Algorithmen II

Peter Sanders

Übungen:

Moritz Laupichler, Nikolai Maas

Institut für Theoretische Informatik

Web:

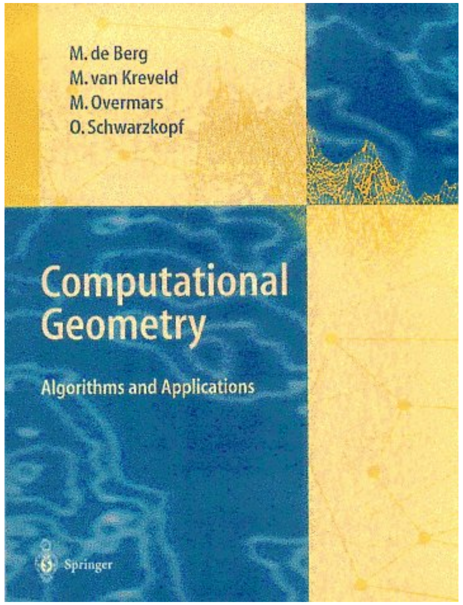
`algo2.iti.kit.edu/AlgorithmenII_WS23.php`

12 Geometrische Algorithmen

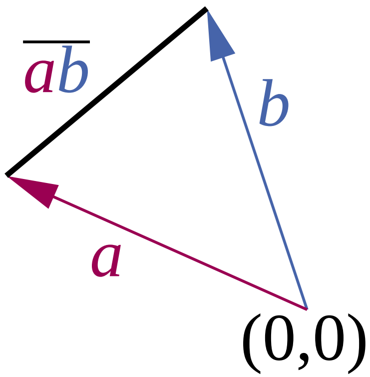
- Womit beschäftigen sich geom. Algorithmen?
- Schnitt von Strecken: Bentley-Ottmann-Algorithmus
- Konvexe Hüllen
- Kleinste einschließende Kugel
- Range Search

Quelle:

[Computational Geometry – Algorithms and Applications
de Berg, van Kreveld, Overmars, Schwartzkopf
Springer, 1997]



Elementare geometrische Objekte



Punkte: $x \in \mathbb{R}^d$

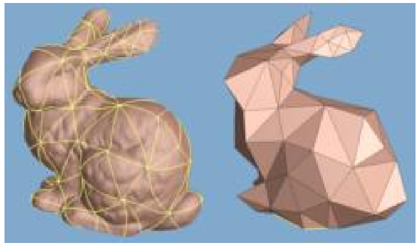
Strecken: $\overline{ab} := \{\alpha a + (1-\alpha) b : \alpha \in [0,1]\}$

uvam: Halbräume, Ebenen, Kurven,...

Dimension d:

1: Oft trivial. Gilt i. allg. nicht als geometrisches Problem

2: Geogr. Informationssysteme (GIS), Bildverarbeitung,...



3: Computergrafik, Simulationen,...

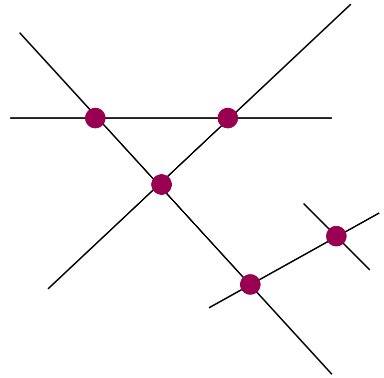
≥ 4: Optimierung, Datenbanken, maschinelles Lernen,...

curse of dimensionality!

n: Anzahl vorliegender Objekte

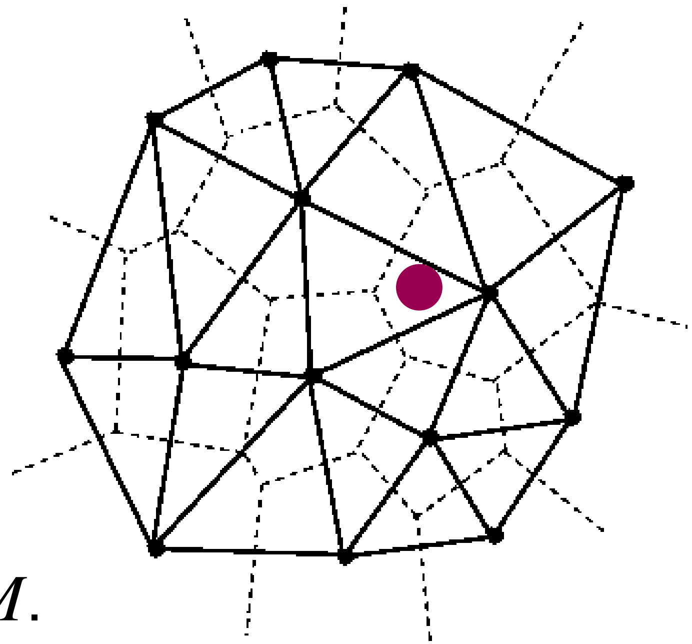
Typische Fragestellungen

- Schnittpunkte zwischen n Strecken



Typische Fragestellungen

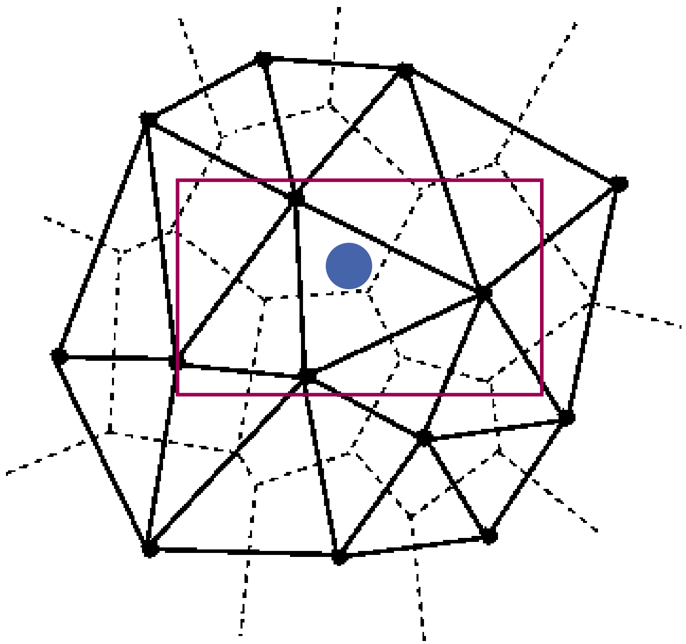
- Schnittpunkte zwischen n Strecken
- Konvexe Hülle
- Triangulation von Punktmengen
  (2D, verallgemeinerbar)
  z.B. Delaunaytriangulierung:
  Kein Dreieck enthält weiteren Punkt



- Voronoi-Diagramme: Sei $x \in M \subseteq \mathbb{R}^d$.
  $\forall y \in \mathbb{R}^d$ bestimme nächstes Element aus M.
  (Unterteilung von M in n Voronoizellen)
- Punklokalisierung: Geg. Unterteilung von R^d, $x \in R^d$:
  in welchem Teil liegt x ?

