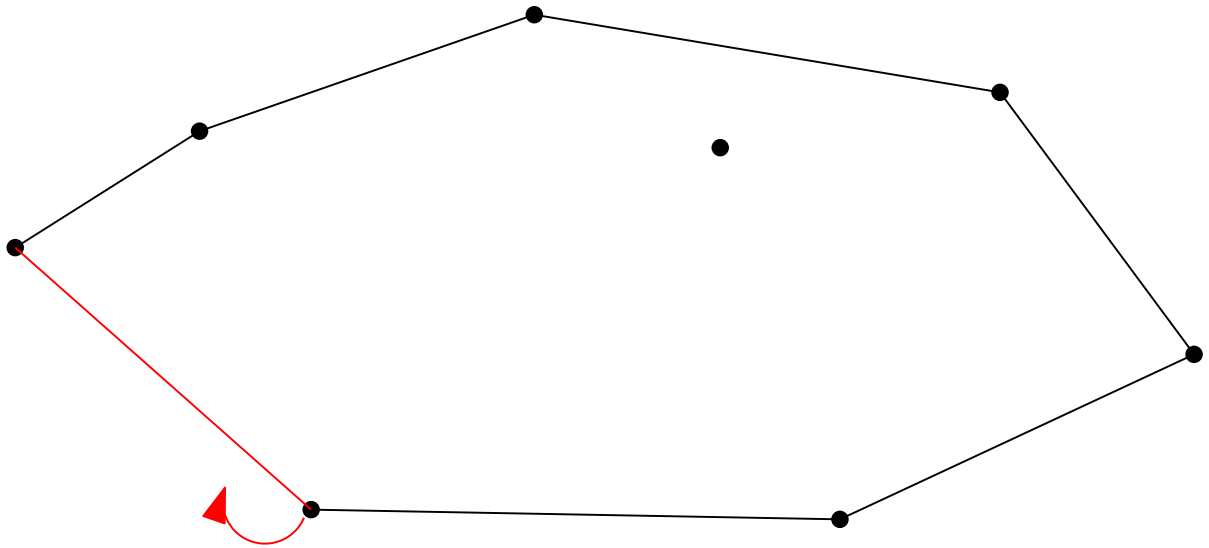
Gift Wrapping



Gift Wrapping



Gift Wrapping



Gift Wrapping - Pseudocode

$P \leftarrow \{$Punkt ganz links$\}$
$start \leftarrow$ Punkt ganz links
$current \leftarrow$ Punkt ganz links
repeat
  for alle $p \in P$ **do**
    if kein Punkt befindet sich links von der Gerade current—p **then**
      $P += p$
      $current \leftarrow p$
    end if
  end for
until current == start

$\Rightarrow O(n * k)$

Gift Wrapping - Pseudocode

$P \leftarrow \{$Punkt ganz links$\}$
$start \leftarrow$ Punkt ganz links
$current \leftarrow$ Punkt ganz links
repeat
  for alle $p \in P$ **do**
    if kein Punkt befindet sich links von der Gerade current—p **then**
      $P+ = p$
      $current \leftarrow p$
    end if
  end for
until current == start

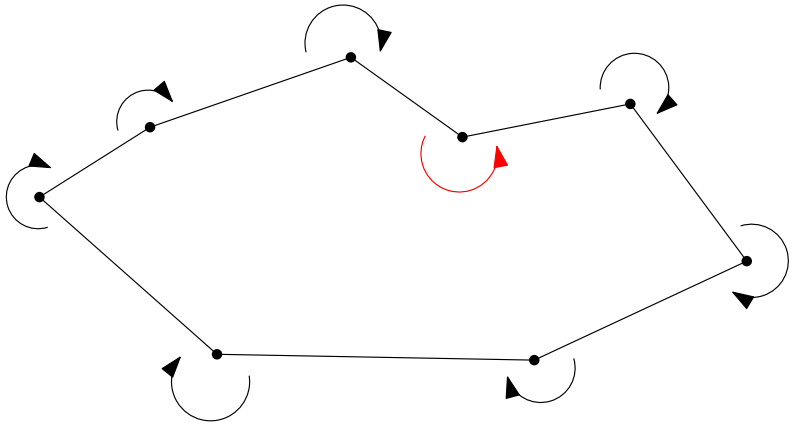
$\Rightarrow O(n * k)$

Gift Wrapping - Pseudocode

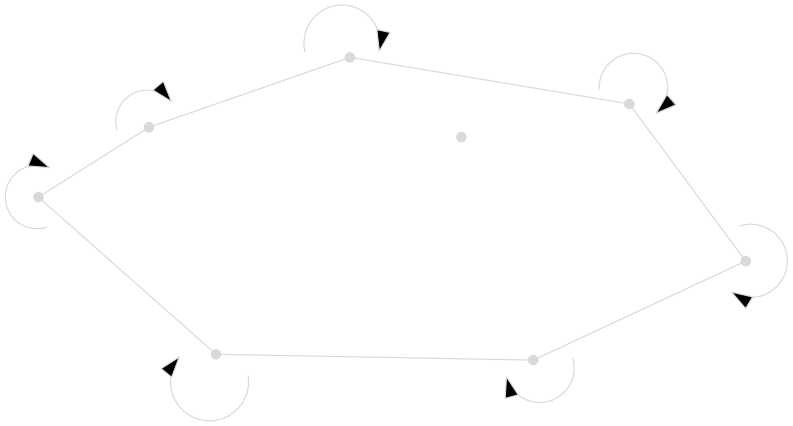
$P \leftarrow \{$Punkt ganz links$\}$
$start \leftarrow$ Punkt ganz links
$current \leftarrow$ Punkt ganz links
repeat
  for alle $p \in P$ **do**
    if kein Punkt befindet sich links von der Gerade current—p **then**
      $P += p$
      $current \leftarrow p$
    end if
  end for
until current == start

$\Rightarrow O(n * k)$

Weitere Beobachtungen zu konvexen Hüllen



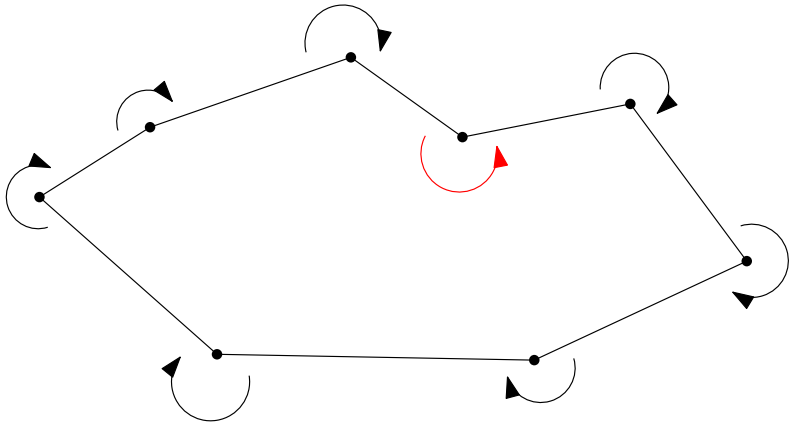
Beliebige Hülle mit Rechts- und Linksabbiegungen



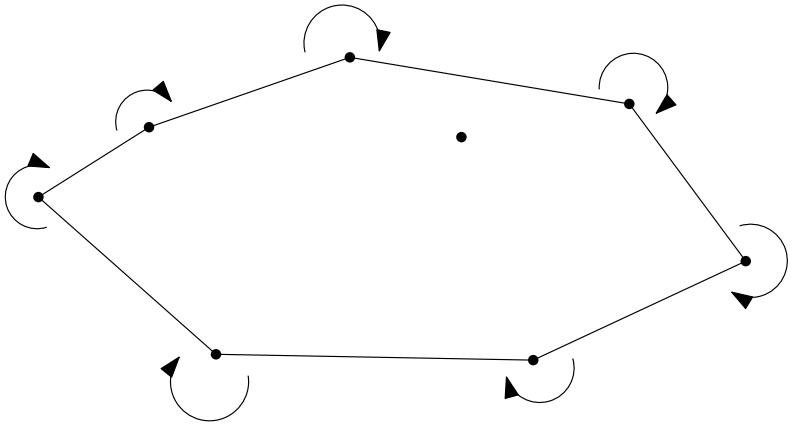
Konvexe Hülle nur mit Rechtsabbiegungen

$\Rightarrow$ vom Punkt ganz links bis zum Punkt ganz rechts schreitet die konvexe Hülle mit jeder Kante weiter immer weiter nach rechts.

Weitere Beobachtungen zu konvexen Hüllen



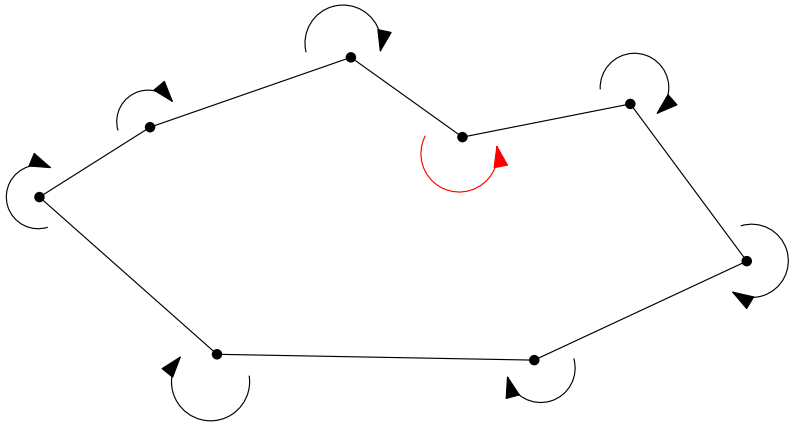
Beliebige Hülle mit Rechts- und Linksabbiegungen



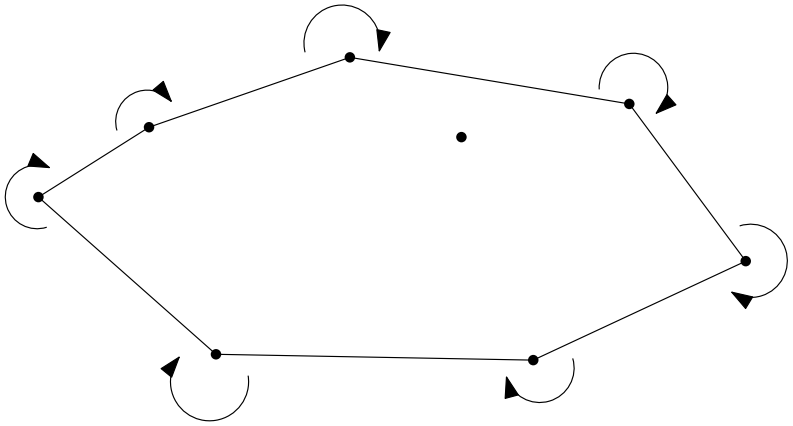
Konvexe Hülle nur mit Rechtsabbiegungen

$\Rightarrow$ vom Punkt ganz links bis zum Punkt ganz rechts schreitet die konvexe Hülle mit jeder Kante weiter immer weiter nach rechts.

Weitere Beobachtungen zu konvexen Hüllen



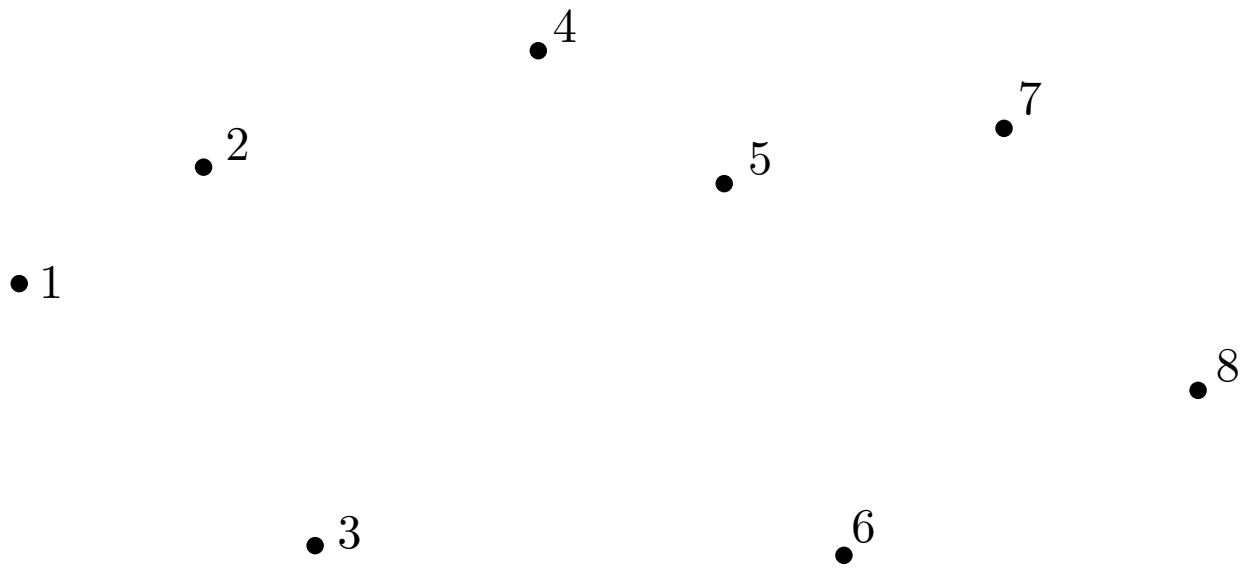
Beliebige Hülle mit Rechts- und Linksabbiegungen



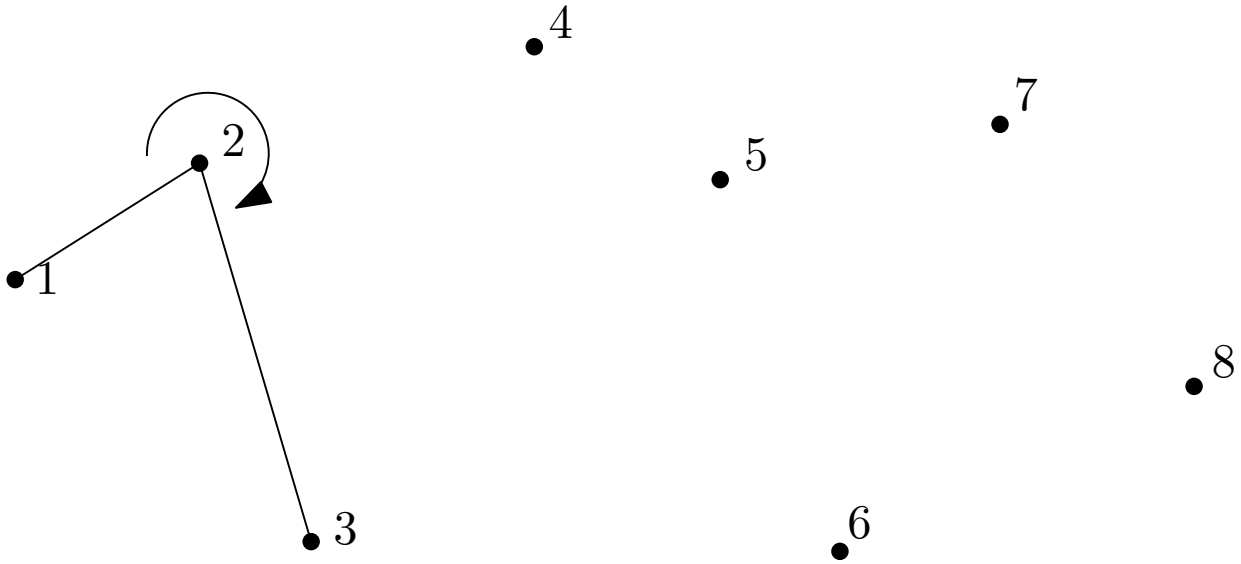
Konvexe Hülle nur mit Rechtsabbiegungen

$\Rightarrow$ vom Punkt ganz links bis zum Punkt ganz rechts schreitet die konvexe Hülle mit jeder Kante weiter immer weiter nach rechts.

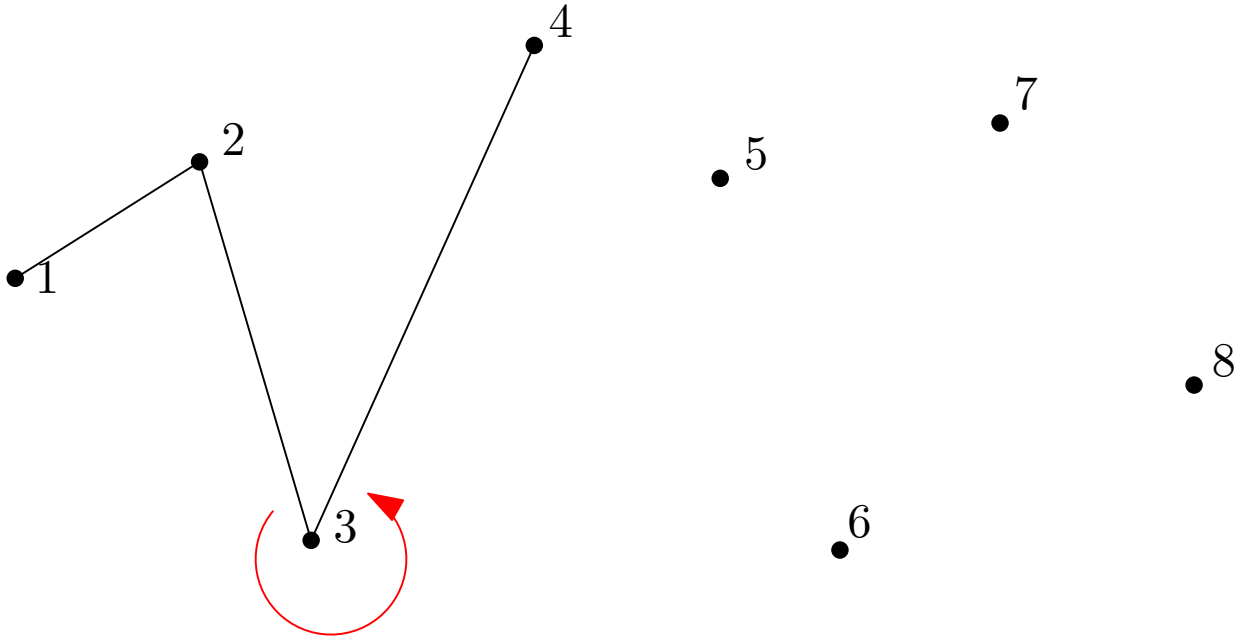
Konvexe Hülle - Weitere Überlegungen



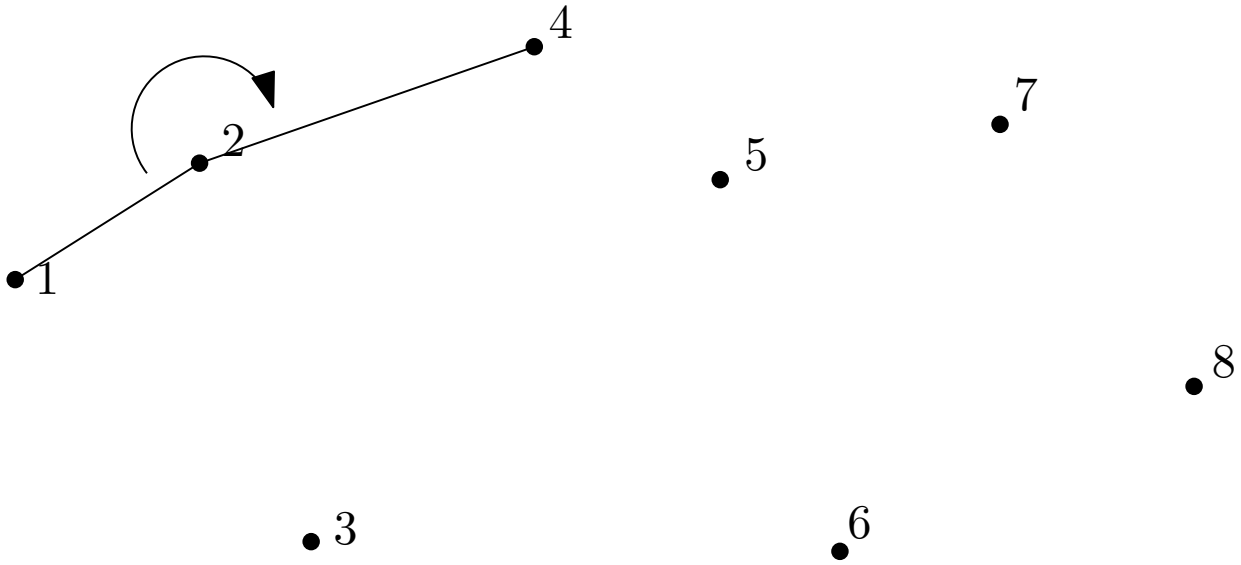
Konvexe Hülle - Weitere Überlegungen



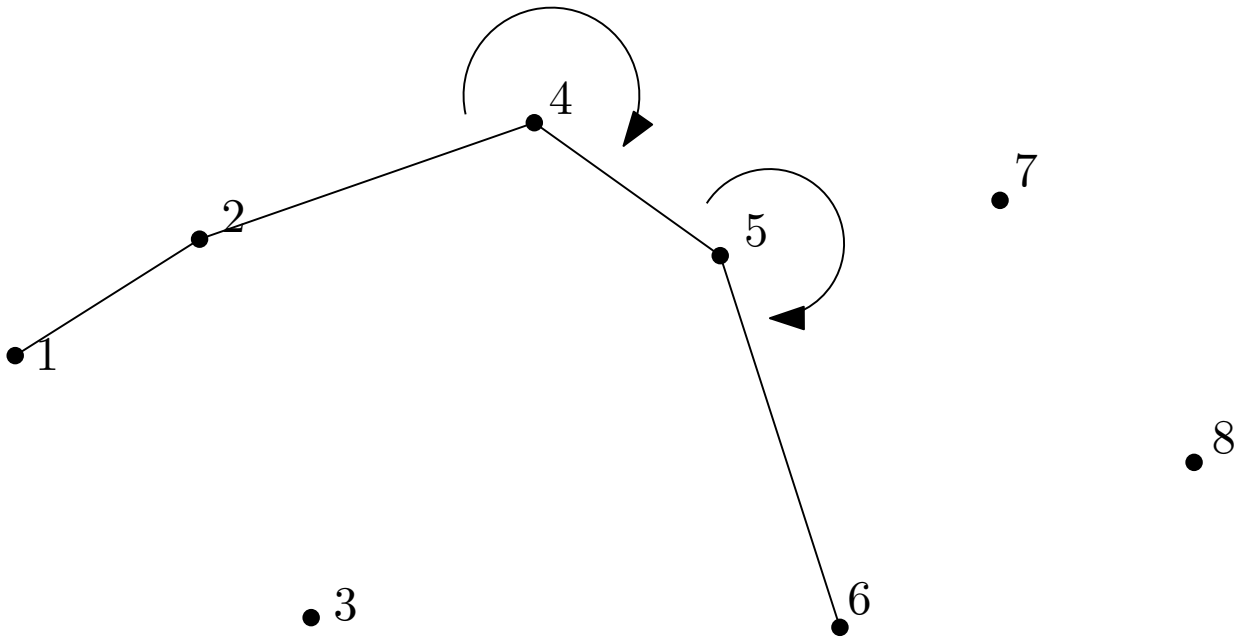
Konvexe Hülle - Weitere Überlegungen



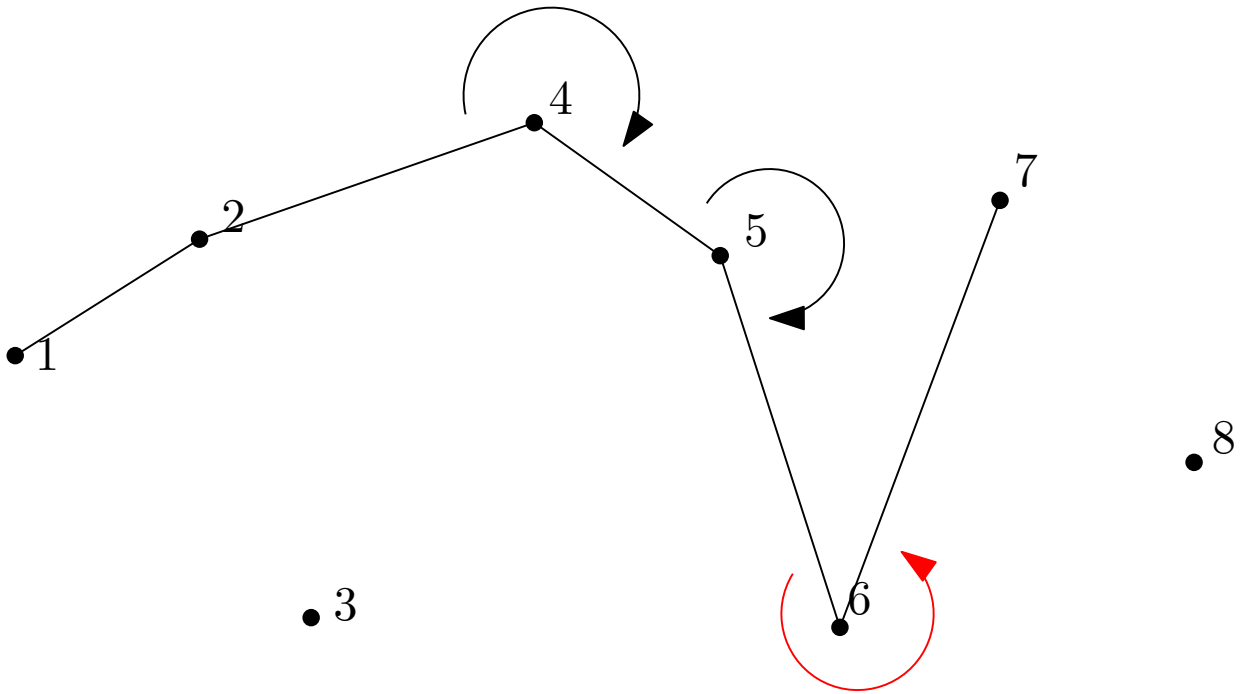
Konvexe Hülle - Weitere Überlegungen



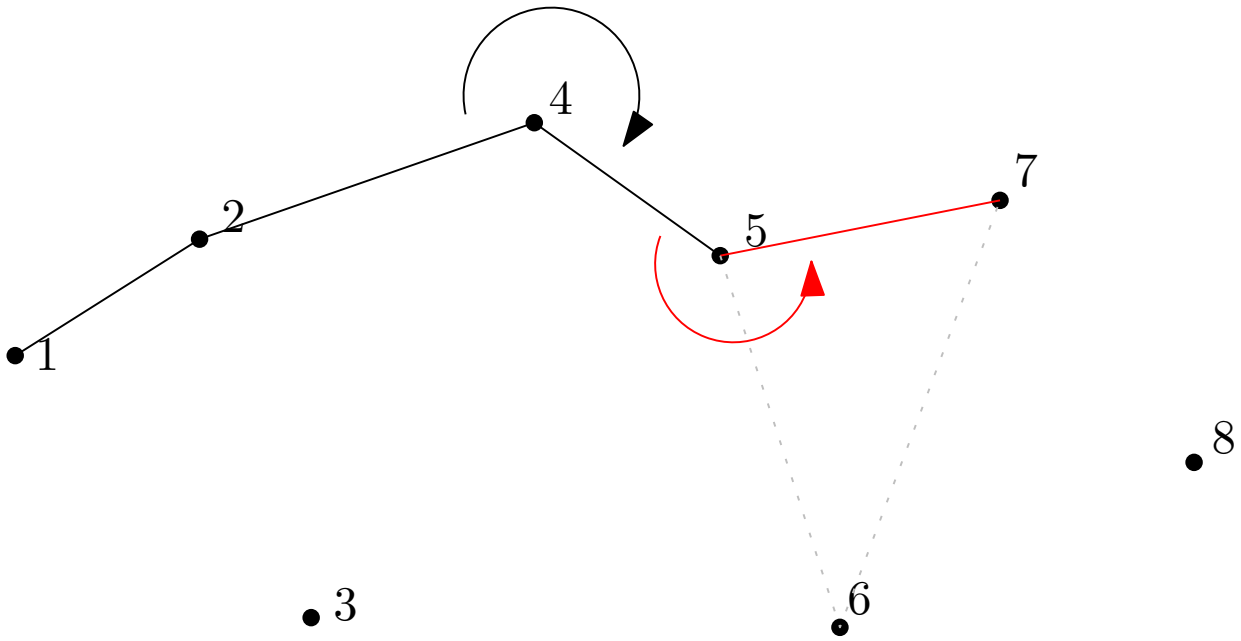
Konvexe Hülle - Weitere Überlegungen



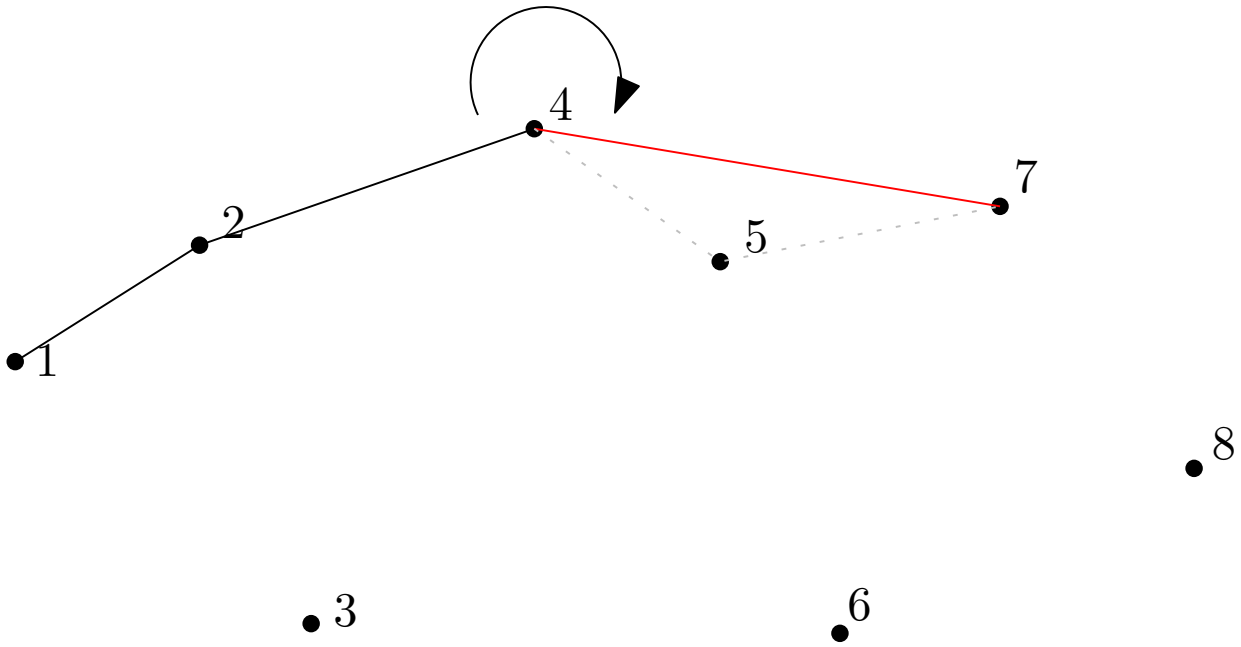
Konvexe Hülle - Weitere Überlegungen



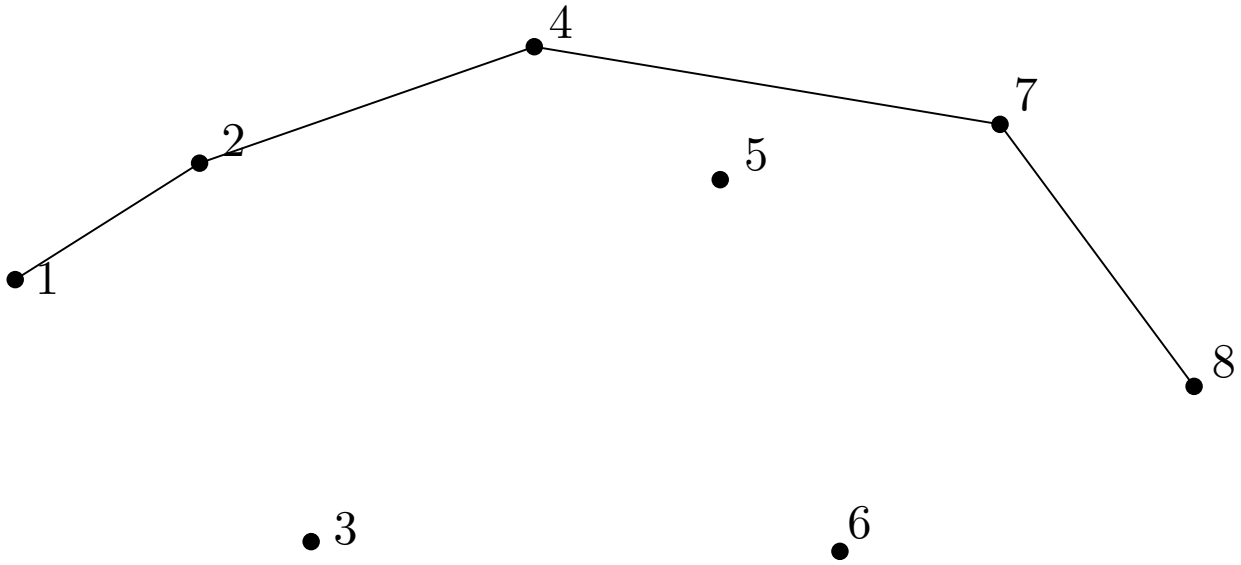
Konvexe Hülle - Weitere Überlegungen



Konvexe Hülle - Weitere Überlegungen



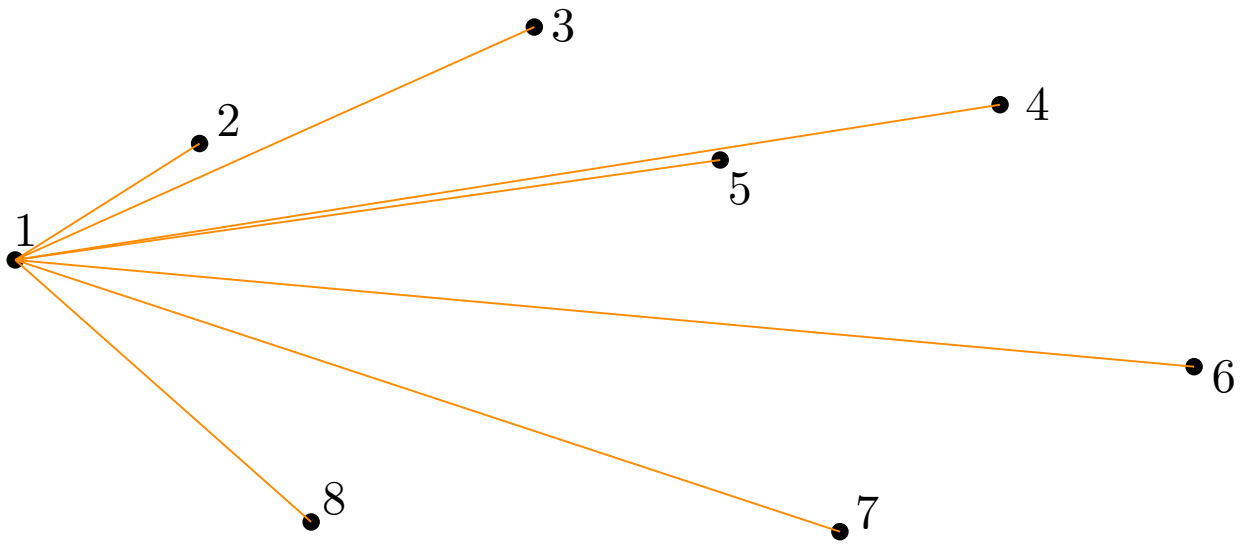
Konvexe Hülle - Weitere Überlegungen



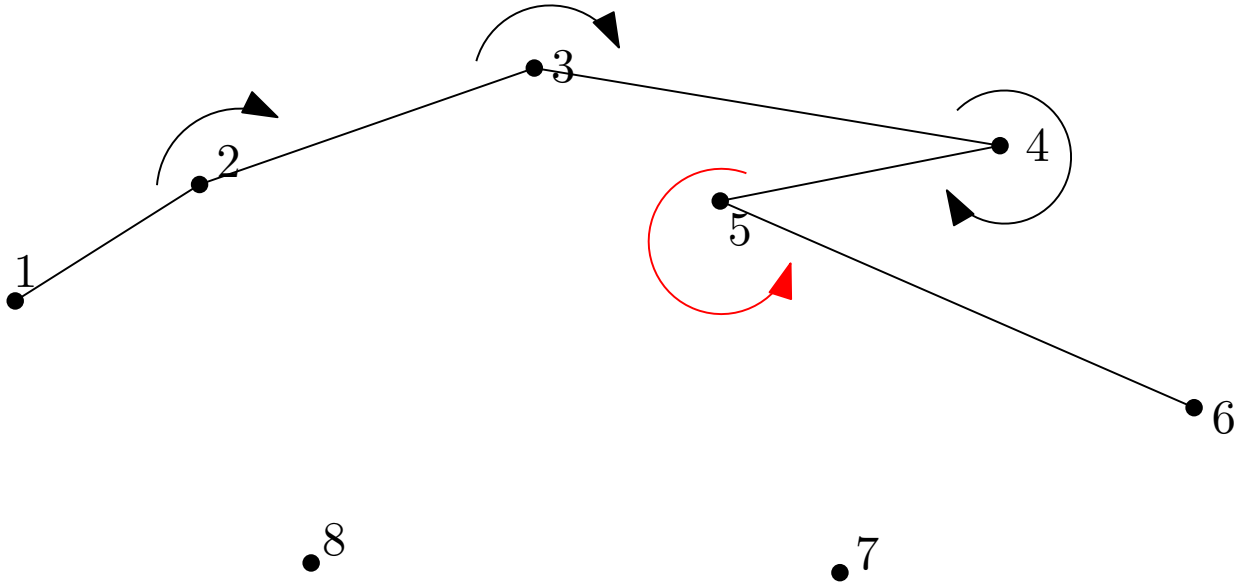
Graham Scan

Kombiniert man die Tatsache der Rechtsabbiegungen und der Möglichkeit von links nach rechts zu gehen, kann man die Punkte stattdessen auch nach Winkeln sortieren. Diesen Algorithmus nennt man Graham Scan.

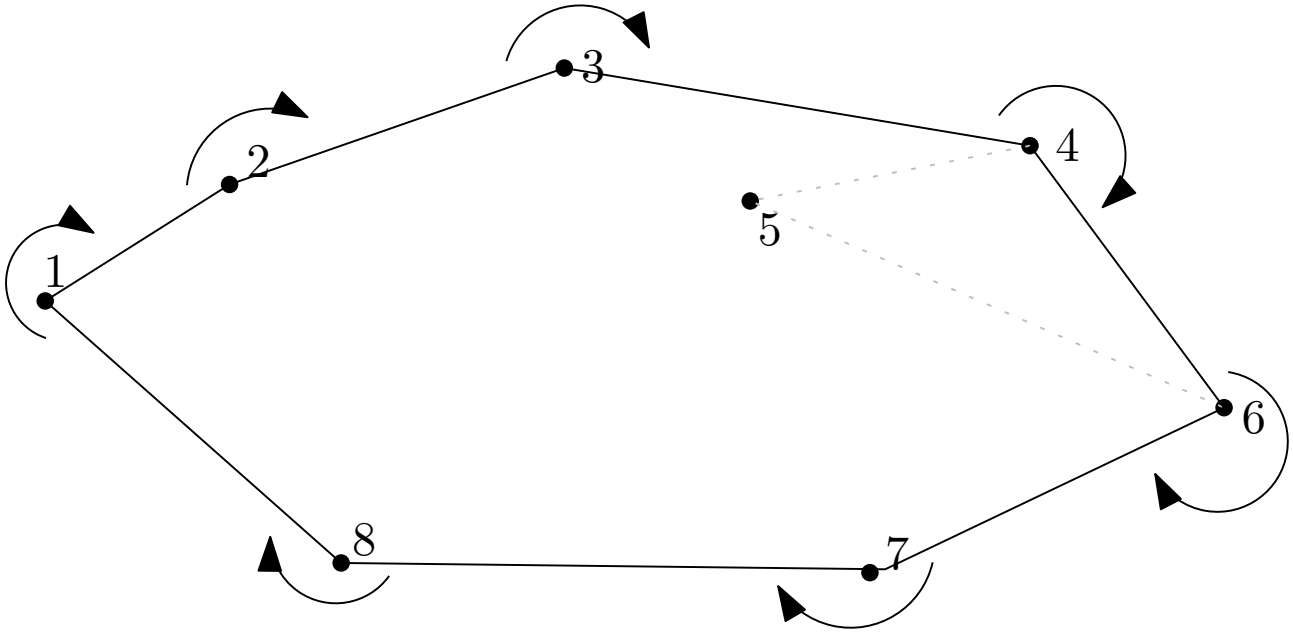
Graham Scan



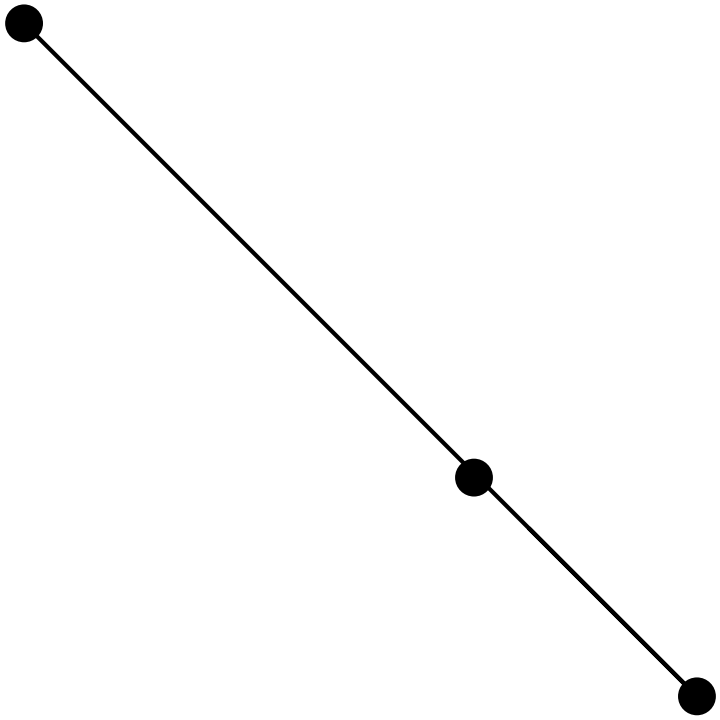
Graham Scan



Graham Scan



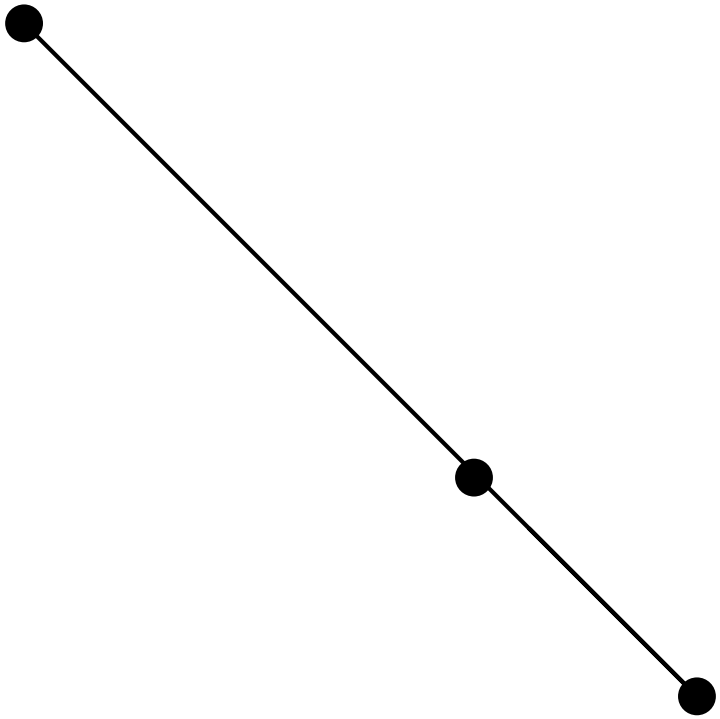
Noch eine paar letzte Sonderfälle



3 Punkte auf einer Geraden $\Rightarrow$ muss wie eine Linksabbiegung interpretiert werden

2 Punkte mit gleichem Winkel $\Rightarrow$ Punkte müssen lexikographisch sortiert sein

Noch eine paar letzte Sonderfälle



3 Punkte auf einer Geraden $\Rightarrow$ muss wie eine Linksabbiegung interpretiert werden

2 Punkte mit gleichem Winkel $\Rightarrow$ Punkte müssen lexikographisch sortiert sein

Graham Scan - Pseudocode

Sei P eine Liste mit den Punkten P_1 bis P_n.
Sei S die Liste, die am Ende ausgegeben wird.

$S \leftarrow sortiereLexikographisch(P)$
$i \leftarrow 1$
while $i < |P|$ **do**
  if $S_{i-1} \rightarrow S_i$ nach $S_i \rightarrow S_{i+1}$ eine Rechtsabbiegung **then**