Datenstrukturen für Punktmengen

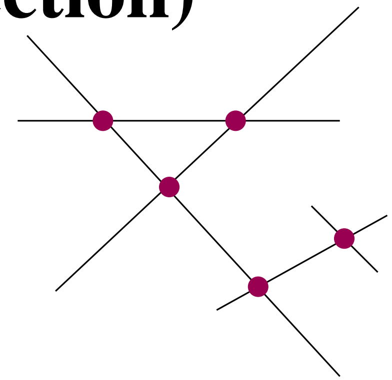
- nächsten Nachbarn berechnen
- Bereichsanfragen
- ...



Mehr Fragestellungen

- Sichtbarkeitsberechnungen
- Lineare Programmierung
- Geometrische Versionen von Optimierungsproblemen
  - Kürzeste Wege, z.B.
    energieeffiziente Kommunikation in Radionetzwerken
  - minimale Spannbäume
    reduzierbar auf Delaunay-Triangulierung + Graphalgorithmus
  - Matchings
  - Handlungsreisendenproblem
  - ...
- ...

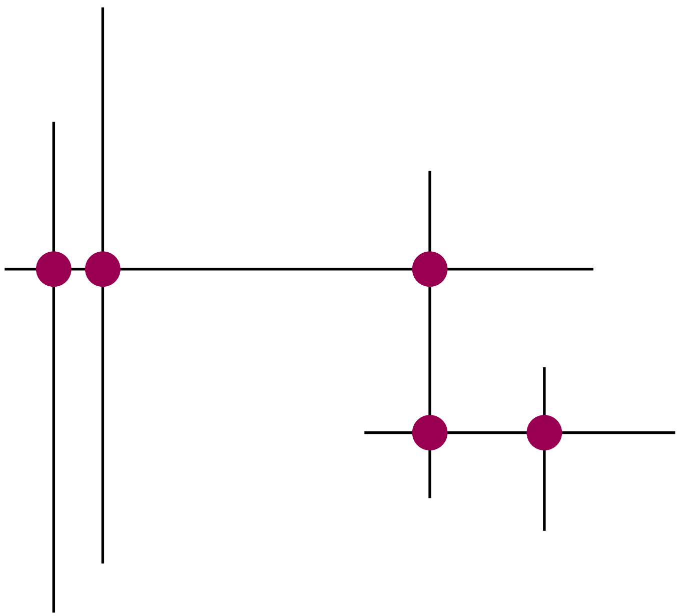
12.1 Streckenschnitt (line segment intersection)



Gegeben: $S=\{s_1,\ldots,s_n\}$, n Strecken

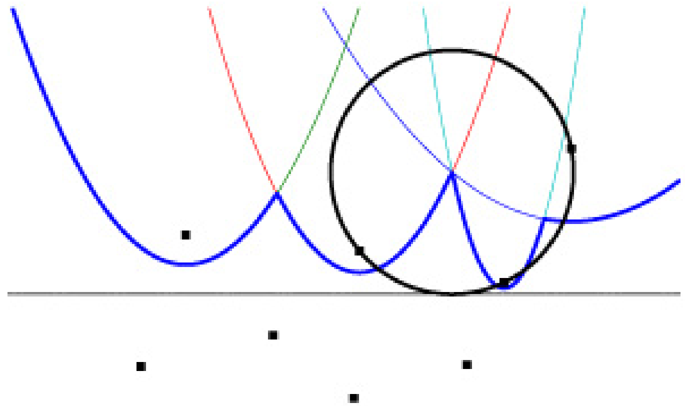
Gesucht: Schnittpunkte $\bigcup_{s,t\in S} s\cap t$

Zum Warmwerden: Orthogonaler Streckenschnitt – die Strecken sind parallel zur x- oder y-Achse



Streckenschnitt: Anwendungen

- Schaltungsentwurf: wo kreuzen sich Leiterbahnen?
- GIS: Strassenkreuzungen, Brücken,. . .
- Erweiterungen: z.B. Graphen benachbarter Strecken/Flächen aufbauen/verarbeiten
- Noch allgemeiner: Plane-Sweep-Algorithmen für andere Fragestellungen (z.B. Konstruktion von konvexen Hüllen oder Voronoidiagrammen)

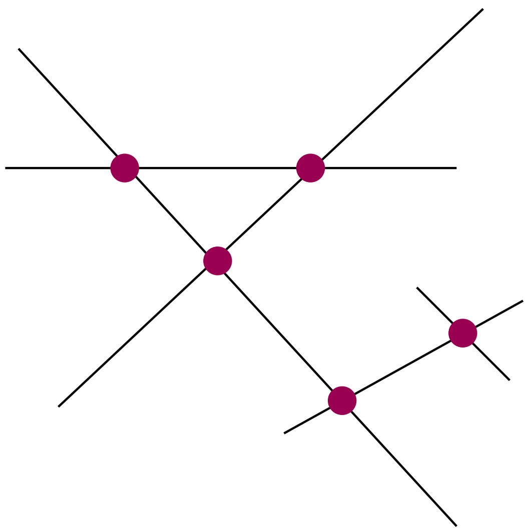


Streckenschnitt: Naiver Algorithmus

foreach $\{s,t\} \subseteq S$ **do**
 if $s \cap t \neq \emptyset$ **then**
  output $\{s,t\}$

Problem: Laufzeit $\Theta\left(n^2\right)$.

Zu langsam für große Datenmengen

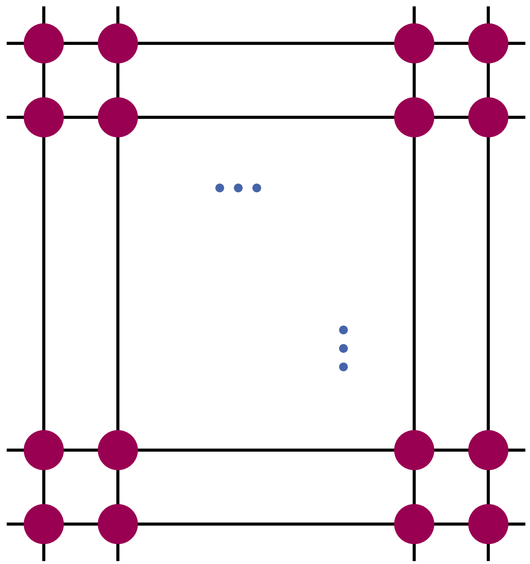


Streckenschnitt: Untere Schranke

$\Omega(n+k)$ mit k:= Anzahl ausgegebener Schnitte.

Vergleichsbasiert: $\Omega(n\log n+k)$ (Beweis: nicht hier)

Beobachtung $k=\Theta\left(n^2\right)$ ist möglich, aber reale Eingaben haben meist $k=\mathrm{O}(n)$.

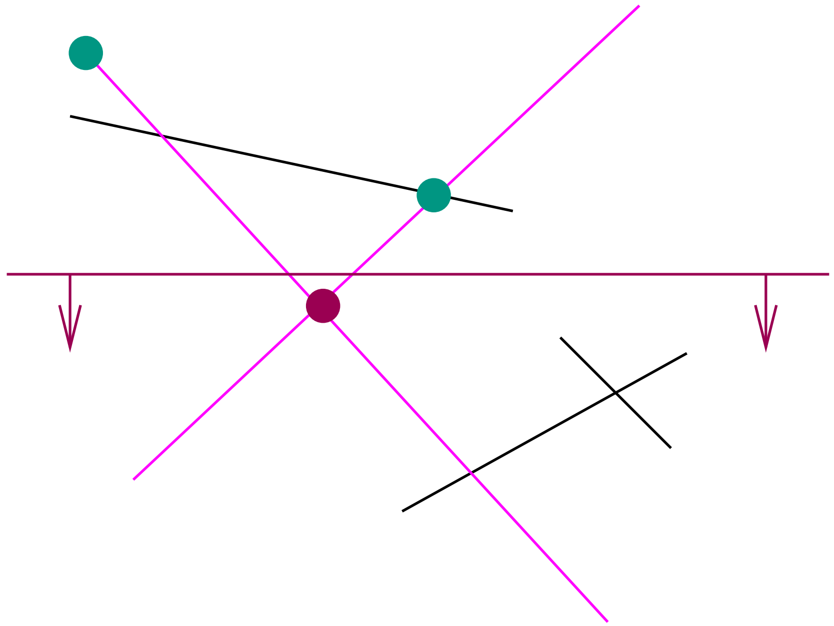


Idee: Plane-Sweep-Algorithmen

(Waagereche) Sweep-Line ℓ läuft von oben nach unten.

Invariante: Schnittpunkte oberhalb von ℓ wurden korrekt ausgegeben.

Ansatz: speichere jeweils Segmente, die ℓ schneiden und finde deren Schnittpunkte.



Plane-Sweep für orth. Streckenschnitt

Erstmal nur: Schnitte zwischen vertikalen und horizontalen Strecken.

$T = \langle\rangle$: SortedSequence **of** Segment
invariant T stores the vertical segments intersecting ℓ
Q:= sort($\langle(y,s)$: ∃hor. seg. s at y or ∃vert. seg. s starting/ending at $y\rangle$)
    // tie breaking: vert. starting events first, vert. finishing events last
foreach $(y,s) \in Q$ in descending order **do**
    if s is a vertical segment and starts at y **then** T.insert(s)
    else if s is a vertical segment and ends at y **then** T.remove(s)
    else // we have a horizontal segment $s = \overline{(x_1,y)(x_2,y)}$
        foreach $t = \overline{(x,y_1)(x,y_2)} \in T$ with $x \in [x_1,x_2]$ **do**
            output $\{s,t\}$

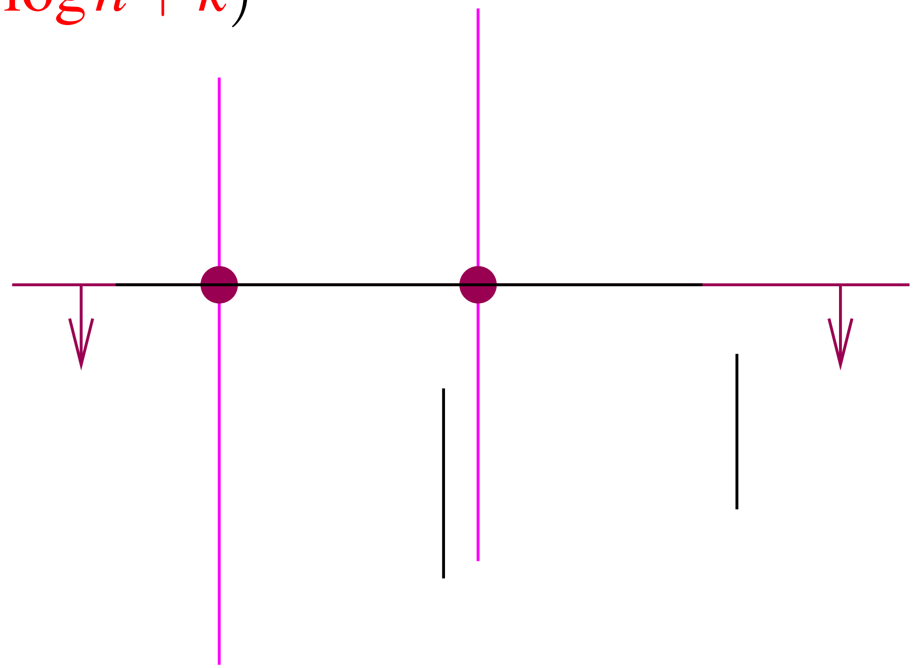
Analyse orth. Streckenschnitt

insert: $O(\log n)$ ($\leq n\times$)

remove: $O(\log n)$ ($\leq n\times$)

rangeQuery: $O(\log n + k_s)$, k_s Schnitte mit hor. Segment s

Insgesamt: $O(n \log n + \sum_s k_s) = O(n \log n + k)$



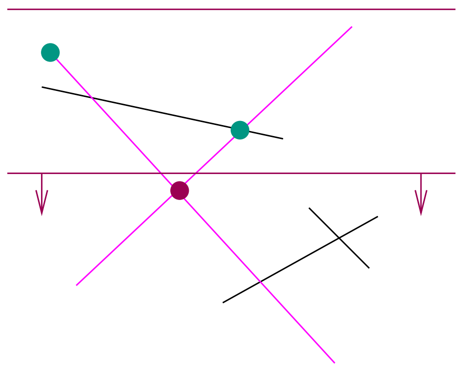
Verallgemeinerung – aber erstmal “nicht ganz”

Annahme zunächst:

Allgemeine Lage, d.h. hier

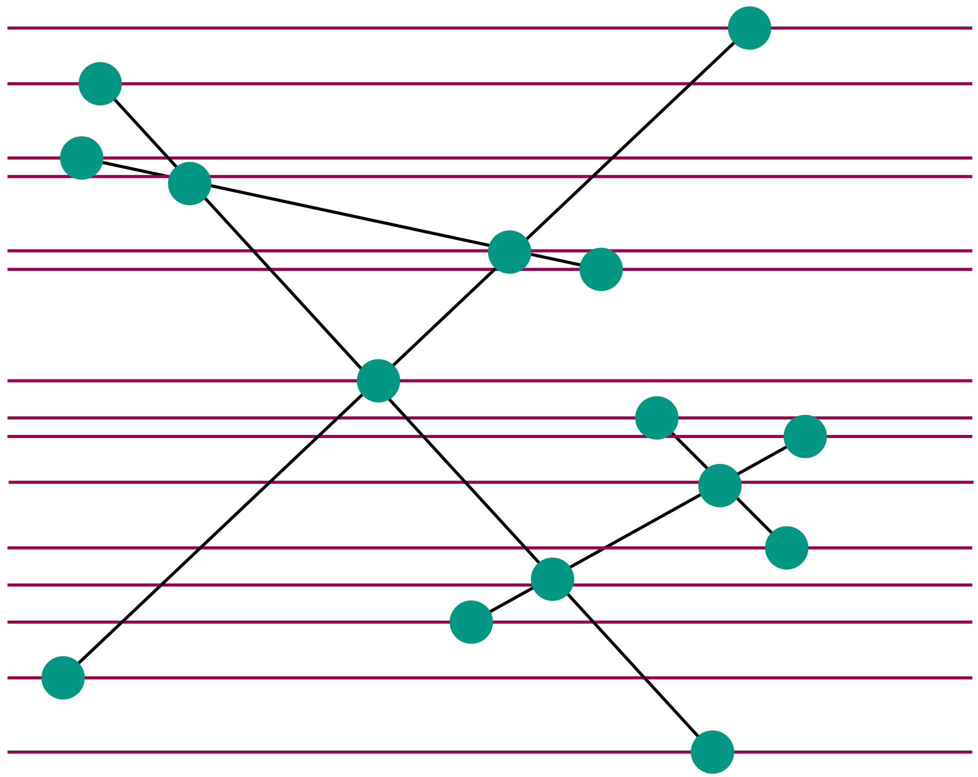
- Keine horizontalen Strecken
- Keine Überlappungen
- Schnittpunkte jeweils im Inneren von genau zwei Strecken