    $i \leftarrow i + 1$
  else
    Entferne Element S_i aus S
    $i \leftarrow i - 1$
  end if
end while

$\Rightarrow O(n * log(n))$

Graham Scan - Pseudocode

Sei P eine Liste mit den Punkten P_1 bis P_n.
Sei S die Liste, die am Ende ausgegeben wird.

$S \leftarrow sortiereLexikographisch(P)$
$i \leftarrow 1$
while $i < |P|$ **do**
  if $S_{i-1} \rightarrow S_i$ nach $S_i \rightarrow S_{i+1}$ eine Rechtsabbiegung **then**
    $i \leftarrow i + 1$
  else
    Entferne Element S_i aus S
    $i \leftarrow i - 1$
  end if
end while

$\Rightarrow O(n * log(n))$

Graham Scan - Pseudocode

Sei P eine Liste mit den Punkten P_1 bis P_n.
Sei S die Liste, die am Ende ausgegeben wird.

$S \leftarrow sortiereLexikographisch(P)$
$i \leftarrow 1$
while $i < |P|$ **do**
  if $S_{i-1} \rightarrow S_i$ nach $S_i \rightarrow S_{i+1}$ eine Rechtsabbiegung **then**
    $i \leftarrow i + 1$
  else
    Entferne Element S_i aus S
    $i \leftarrow i - 1$
  end if
end while

$\Rightarrow O(n * log(n))$

Graham Scan - Pseudocode

Sei P eine Liste mit den Punkten P_1 bis P_n.
Sei S die Liste, die am Ende ausgegeben wird.

$S \leftarrow sortiereLexikographisch(P)$
$i \leftarrow 1$
while $i < |P|$ **do**
  if $S_{i-1} \rightarrow S_i$ nach $S_i \rightarrow S_{i+1}$ eine Rechtsabbiegung **then**
    $i \leftarrow i + 1$
  else
    Entferne Element S_i aus S
    $i \leftarrow i - 1$
  end if
end while

$\Rightarrow O(n * log(n))$

Graham Scan - Letzte Bemerkungen

Rechts- oder Linksabbiegung: Für 3 Punkte $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ kann man mit Hilfe des Kreuzproduktes herausfinden, wie sie zueinander stehen:

$$(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

Ist das Ergebnis 0, so sind sie colinear, falls positiv, dann ist dies eine Linksabbiegung und sonst eine Rechtsabbiegung.