Beobachtung: kleine zuf. Perturbationen produzieren allg. Lage.



Verallgemeinerung – Grundidee

- Plane-Sweep mit Sweep-Line ℓ
- Status T := nach x geordnete Folge der ℓ schneidenden Strecken
- Ereignis:= Statusänderung
  - Startpunkte
  - Endpunkte
  - Schnittpunkte
- Schnitttest nur für Segmente, die an einem Ereignispunkt in T benachbart sind.



Verallgemeinerung – Korrektheit

Lemma:

$s \cap t = \{(x,y)\} \longrightarrow \exists$ Ereignis : s,t werden Nachbarn auf ℓ

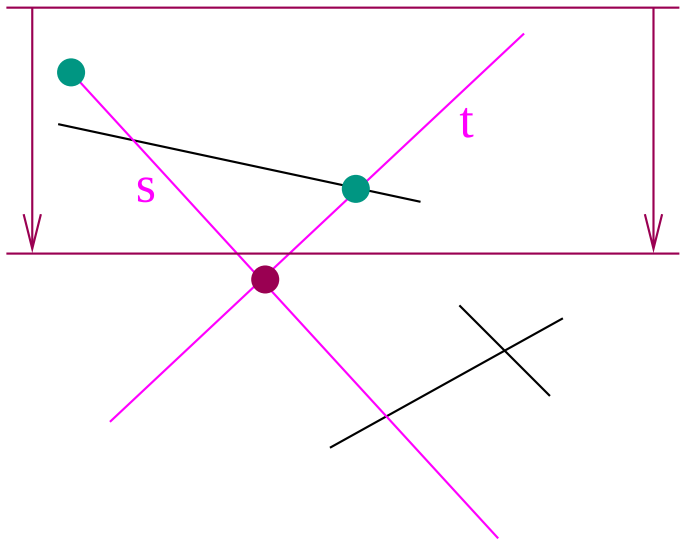
Beweis:

Anfangs : $T = \langle\rangle \longrightarrow s,t$ sind nicht in ℓ benachbart.

$@y+\varepsilon$: s,t sind in ℓ benachbart.

$\longrightarrow$

$\exists$ Ereignis bei dem s und t Nachbarn werden.



Verallgemeinerung – Implementierung

$T = \langle\rangle$: SortedSequence **of** Segment

invariant T stores the relative order of the segments intersecting ℓ

Q : MaxPriorityQueue

$Q := Q \cup \left\{ (\max\{y, y'\}, \text{start}, s) : s = \overline{(x,y)(x',y')} \in S \right\}$ // $\mathrm{O}(n \log n)$

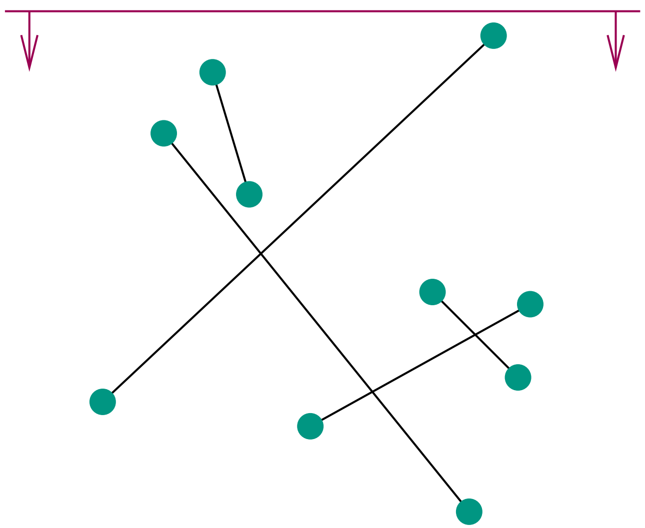
$Q := Q \cup \left\{ (\min\{y, y'\}, \text{finish}, s) : s = \overline{(x,y)(x',y')} \in S \right\}$ // $\mathrm{O}(n \log n)$

while $Q \neq \emptyset$ **do**

  $(y, \text{type}, s) := Q.\text{deleteMax}$

  // $\mathrm{O}((n+k)\log n)$

  handleEvent$(y, \text{type}, s, T, Q)$



handleEvent$(y, \text{start}, s, T, Q)$ // $n\times$

$h{:=}\ T$.insert(s) // $O(\log n)$

prev:= pred(h) // $O(1)$

next:= succ(h) // $O(1)$

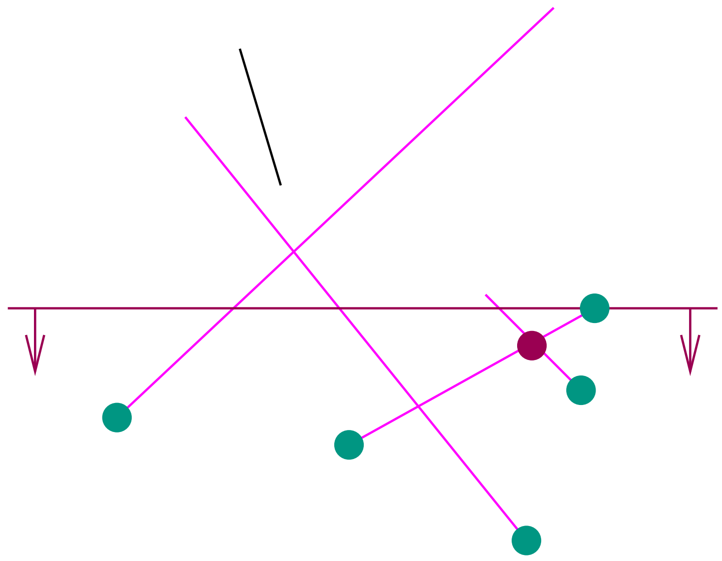
findNewEvent(prev, h)

findNewEvent(h, next)