Bereichssuchen - Motivation

- Datenbankabfrage
- Google Maps

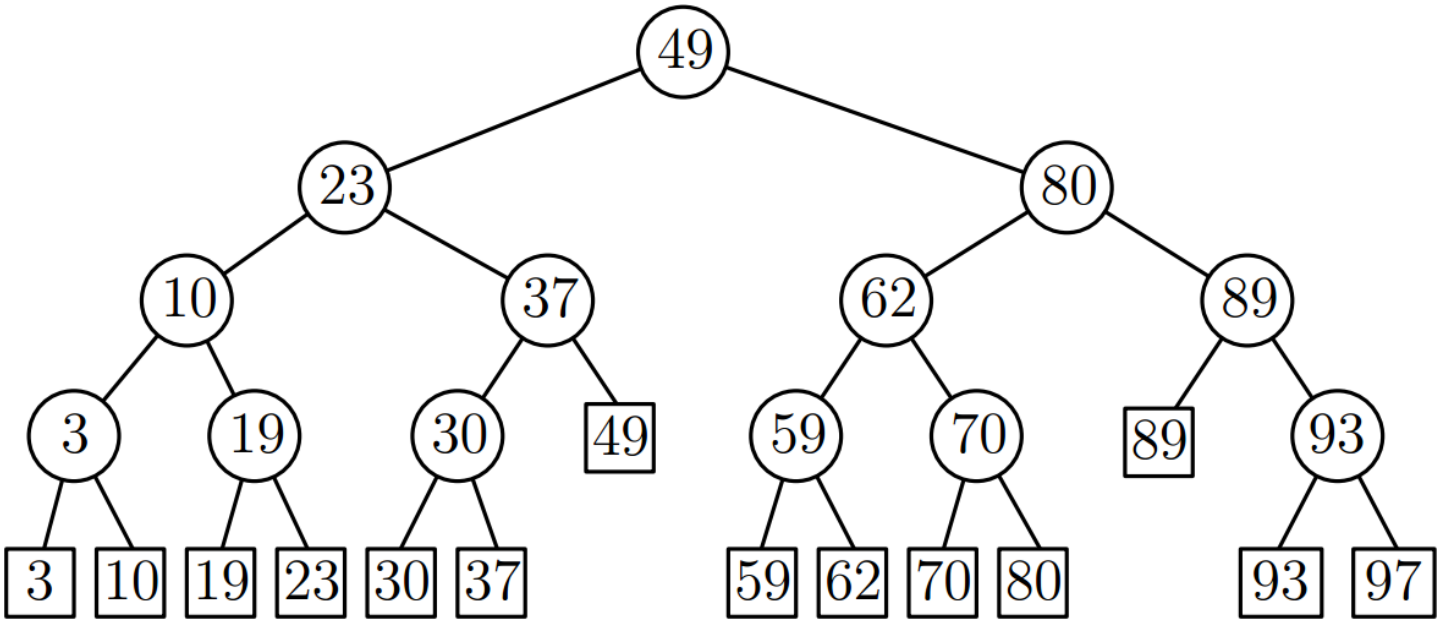
1D Bereichssuchen

Gegeben sind n verschiedene Zahlen $z_1, \ldots z_n$ und der Bereich [x,y].
Gesucht sind nun die Zahlen z_i, die sich im Bereich [x,y] befinden
Idee: Sortiere die Zahlen und erstelle einen binären Baum. In diesem werden nun die Zahlen gesucht.

1D Bereichssuchen

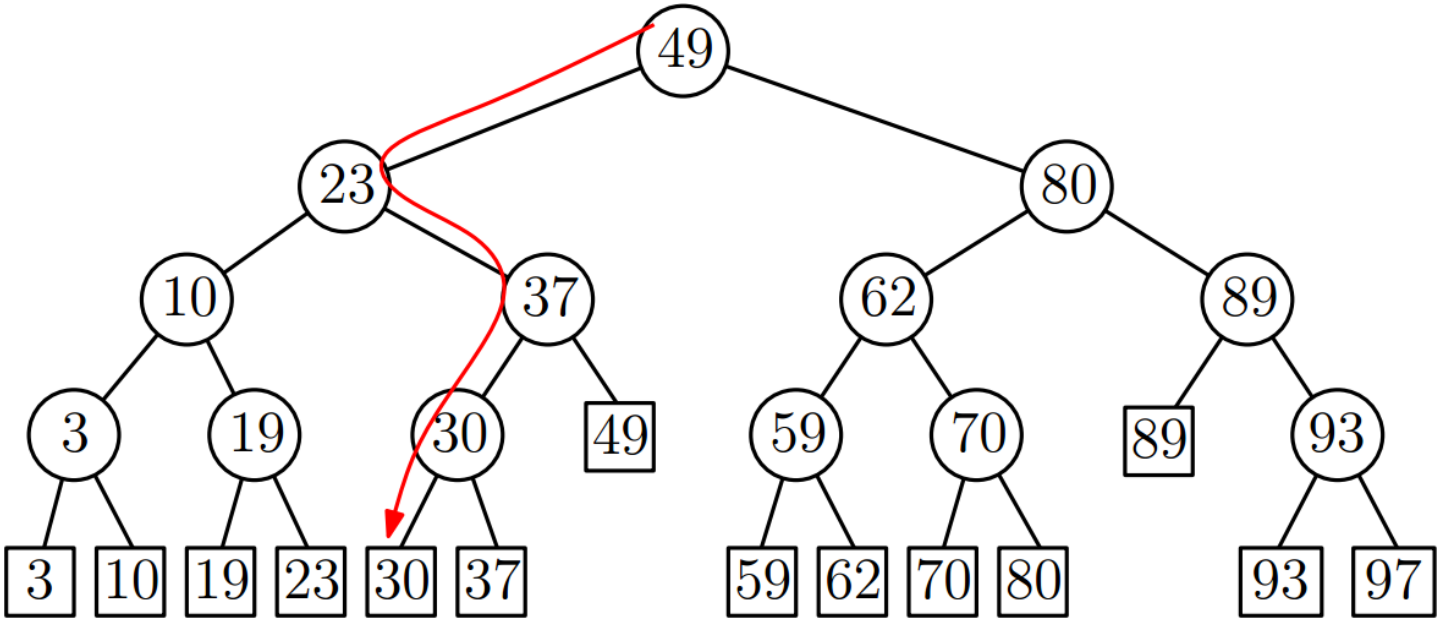
Gegeben sind n verschiedene Zahlen $z_1, \ldots z_n$ und der Bereich [x,y].
Gesucht sind nun die Zahlen z_i, die sich im Bereich [x,y] befinden
Idee: Sortiere die Zahlen und erstelle einen binären Baum. In diesem werden nun die Zahlen gesucht.

1D Bereichssuchen



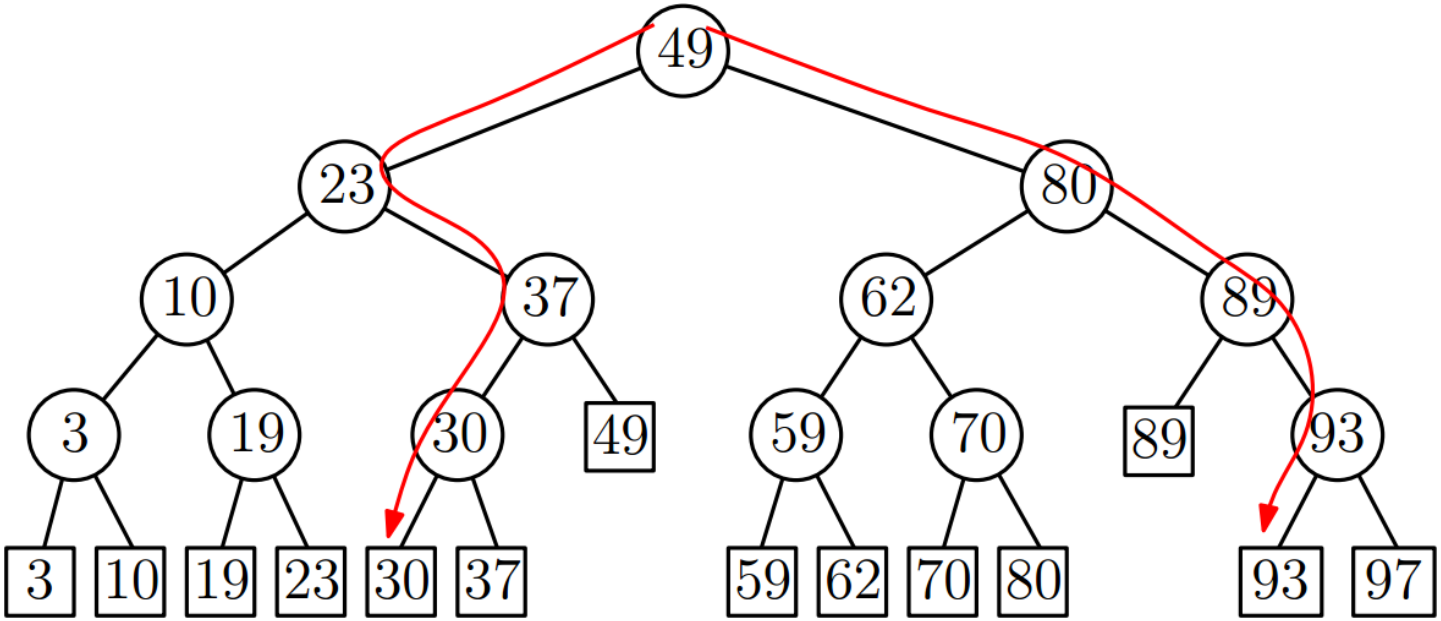
Binärer Baum zu den Zahlen 3, 10, 19, 23, 30, 37, 49, 59, 62, 70, 80, 89, 93, 97

1D Bereichssuchen



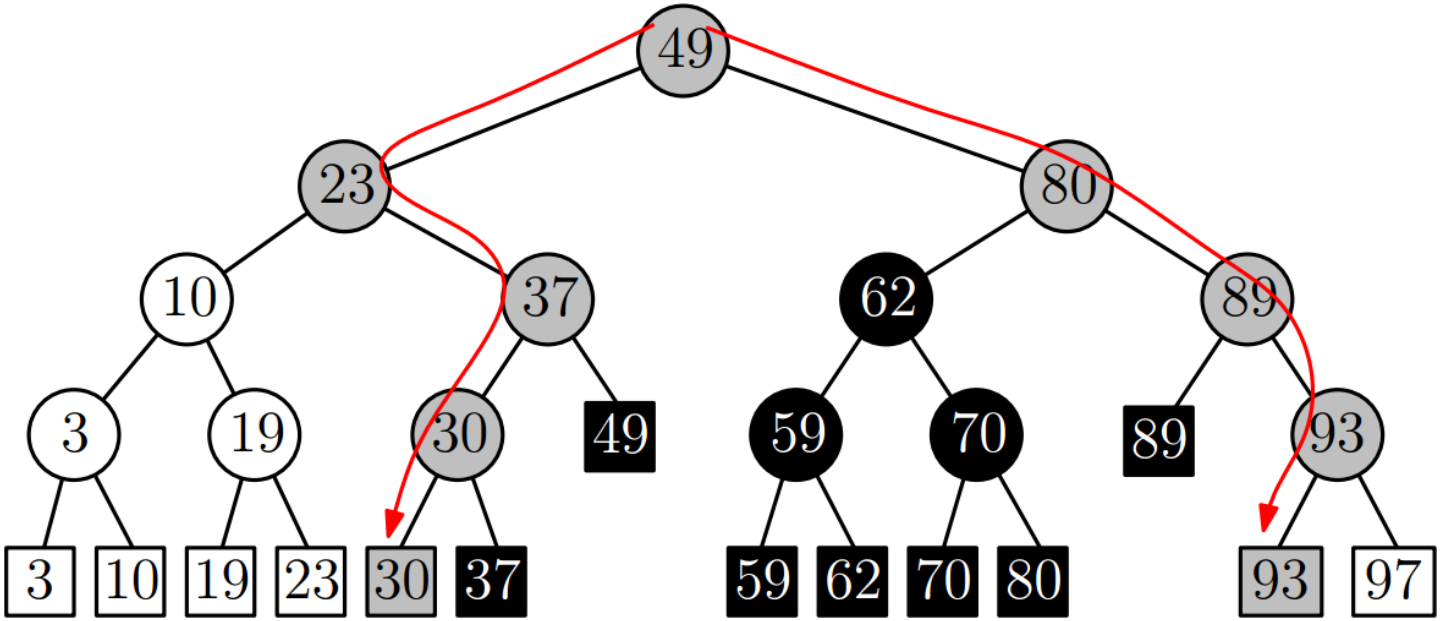
Suchpfad für die Zahl 25

1D Bereichssuchen



Suchpfad für 25 und 90

1D Bereichssuchen

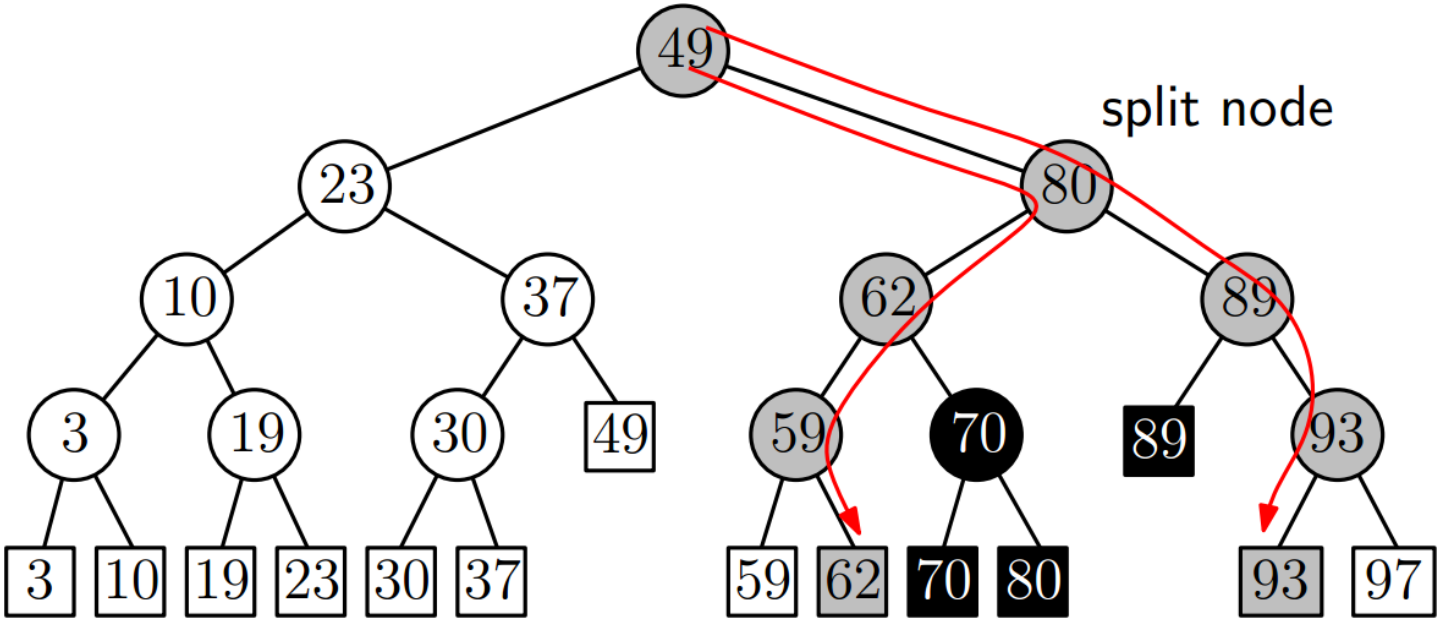


Bereichssuche für [25, 90]

1D Bereichssuchen

Weiße Knoten: werden für die Suche nie betrachtet
Graue Knoten: wurden besucht, es ist aber unklar, ob sie zur Ausgabe gehören
Schwarze Knoten: wurden besucht und gesamter Unterteilbaum gehört zur Ausgabe.

1D Bereichssuchen



Bereichssuche für [61, 90]

1D Bereichssuchen - SplitNode

Sei T der Baum und x, x' zwei Werte mit $x \leq x'$

FindeSplitNode(T, x, x')

$v \leftarrow root(T)$

while v ist kein Blatt und ($x' \leq x_v$ oder $x > x_v$) **do**

  if $x' \leq x_v$ **then**

    $v \leftarrow lc(v)$

  else

    $v \leftarrow rc(v)$

  end if

end while

return v

1D Bereichssuchen - SplitNode

Sei T der Baum und x, x' zwei Werte mit $x \leq x'$

FindeSplitNode(T, x, x')

$v \leftarrow root(T)$
while v ist kein Blatt und ($x' \leq x_v$ oder $x > x_v$) **do**
  if $x' \leq x_v$ **then**
    $v \leftarrow lc(v)$
  else
    $v \leftarrow rc(v)$
  end if
end while
return v

1D Bereichssuchen - SplitNode

Sei T der Baum und x, x' zwei Werte mit $x \leq x'$

FindeSplitNode(T, x, x')

$v \leftarrow root(T)$
while v ist kein Blatt und ($x' \leq x_v$ oder $x > x_v$) **do**
  if $x' \leq x_v$ **then**
    $v \leftarrow lc(v)$
  else
    $v \leftarrow rc(v)$
  end if
end while
return v

1D Bereichssuchen - Pseudocode

$1DBereichssuche(T, x, x')$

$v_{split} \leftarrow FindSplitNode(T, x, x')$

if v_{split} ist ein Blatt **then**

Prüfe ob v_{split} gemeldet werden muss

else

$v \leftarrow lc(v_{split})$

while v ist kein Blatt **do**

if $x \leq x_v$ **then**

$meldeTeilbaum(rc(v))$

$v \leftarrow lc(v)$

else

$v \leftarrow rc(v)$

end if

end while

Prüfe, ob v gemeldet werden muss

mache das gleiche noch äquivalent mit x'...

1D Bereichssuchen - Pseudocode

$1DBereichssuche(T, x, x')$

$v_{split} \leftarrow FindSplitNode(T, x, x')$
if v_{split} ist ein Blatt **then**
  Prüfe ob v_{split} gemeldet werden muss
else
  $v \leftarrow lc(v_{split})$
  while v ist kein Blatt **do**
    if $x \leq x_v$ **then**
      $meldeTeilbaum(rc(v))$
      $v \leftarrow lc(v)$
    else
      $v \leftarrow rc(v)$
    end if
  end while
  Prüfe, ob v gemeldet werden muss
  mache das gleiche noch äquivalent mit x'...

1D Bereichssuchen - Pseudocode

$1DBereichssuche(T, x, x')$

$v_{split} \leftarrow FindSplitNode(T, x, x')$
if v_{split} ist ein Blatt **then**
    Prüfe ob v_{split} gemeldet werden muss
else
    $v \leftarrow lc(v_{split})$
    while v ist kein Blatt **do**
        if $x \leq x_v$ **then**
            $meldeTeilbaum(rc(v))$
            $v \leftarrow lc(v)$
        else
            $v \leftarrow rc(v)$
        end if
    end while
    Prüfe, ob v gemeldet werden muss
    mache das gleiche noch äquivalent mit x'...

1D Bereichssuchen - Pseudocode

$1DBereichssuche(T, x, x')$

$v_{split} \leftarrow FindSplitNode(T, x, x')$
if v_{split} ist ein Blatt **then**
    Prüfe ob v_{split} gemeldet werden muss
else
    $v \leftarrow lc(v_{split})$
    while v ist kein Blatt **do**
        if $x \leq x_v$ **then**
            $meldeTeilbaum(rc(v))$
            $v \leftarrow lc(v)$
        else
            $v \leftarrow rc(v)$
        end if
    end while
    Prüfe, ob v gemeldet werden muss
    mache das gleiche noch äquivalent mit x'...

1D Bereichssuchen - Pseudocode

1*DBereichssuche*(T, x, x')

$v_{split} \leftarrow$ *FindSplitNode*(T, x, x')

if v_{split} ist ein Blatt **then**

Prüfe ob v_{split} gemeldet werden muss

else

$v \leftarrow lc(v_{split})$

while v ist kein Blatt **do**

if $x \leq x_v$ **then**

meldeTeilbaum$(rc(v))$

$v \leftarrow lc(v)$

else

$v \leftarrow rc(v)$

end if

end while

Prüfe, ob v gemeldet werden muss

mache das gleiche noch äquivalent mit x'...

1D Bereichssuchen - Laufzeit

Graue Knoten: Es gibt genau zwei Pfade mit grauen Knoten. Da der Baum balanciert ist $\Rightarrow O(log(n))$

Schwarze Knoten: Ein Teilbaum mit k Knoten hat k-1 interne Knoten. Wir finden also k Knoten durch das Traversieren $\Rightarrow O(k)$

$\Rightarrow O(log(n) + k)$

Der Baum muss allerdings auch erstmal erstellt werden. Das liegt in $O(n)$.

1D Bereichssuchen - Laufzeit

Graue Knoten: Es gibt genau zwei Pfade mit grauen Knoten. Da der Baum balanciert ist $\Rightarrow O(log(n))$
Schwarze Knoten: Ein Teilbaum mit k Knoten hat k-1 interne Knoten. Wir finden also k Knoten durch das Traversieren $\Rightarrow O(k)$

$\Rightarrow O(log(n) + k)$

Der Baum muss allerdings auch erstmal erstellt werden. Das liegt in $O(n)$.

1D Bereichssuchen - Laufzeit

Graue Knoten: Es gibt genau zwei Pfade mit grauen Knoten. Da der Baum balanciert ist $\Rightarrow O(log(n))$
Schwarze Knoten: Ein Teilbaum mit k Knoten hat k-1 interne Knoten. Wir finden also k Knoten durch das Traversieren $\Rightarrow O(k)$

$\Rightarrow O(log(n) + k)$

Der Baum muss allerdings auch erstmal erstellt werden. Das liegt in $O(n)$.

1D Bereichssuchen - Laufzeit

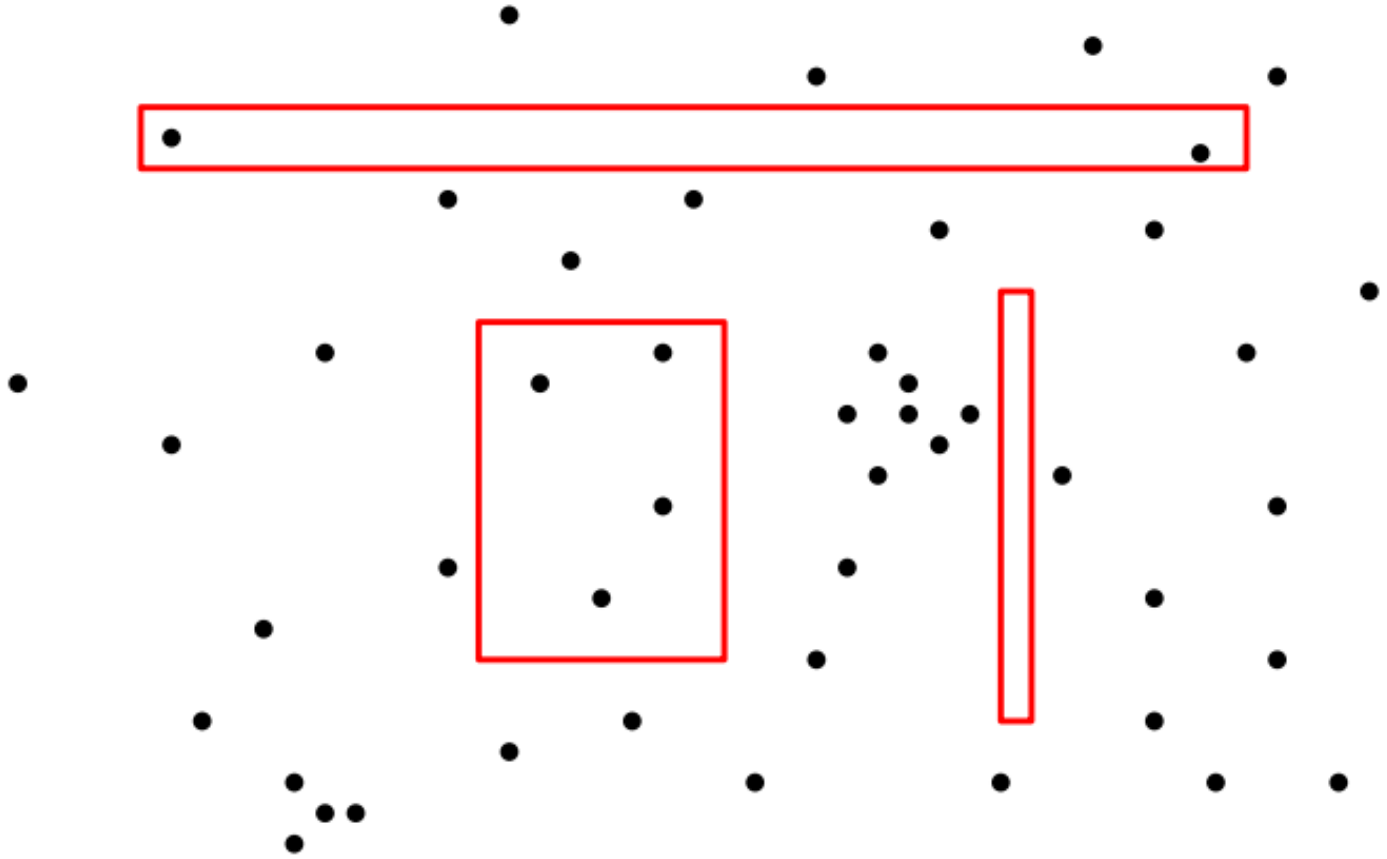
Graue Knoten: Es gibt genau zwei Pfade mit grauen Knoten. Da der Baum balanciert ist $\Rightarrow O(log(n))$
Schwarze Knoten: Ein Teilbaum mit k Knoten hat k-1 interne Knoten. Wir finden also k Knoten durch das Traversieren $\Rightarrow O(k)$

$\Rightarrow O(log(n) + k)$

Der Baum muss allerdings auch erstmal erstellt werden. Das liegt in $O(n)$.

kd-Baum

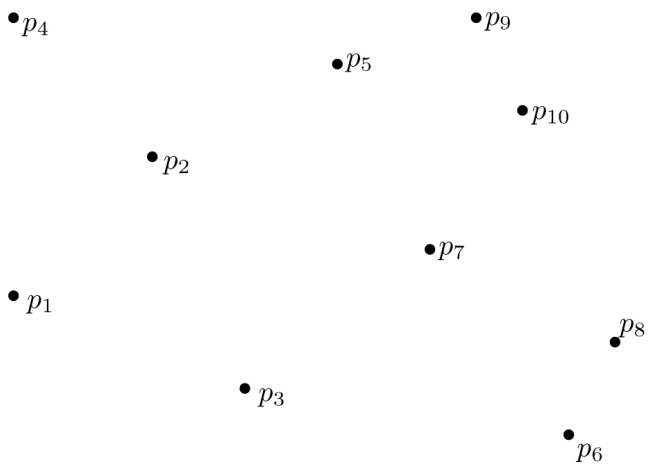
Im 2-dimensionalen sind Bereichssuchen ähnlich zu machen.
Dafür werden **kd-Bäume** benutzt.



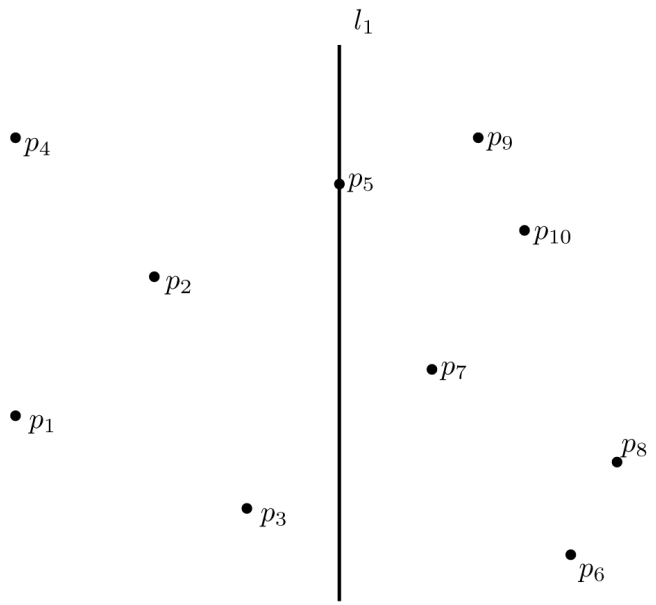
kd-Baum - Die Idee

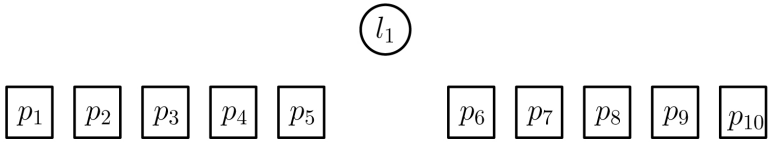
Wir bauen einen Baum ähnlich wie den für die 1-D Suche. Da wir im 2D x und y betrachten müssen, splitten wir beim kd-Baum nacheinander erst nach x und dann nach y auf.