Procedure findNewEvent(s, t) // $O(1 + \log n)$

if $*s$ and $*t$ intersect at $y' < y$ **then**

Q.insert$((y', \text{intersection}, (s, t)))$



handleEvent$(y,$ finish$, s, T, Q)$ // $n\times$

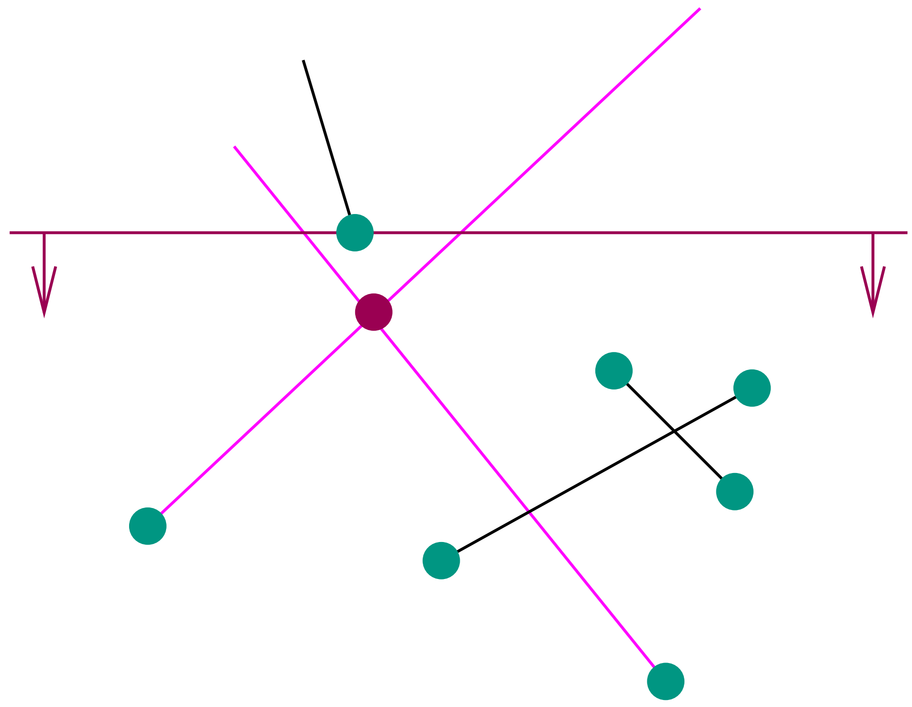
h:= T.locate(s) // $O(\log n)$

prev:= pred(h) // $O(1)$

next:= succ(h) // $O(1)$

T.remove(s) // $O(\log n)$

findNewEvent(prev, next)



handleEvent$(y,$ intersection, $(a,b),T,Q)$ // $k\times$

output $(*s \cap *t)$ // $\mathrm{O}(1)$

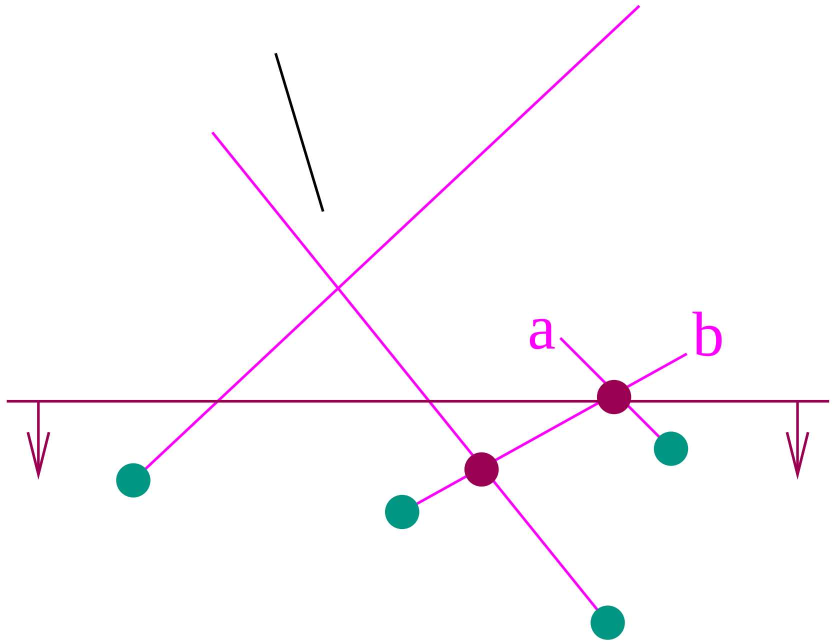
T.swap(a,b) // $\mathrm{O}(\log n)$

prev:= pred(b) // $\mathrm{O}(1)$

next:= succ(a) // $\mathrm{O}(1)$

findNewEvent(prev, b)

findNewEvent(a, next)



Implementierungsdetail: Sortierte Folge

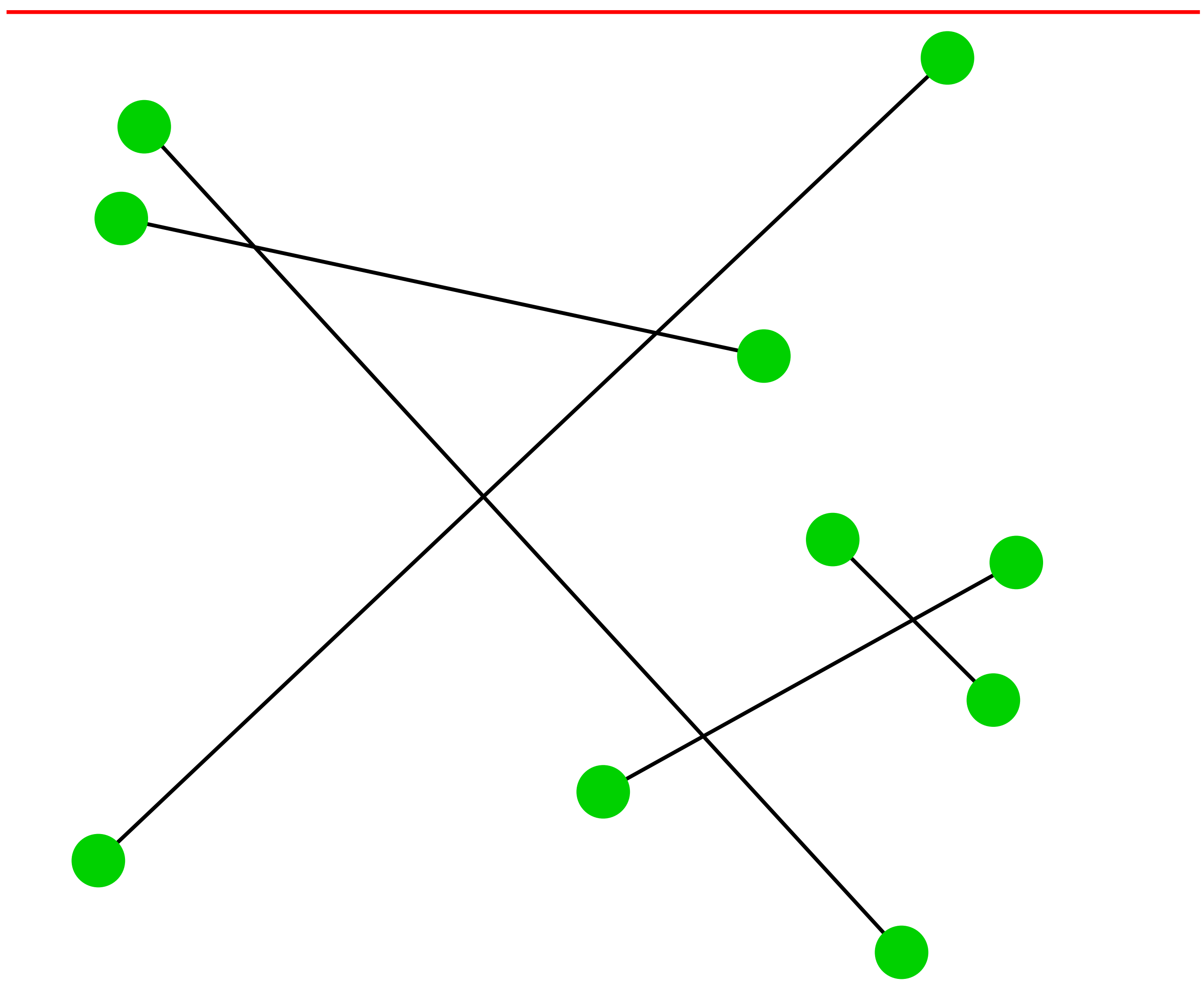
Schlüssel ist die x-Koordinate des Schnittpunkte mit der Sweep-Line.