kd-Baum

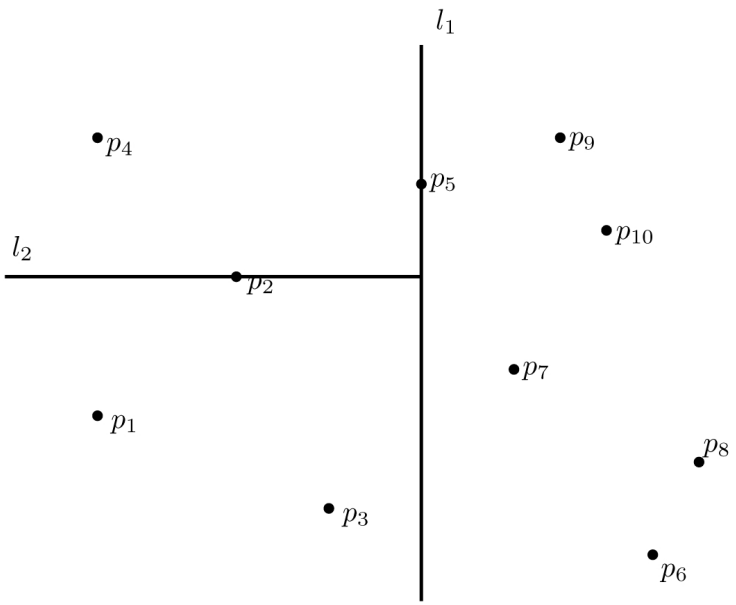


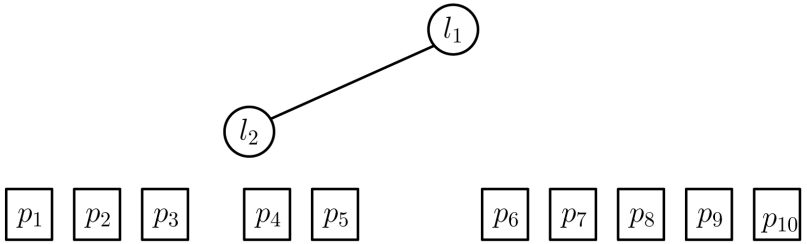
kd-Baum



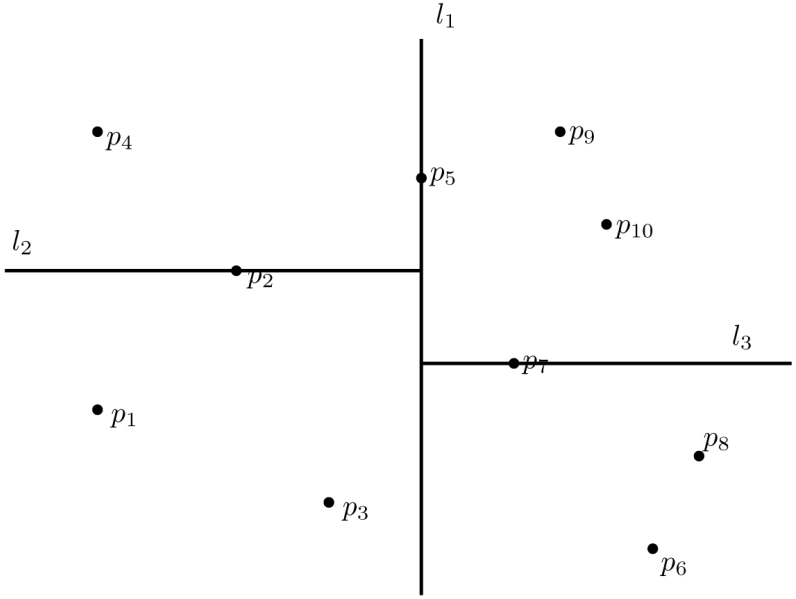


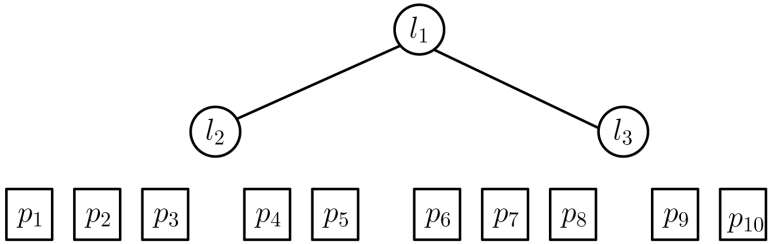
kd-Baum


l_1
l_2
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}

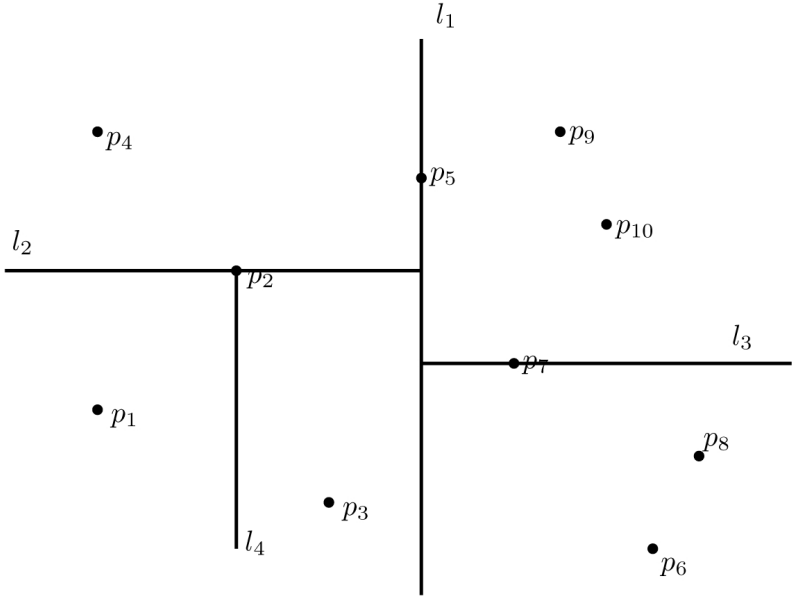

l_1
l_2
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}

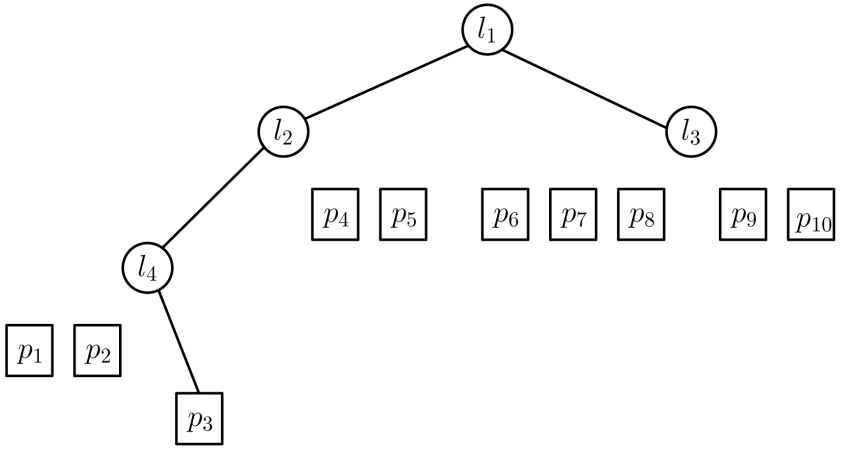
kd-Baum


l_1
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
p_1
p_8
p_3
p_6

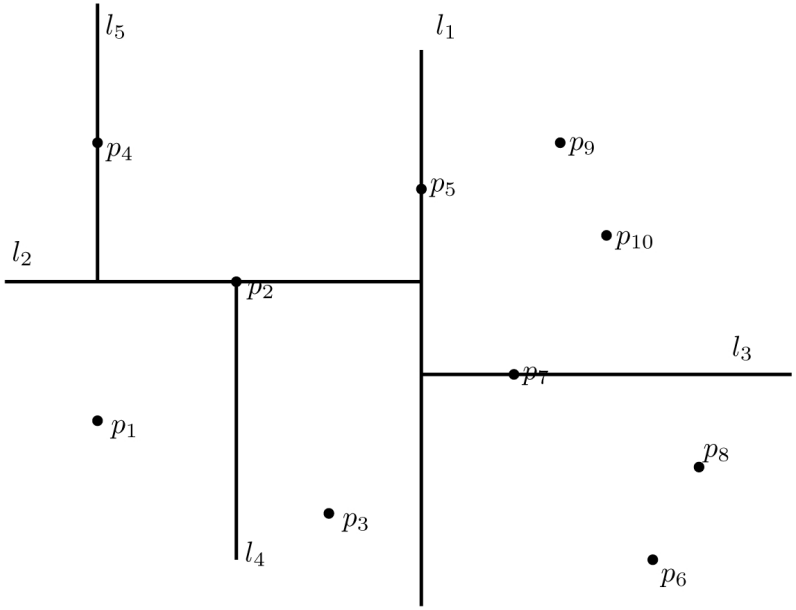

l_1
l_2
l_3
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10}

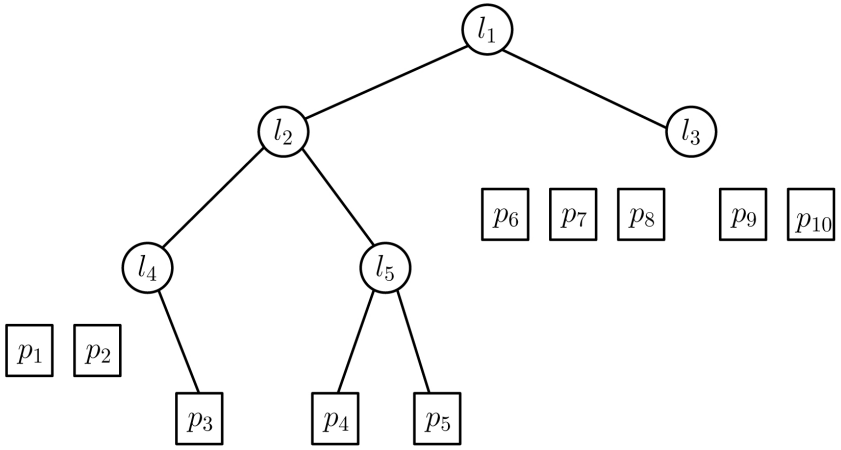
kd-Baum


l_1
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
p_1
p_8
p_3
l_4
p_6

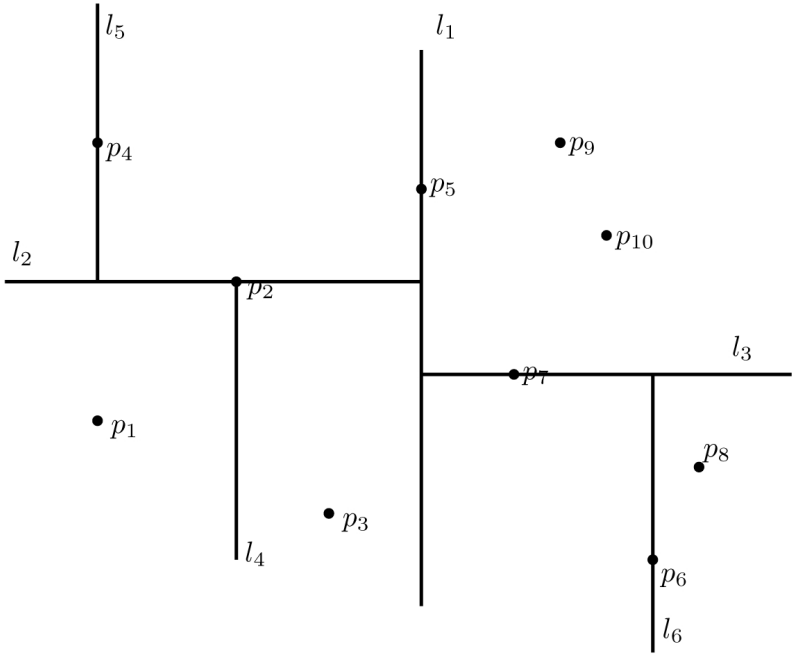

l_1
l_2
l_3
p_4 p_5 p_6 p_7 p_8 p_9 p_{10}
l_4
p_1 p_2
p_3

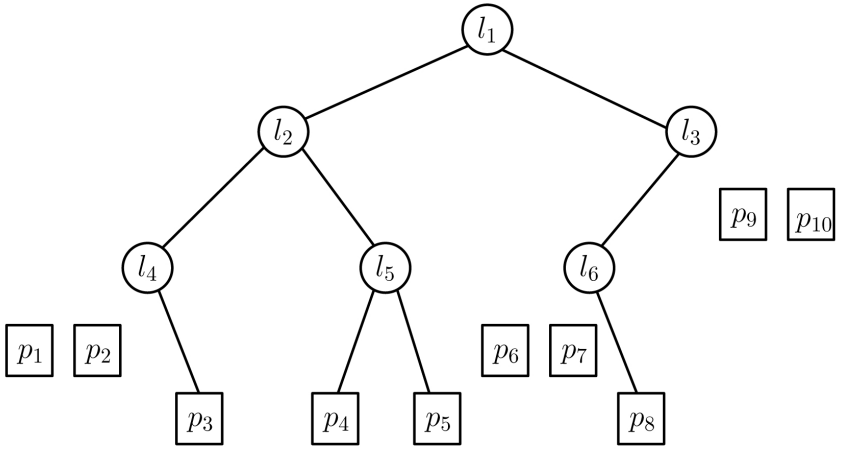
kd-Baum


l_5
l_1
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
p_1
p_8
p_3
l_4
p_6

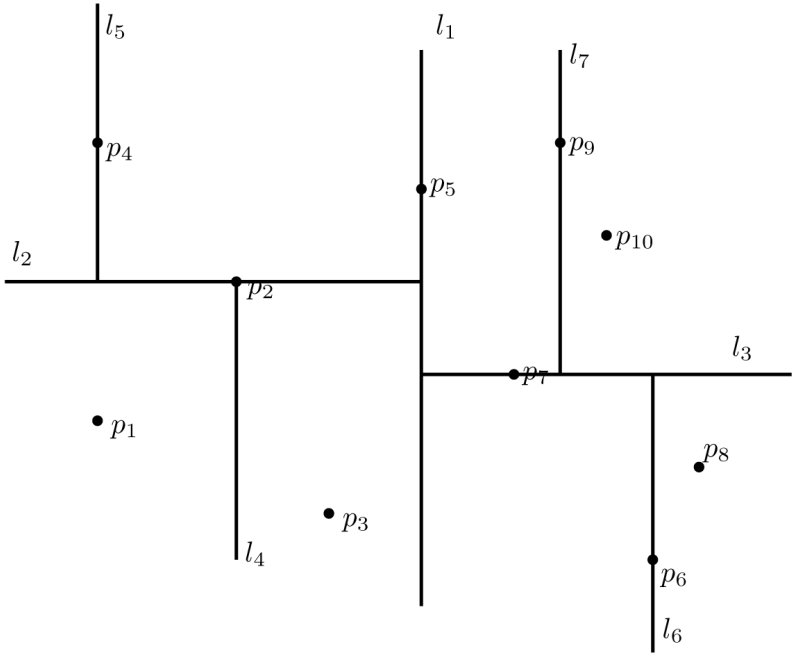

l_1
l_2
l_3
p_6
p_7
p_8
p_9
p_{10}
l_4
l_5
p_1
p_2
p_3
p_4
p_5

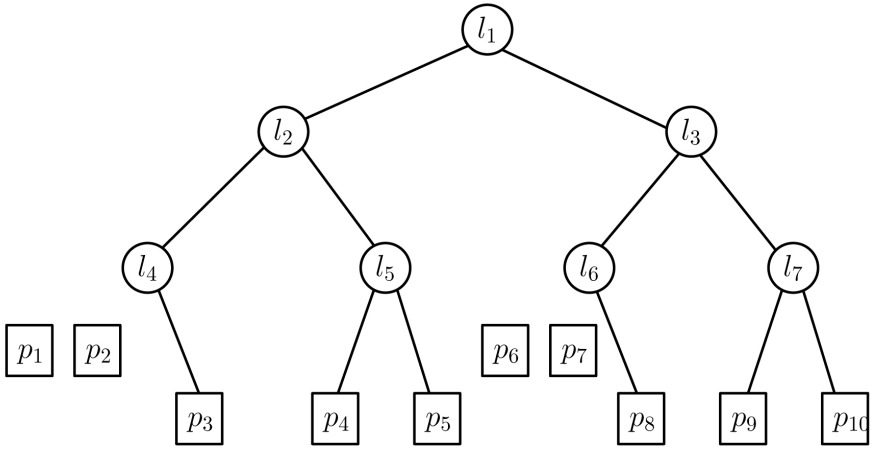
kd-Baum


l_5
l_1
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
p_1
p_8
p_3
l_4
p_6
l_6

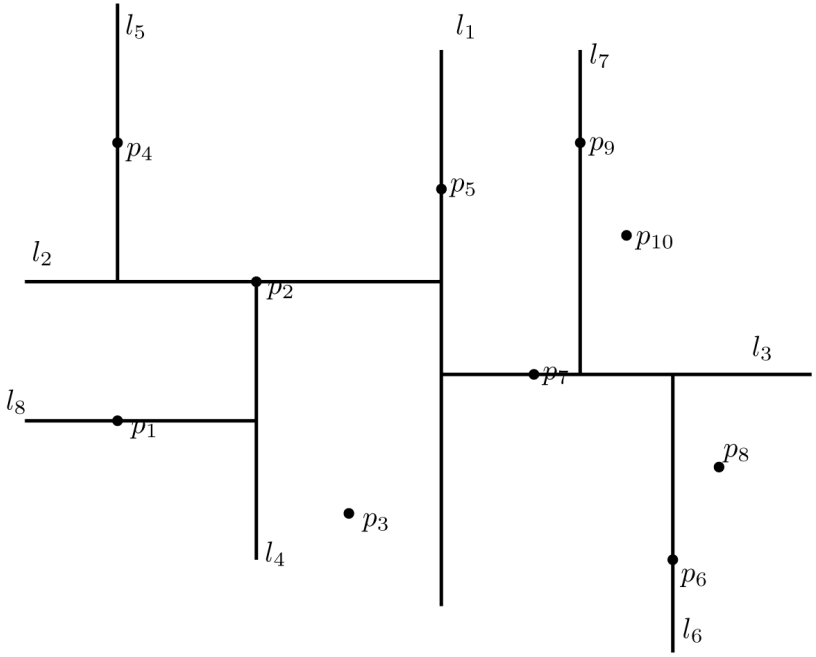

l_1
l_2
l_3
p_9
p_{10}
l_4
l_5
l_6
p_1
p_2
p_6
p_7
p_3
p_4
p_5
p_8

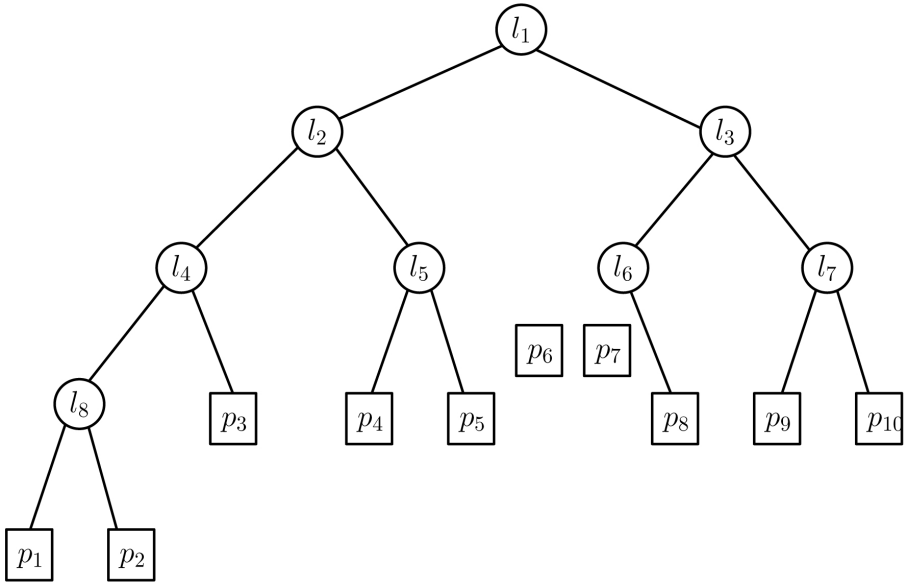
kd-Baum


l_5
l_1
l_7
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
p_1
p_8
p_3
l_4
p_6
l_6

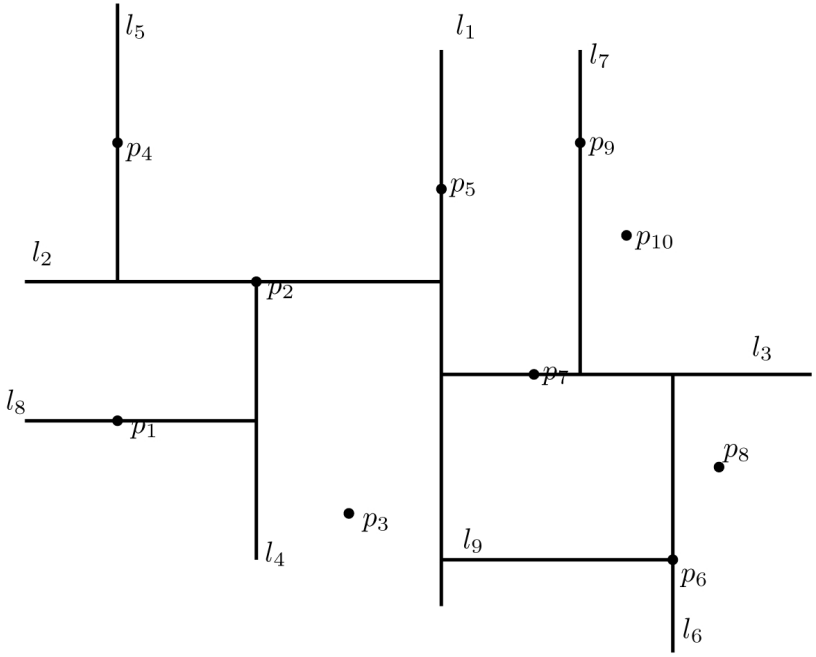

l_1
l_2
l_3
l_4
l_5
l_6
l_7
p_1
p_2
p_6
p_7
p_3
p_4
p_5
p_8
p_9
p_{10}

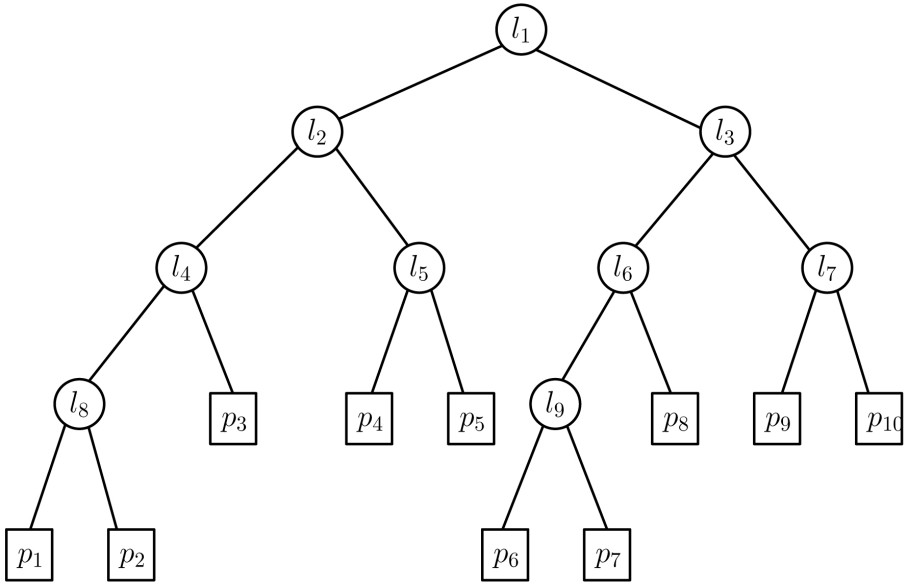
kd-Baum


l_5
l_1
l_7
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
l_8
p_1
p_8
p_3
l_4
p_6
l_6


l_1
l_2
l_3
l_4
l_5
l_6
l_7
p_6
p_7
l_8
p_3
p_4
p_5
p_8
p_9
p_{10}
p_1
p_2

kd-Baum


l_5
l_1
l_7
p_4
p_9
p_5
p_{10}
l_2
p_2
l_3
p_7
l_8
p_1
p_8
p_3
l_4
l_9
p_6
l_6


l_1
l_2
l_3
l_4
l_5
l_6
l_7
l_8
p_3
p_4
p_5
l_9
p_8
p_9
p_{10}
p_1
p_2
p_6
p_7

kd-Baum - Pseudocode

kdBaumBauen(P, *tiefe*)

if P hat nur einen Punkt **then**

return Knoten mit diesem Punkt

else

if Tiefe ist gerade **then**

Teile P mit einer vertikalen Linie l durch den Median der x-Koordinate in P_1 (links von oder auf l) und P_2 (rechts von l)

else

Teile P mit einer horizontalen Linie l durch den Median der y-Koordinate in P_1 (unter oder auf l) und P_2 (über l)

end if

$v_{links} \leftarrow kdBaumBauen(P_1, tiefe + 1)$

$v_{rechts} \leftarrow kdBaumBauen(P_2, tiefe + 1)$

Erstelle Knoten v, welches l enthält mit v_{left} als linkem und v_{right} als rechtem Kind von v

return v

kd-Baum - Pseudocode

kdBaumBauen(P, *tiefe*)

if P hat nur einen Punkt **then**

return Knoten mit diesem Punkt

else

if Tiefe ist gerade **then**

Teile P mit einer vertikalen Linie l durch den Median der x-Koordinate in P_1 (links von oder auf l) und P_2 (rechts von l)

else

Teile P mit einer horizontalen Linie l durch den Median der y-Koordinate in P_1 (unter oder auf l) und P_2 (über l)

end if

$v_{links} \leftarrow kdBaumBauen(P_1, tiefe + 1)$

$v_{rechts} \leftarrow kdBaumBauen(P_2, tiefe + 1)$

Erstelle Knoten v, welches l enthält mit v_{left} als linkem und v_{right} als rechtem Kind von v

return v

kd-Baum - Pseudocode

kdBaumBauen(P, *tiefe*)

if P hat nur einen Punkt **then**

return Knoten mit diesem Punkt

else

if Tiefe ist gerade **then**

Teile P mit einer vertikalen Linie l durch den Median der x-Koordinate in P_1 (links von oder auf l) und P_2 (rechts von l)

else

Teile P mit einer horizontalen Linie l durch den Median der y-Koordinate in P_1 (unter oder auf l) und P_2 (über l)

end if

$v_{links} \leftarrow kdBaumBauen(P_1, tiefe + 1)$

$v_{rechts} \leftarrow kdBaumBauen(P_2, tiefe + 1)$

Erstelle Knoten v, welches l enthält mit v_{left} als linkem und v_{right} als rechtem Kind von v

return v

kd-Baum - Pseudocode

kdBaumBauen$(P, tiefe)$

if P hat nur einen Punkt **then**

return Knoten mit diesem Punkt

else

if Tiefe ist gerade **then**

Teile P mit einer vertikalen Linie l durch den Median der x-Koordinate in P_1 (links von oder auf l) und P_2 (rechts von l)

else

Teile P mit einer horizontalen Linie l durch den Median der y-Koordinate in P_1 (unter oder auf l) und P_2 (über l)

end if

$v_{links} \leftarrow kdBaumBauen(P_1, tiefe + 1)$

$v_{rechts} \leftarrow kdBaumBauen(P_2, tiefe + 1)$

Erstelle Knoten v, welches l enthält mit v_{left} als linkem und v_{right} als rechtem Kind von v

return v

kd-Baum - Pseudocode

kdBaumBauen$(P, tiefe)$

if P hat nur einen Punkt **then**

return Knoten mit diesem Punkt

else

if Tiefe ist gerade **then**

Teile P mit einer vertikalen Linie l durch den Median der x-Koordinate in P_1 (links von oder auf l) und P_2 (rechts von l)

else

Teile P mit einer horizontalen Linie l durch den Median der y-Koordinate in P_1 (unter oder auf l) und P_2 (über l)

end if

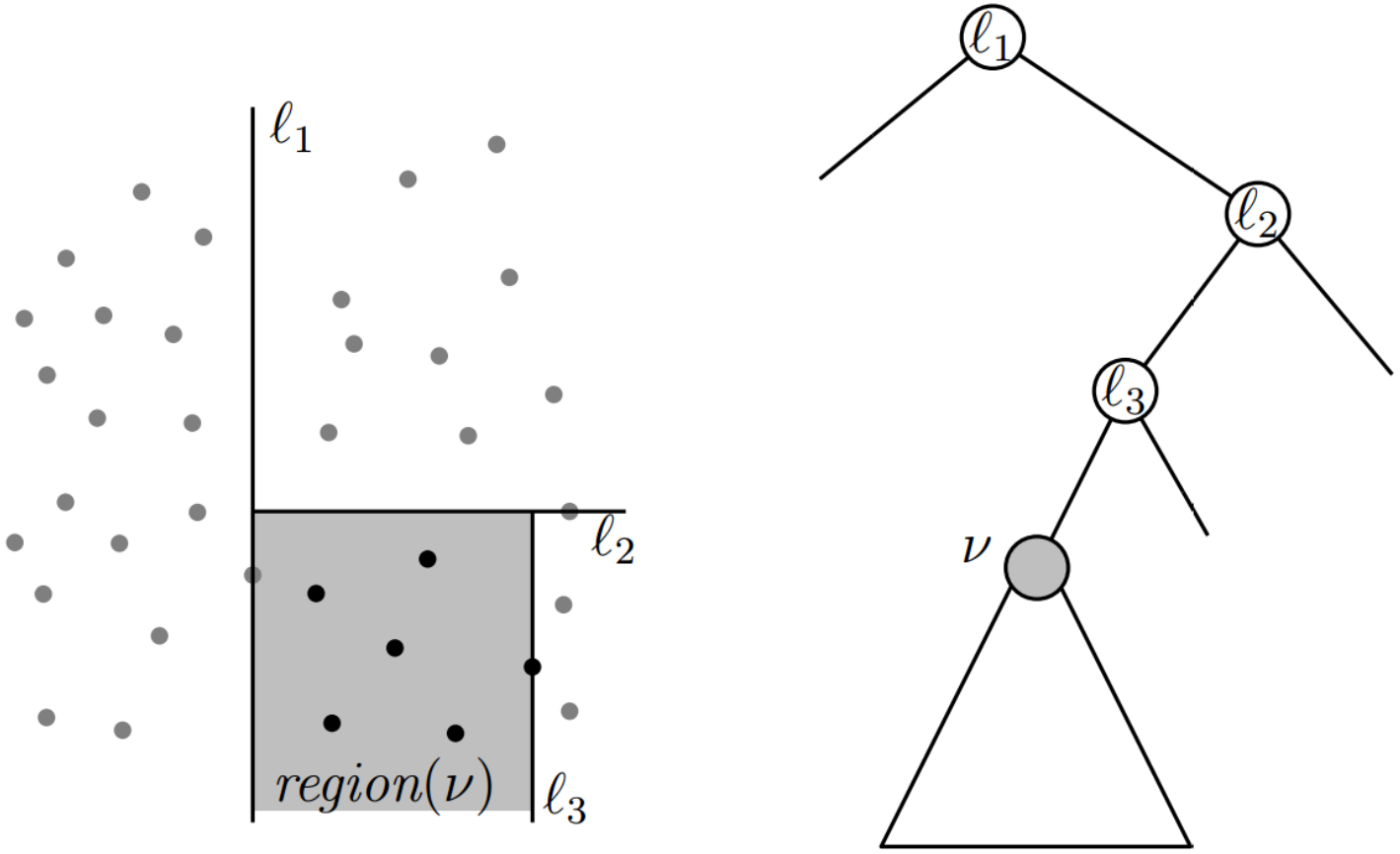
$v_{links} \leftarrow kdBaumBauen(P_1, tiefe + 1)$

$v_{rechts} \leftarrow kdBaumBauen(P_2, tiefe + 1)$

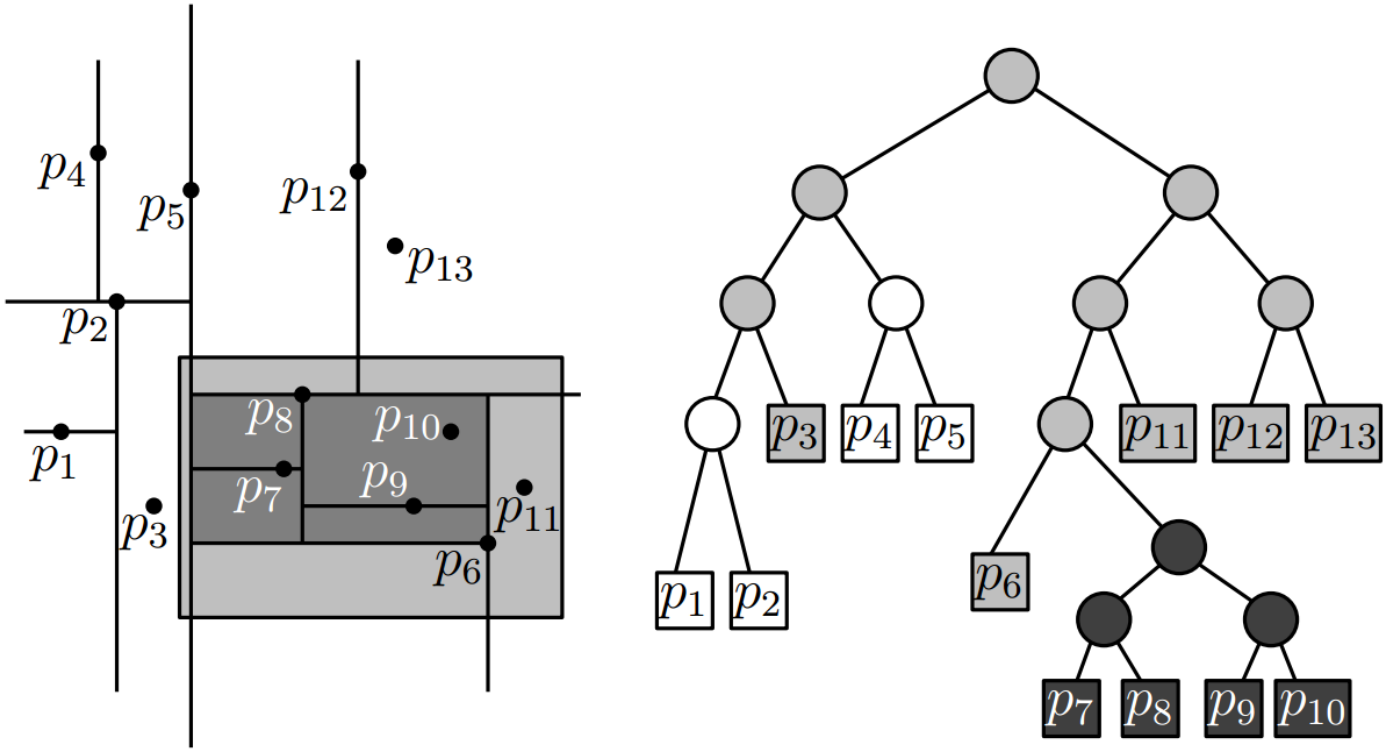
Erstelle Knoten v, welches l enthält mit v_{left} als linkem und v_{right} als rechtem Kind von v

return v

kd-Baum



kd-Baum


p_4
p_5
p_{12}
p_{13}
p_2
p_8
p_{10}
p_1
p_7
p_9
p_{11}
p_3
p_6
p_3 p_4 p_5 p_{11} p_{12} p_{13}
p_1 p_2
p_6
p_7 p_8 p_9 p_{10}

kd-Baum - Pseudocode

Seien v die Wurzel eines (Teil-)Baums und R der Bereich in dem gesucht wird.