Diese verändert sich ständig!

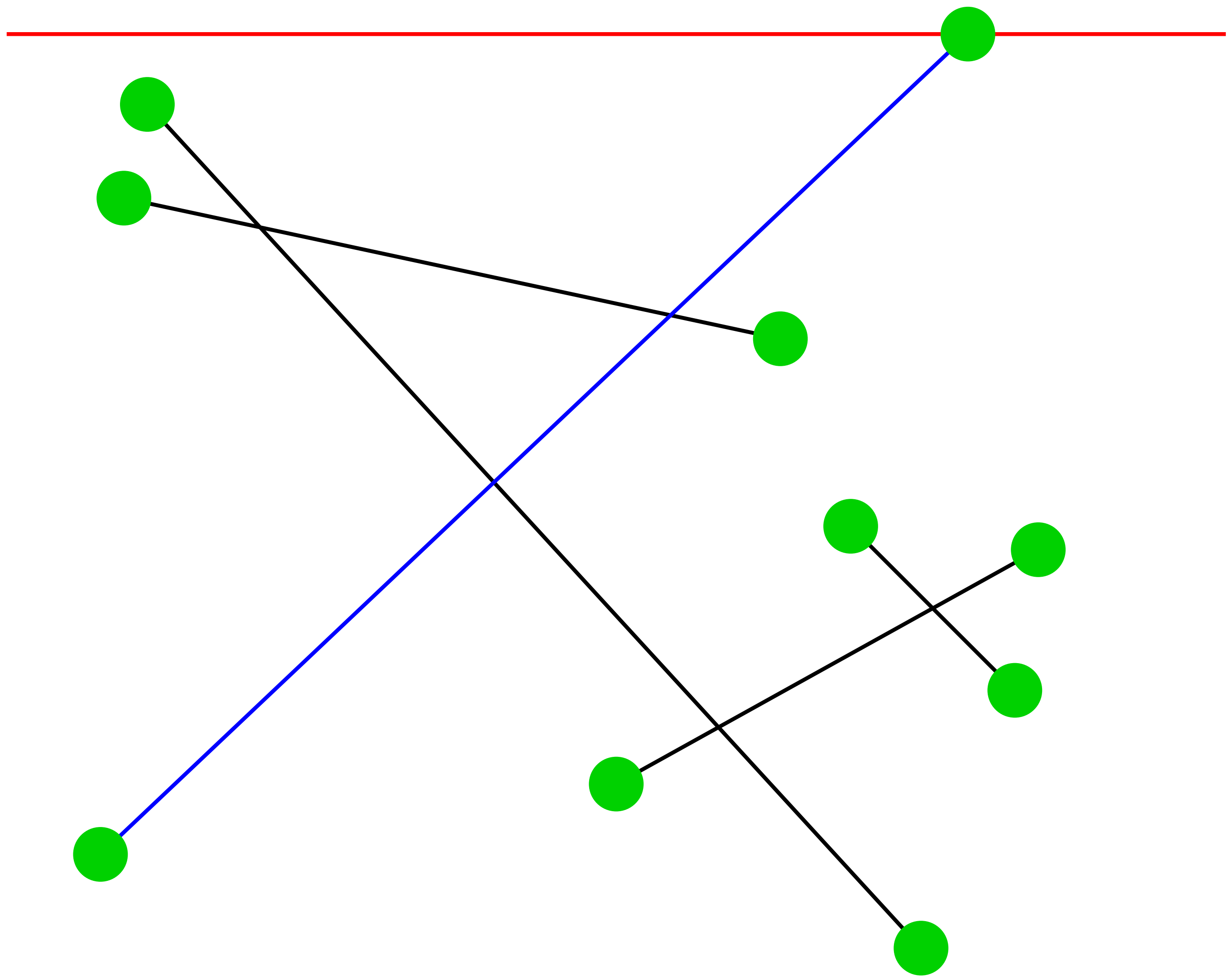
Aber: zwischen Events bleibt die Reihenfolge der Elemente gleich.

Alles funktioniert solange ein vergleichsbasierter Suchbaum verwendet wird.