SucheKdBaum(v, R)

if (**then** v ist ein Blatt)
    Melde Punkt in v, falls es in R liegt.
else
    if *region*($lc(v)$) ist vollständig in R **then**
        MeldeTeilbaum($lc(v)$)
    else
        if *region*($lc(v)$) überschneidet sich mit R **then**
            SucheKdBaum($lc(v)$, R)
        end if
    end if
    Das ganze dann nochmal äquivalent mit rc(v)
end if

kd-Baum - Pseudocode

Seien v die Wurzel eines (Teil-)Baums und R der Bereich in dem gesucht wird.

SucheKdBaum(v, R)

if (**then** v ist ein Blatt)
    Melde Punkt in v, falls es in R liegt.
else
    if *region*$(lc(v))$ ist vollständig in R **then**
        MeldeTeilbaum$(lc(v))$
    else
        if *region*$(lc(v))$ überschneidet sich mit R **then**
            SucheKdBaum$(lc(v), R)$
        end if
    end if
    Das ganze dann nochmal äquivalent mit rc(v)
end if

kd-Baum - Pseudocode

Seien v die Wurzel eines (Teil-)Baums und R der Bereich in dem gesucht wird.

SucheKdBaum(v, R)

if (**then** v ist ein Blatt)
    Melde Punkt in v, falls es in R liegt.
else
    if *region*($lc(v)$) ist vollständig in R **then**
        MeldeTeilbaum($lc(v)$)
    else
        if *region*($lc(v)$) überschneidet sich mit R **then**
            SucheKdBaum($lc(v)$, R)
        end if
    end if
    Das ganze dann nochmal äquivalent mit rc(v)
end if

kd-Baum - Pseudocode

Seien v die Wurzel eines (Teil-)Baums und R der Bereich in dem gesucht wird.

SucheKdBaum(v, R)

if (**then** v ist ein Blatt)
    Melde Punkt in v, falls es in R liegt.
else
    if *region*($lc(v)$) ist vollständig in R **then**
        MeldeTeilbaum($lc(v)$)
    else
        if *region*($lc(v)$) überschneidet sich mit R **then**
            SucheKdBaum($lc(v)$, R)
        end if
    end if
    Das ganze dann nochmal äquivalent mit rc(v)
end if

kd-Baum - Laufzeit

Kurz noch einige ungezeigte Behauptungen:

- Ein kd-Baum braucht $O(n)$ Speicher
- Ein kd-Baum lässt sich in $O(n * log(n))$ erstellen
- Eine Bereichssuche in einem kd-Baum liegt in $O(\sqrt{n} + k)$

Koordinatenkompression

Koordinatenkompression

- Abbilden von gegebenen Koordinaten auf geschicktere Koordinaten
- Was heißt geschickt?
  - Der Algorithmus soll dadurch schneller werden
  - Das Ergebnis darf nicht verfälscht werden
- Also: Irrelevante Bereiche aussparen

Koordinatenkompression

- Abbilden von gegebenen Koordinaten auf geschicktere Koordinaten
- Was heißt geschickt?
  - Der Algorithmus soll dadurch schneller werden
  - Das Ergebnis darf nicht verfälscht werden
- Also: Irrelevante Bereiche aussparen

Koordinatenkompression

- Abbilden von gegebenen Koordinaten auf geschicktere Koordinaten
- Was heißt geschickt?
  - Der Algorithmus soll dadurch schneller werden
  - Das Ergebnis darf nicht verfälscht werden
- Also: Irrelevante Bereiche aussparen

Koordinatenkompression

- Abbilden von gegebenen Koordinaten auf geschicktere Koordinaten
- Was heißt geschickt?
  - Der Algorithmus soll dadurch schneller werden
  - Das Ergebnis darf nicht verfälscht werden
- Also: Irrelevante Bereiche aussparen

Koordinatenkompression

- Abbilden von gegebenen Koordinaten auf geschicktere Koordinaten
- Was heißt geschickt?
  - Der Algorithmus soll dadurch schneller werden
  - Das Ergebnis darf nicht verfälscht werden
- Also: Irrelevante Bereiche aussparen

Flächenberechnung

Das Problem

- Eingabe:
  - Eine 2-dimensionale Figur
  - Bestehend aus Rechtecken
    - Punkt unten links
    - Punkt oben rechts
    - Seiten parallel zu den Koordinatenachsen
- Ausgabe:
  - Flächeninhalt der Figur

Flächenberechnung

Das Problem

- Eingabe:
  - Eine 2-dimensionale Figur
  - Bestehend aus Rechtecken
    - Punkt unten links
    - Punkt oben rechts
    - Seiten parallel zu den Koordinatenachsen
- Ausgabe:
  - Flächeninhalt der Figur

Flächenberechnung

Das Problem

- Eingabe:
  - Eine 2-dimensionale Figur
  - Bestehend aus Rechtecken
    - Punkt unten links
    - Punkt oben rechts
    - Seiten parallel zu den Koordinatenachsen
- Ausgabe:
  - Flächeninhalt der Figur

Flächenberechnung

Das Problem

- Eingabe:
  - Eine 2-dimensionale Figur
  - Bestehend aus Rechtecken
    - Punkt unten links
    - Punkt oben rechts
    - Seiten parallel zu den Koordinatenachsen
- Ausgabe:
  - Flächeninhalt der Figur

Flächenberechnung

Das Problem

- Eingabe:
  - Eine 2-dimensionale Figur
  - Bestehend aus Rechtecken
    - Punkt unten links
    - Punkt oben rechts
    - Seiten parallel zu den Koordinatenachsen
- Ausgabe:
  - Flächeninhalt der Figur

Flächenberechnung, erster Ansatz

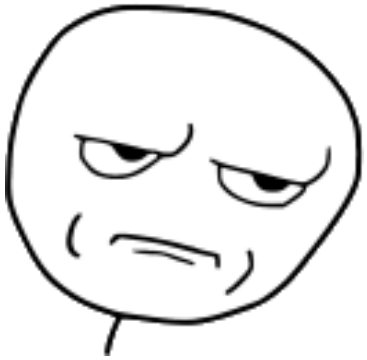
- Iteration über alle Datenfelder
- Aufsummieren aller belegter Datenfelder

Flächenberechnung, erster Ansatz

- Iteration über alle Datenfelder
- Aufsummieren aller belegter Datenfelder

Flächenberechnung, erster Ansatz

- Iteration über alle Datenfelder
- Aufsummieren aller belegter Datenfelder



Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Flächenberechnung mit Koordinatenkompression

- Vorarbeit
  - Kompression auf ein Feld maximal der Größe $(2 \cdot |Rechtecke|) \times (2 \cdot |Rechtecke|)$
  - Kleiner, falls sich Rechtecke eine x- oder y-Stelle teilen
  - Merken der tatsächlichen Größe
- Nun wie gehabt belegte Felder zählen
  - dabei deren tatsächliche Flächen aufsummieren

Sweep

Line Segment Intersection

Das Problem

- Eingabe:
  - n Liniensegmente
- Ausgabe:
  - Anzahl der Schnittpunkte

Line Segment Intersection

Das Problem

- Eingabe:
  - n Liniensegmente
- Ausgabe:
  - Anzahl der Schnittpunkte

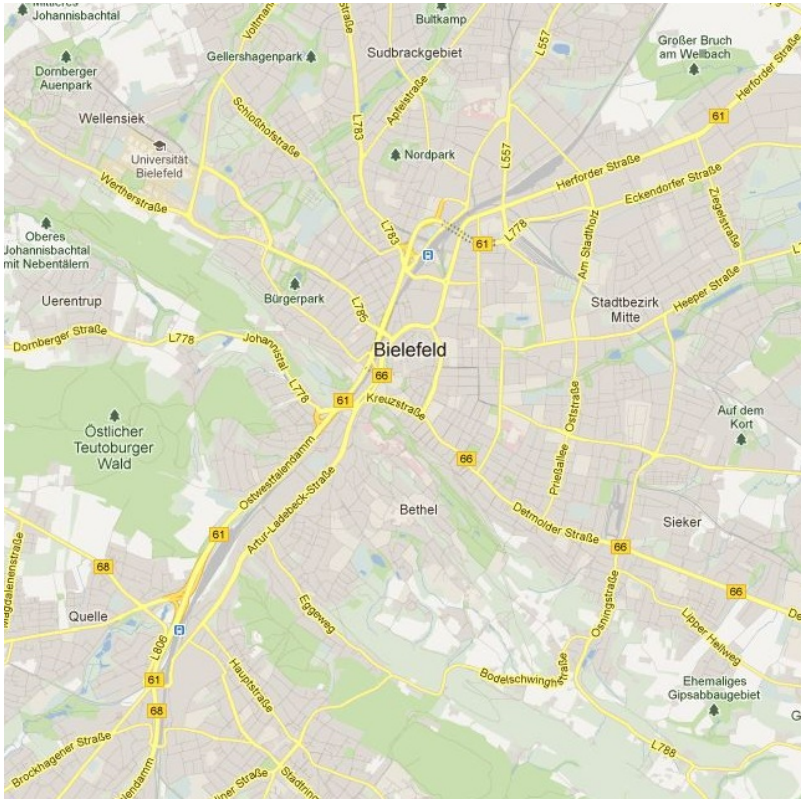
Line Segment Intersection

Das Problem

- Eingabe:
  - n Liniensegmente
- Ausgabe:
  - Anzahl der Schnittpunkte

Line Segment Intersection

- **Anwendungsbeispiel**
- Schnitt von Layern bei Kartographie-Software



Line Segment Intersection

- **Naiv**
  - Vergleiche jedes Segment mit jedem anderen
  - Resultierende Laufzeit: $\mathcal{O}(n^2)$

Line Segment Intersection

- **Naiv**
  - Vergleiche jedes Segment mit jedem anderen
  - Resultierende Laufzeit: $\mathcal{O}(n^2)$

Line Segment Intersection

- **Naiv**
  - Vergleiche jedes Segment mit jedem anderen
  - Resultierende Laufzeit: $\mathcal{O}(n^2)$

Line Segment Intersection

- **Weitere Überlegungen**
- Nur nahe Segmente können sich schneiden
- Betrachten der Situation an der Stelle a

Line Segment Intersection

- **Weitere Überlegungen**
- Nur nahe Segmente können sich schneiden
- Betrachten der Situation an der Stelle a

Line Segment Intersection

- **Weitere Überlegungen**
- Nur nahe Segmente können sich schneiden
- Betrachten der Situation an der Stelle a

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus, erster Ansatz**

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
- Event Queue Q
  - Endpunkte der Segmente, aufsteigend geordnet nach ihrer x-Koordinate

- Beginnt ein Segment
  - Füge das Segment dem Zustand hinzu
  - Überprüfe, ob es Segmente in S schneidet
- Endet ein Segment
  - Entferne das Segment aus dem Zustand
- Das geht noch besser!

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Line Segment Intersection

- **Sweep Line Algorithmus**
- Verlgeiche nur direkt benachbarte Segmente
- Neuer Eventpunkt:
  - Beim Schnitt zweier Segmente

Benötigte Daten

- Zustand der Sweep Line S
  - Segmente, welche die Sweep Line schneiden
  - Für jedes Segment die Nachbarn
- Event Queue Q
  - Endpunkte der Segmente
  - Schnittpunkte zwischen Segmenten

- Bentley Ottmann Algorithmus
  - Laufzeit $\mathcal{O}((n+k) \cdot log(n))$

Allgemeine Eigenschaften von Sweep Algorithmen

- Anwendbar auf Probleme beliebig hoher Dimension
  - Im 2-Dimensionalen Raum
    - Gerade, die über sich über die Ebene bewegt
  - Im 3-Dimensionalen Raum
    - Ebene, die sich durch den Raum bewegt

Allgemeine Eigenschaften von Sweep Algorithmen

- Anwendbar auf Probleme beliebig hoher Dimension
  - Im 2-Dimensionalen Raum
    - Gerade, die über sich über die Ebene bewegt
  - Im 3-Dimensionalen Raum
    - Ebene, die sich durch den Raum bewegt

Allgemeine Eigenschaften von Sweep Algorithmen

- Anwendbar auf Probleme beliebig hoher Dimension
  - Im 2-Dimensionalen Raum
    - Gerade, die über sich über die Ebene bewegt
  - Im 3-Dimensionalen Raum
    - Ebene, die sich durch den Raum bewegt

Weitere Sweep Algorithmen

- Fortune's algorithm
  - Sweep Line Algorithmus zur Erstellung eines Voronoi-Diagramms
  - Mögliches Anwendungsgebiet: Robotik, Hindernissen fern bleiben
- Boolsche Operationen auf Polygone
  - Mögliche Anwendungsgebiete: Computergraphik oder CAD-Systeme
- Closest Pair
- Viele mehr!

Weitere Sweep Algorithmen

- Fortune's algorithm
  - Sweep Line Algorithmus zur Erstellung eines Voronoi-Diagramms
  - Mögliches Anwendungsgebiet: Robotik, Hindernissen fern bleiben
- Boolsche Operationen auf Polygone
  - Mögliche Anwendungsgebiete: Computergraphik oder CAD-Systeme
- Closest Pair
- Viele mehr!

Weitere Sweep Algorithmen

- Fortune's algorithm
  - Sweep Line Algorithmus zur Erstellung eines Voronoi-Diagramms
  - Mögliches Anwendungsgebiet: Robotik, Hindernissen fern bleiben
- Boolsche Operationen auf Polygone
  - Mögliche Anwendungsgebiete: Computergraphik oder CAD-Systeme
- Closest Pair
- Viele mehr!

Weitere Sweep Algorithmen

- Fortune's algorithm
  - Sweep Line Algorithmus zur Erstellung eines Voronoi-Diagramms
  - Mögliches Anwendungsgebiet: Robotik, Hindernissen fern bleiben
- Boolsche Operationen auf Polygone
  - Mögliche Anwendungsgebiete: Computergraphik oder CAD-Systeme
- Closest Pair
- Viele mehr!

Weitere Sweep Algorithmen

- Fortune's algorithm
  - Sweep Line Algorithmus zur Erstellung eines Voronoi-Diagramms
  - Mögliches Anwendungsgebiet: Robotik, Hindernissen fern bleiben
- Boolsche Operationen auf Polygone
  - Mögliche Anwendungsgebiete: Computergraphik oder CAD-Systeme
- Closest Pair
- Viele mehr!

Weitere Sweep Algorithmen

- Fortune's algorithm
  - Sweep Line Algorithmus zur Erstellung eines Voronoi-Diagramms
  - Mögliches Anwendungsgebiet: Robotik, Hindernissen fern bleiben
- Boolsche Operationen auf Polygone
  - Mögliche Anwendungsgebiete: Computergraphik oder CAD-Systeme
- Closest Pair
- Viele mehr!