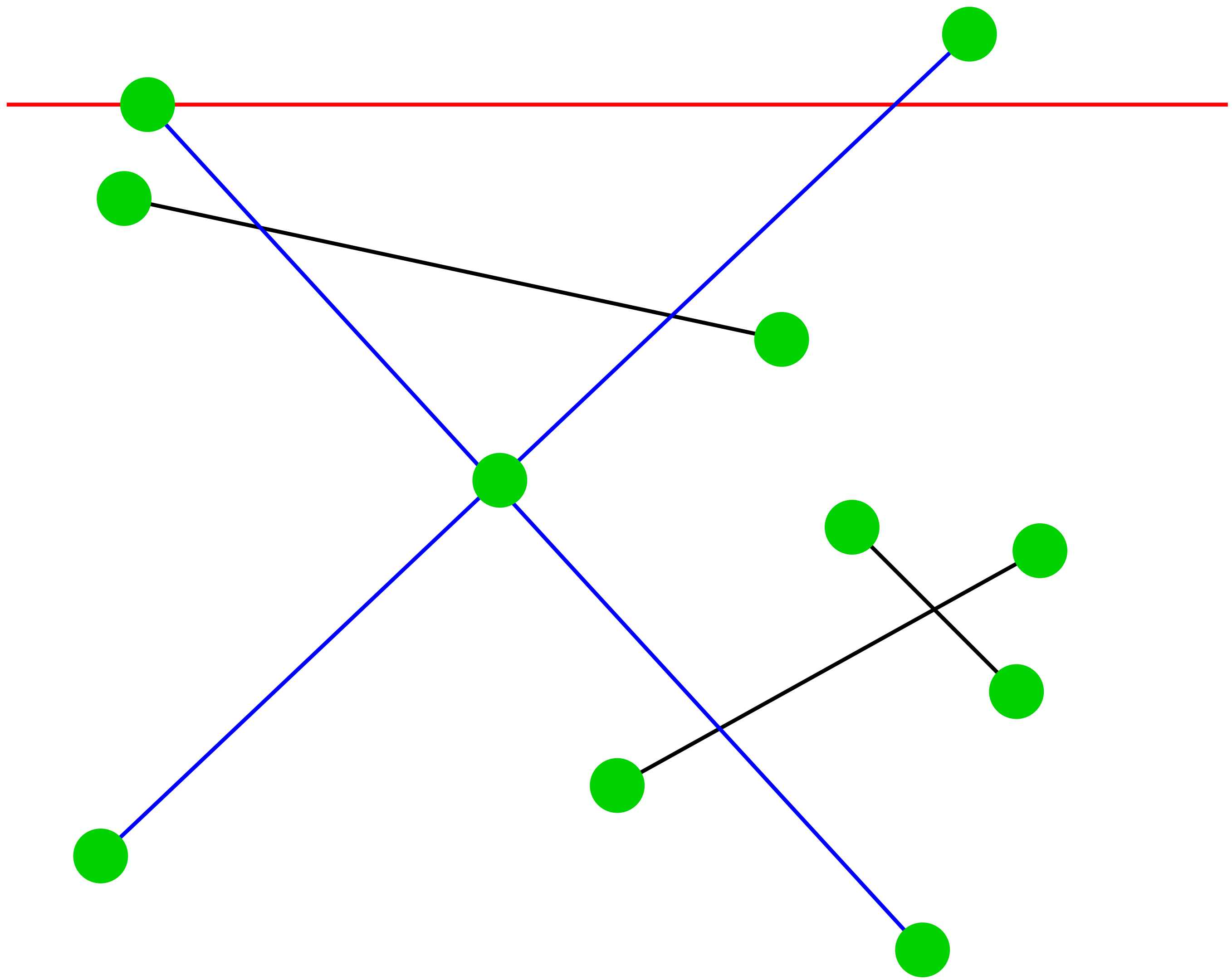
Verallgemeinerung – Beispiel



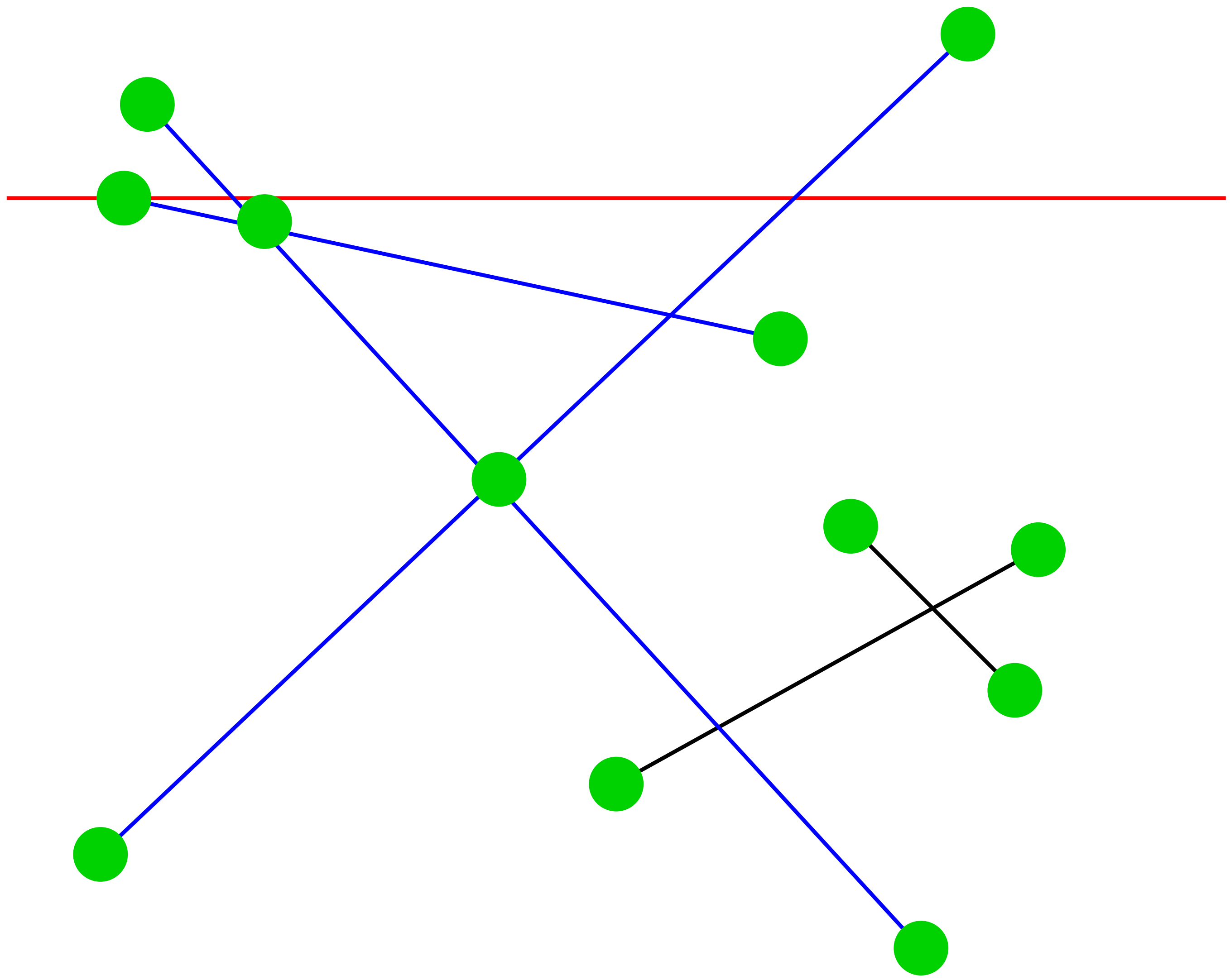
Verallgemeinerung – Beispiel



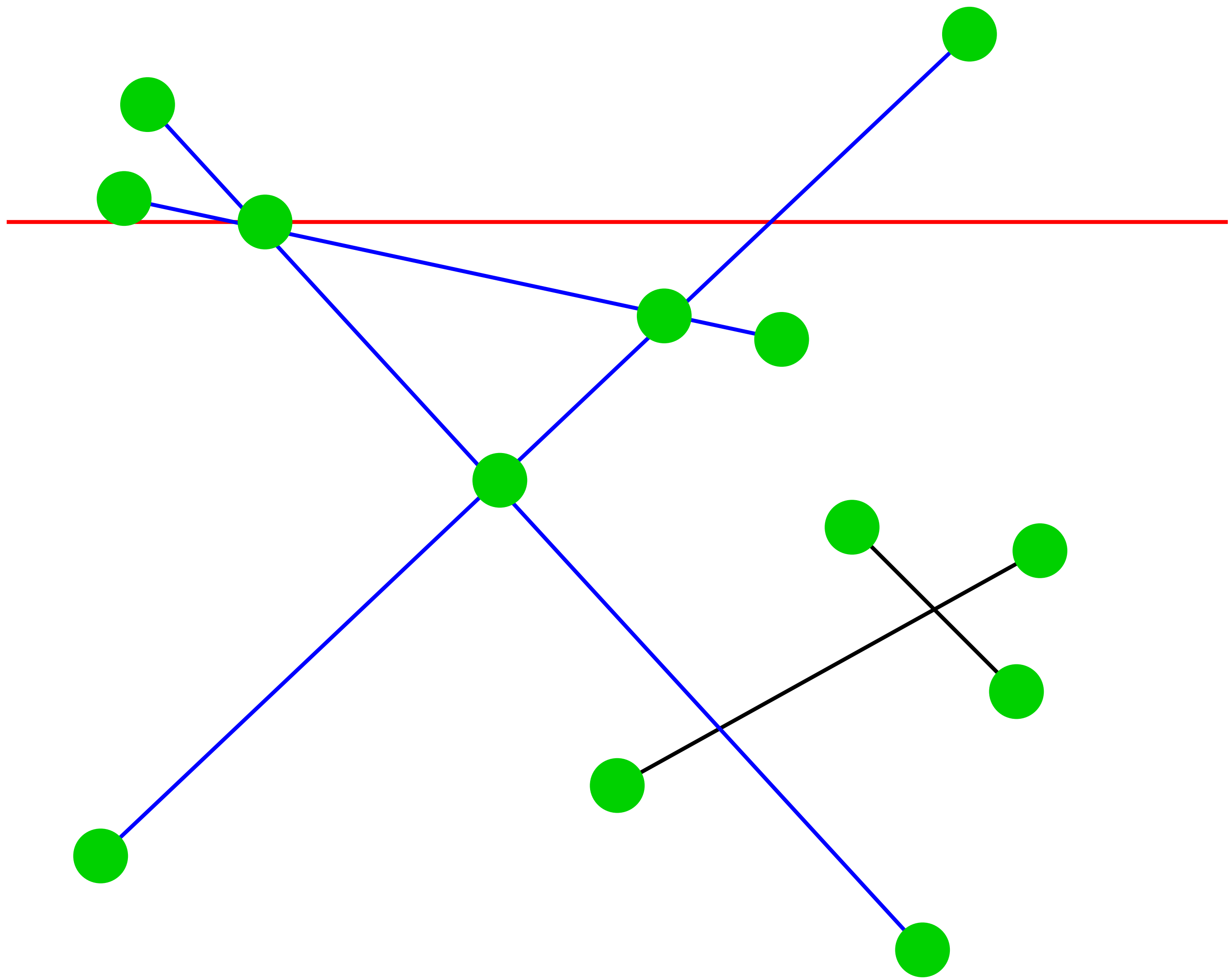
Verallgemeinerung – Beispiel



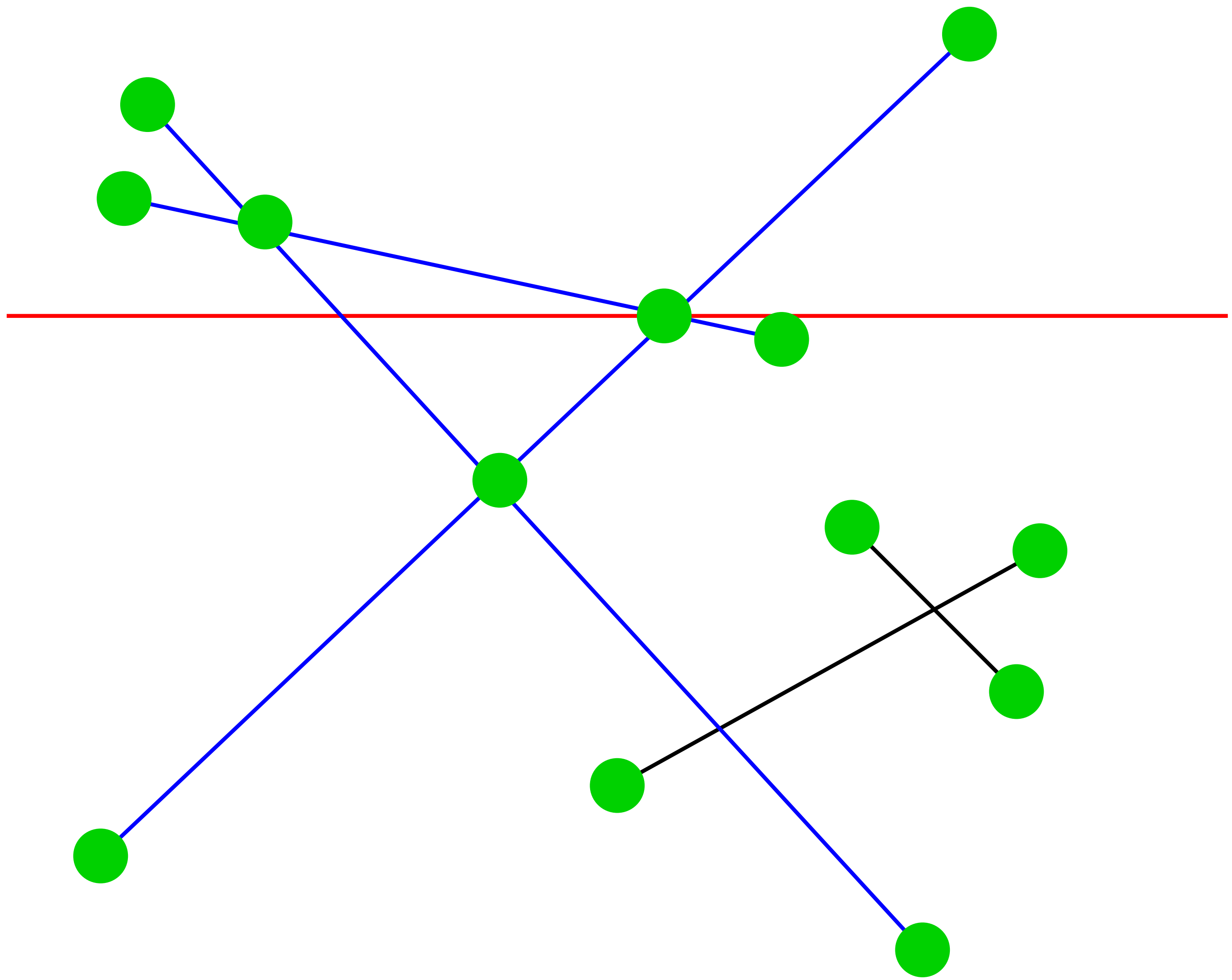
Verallgemeinerung – Beispiel



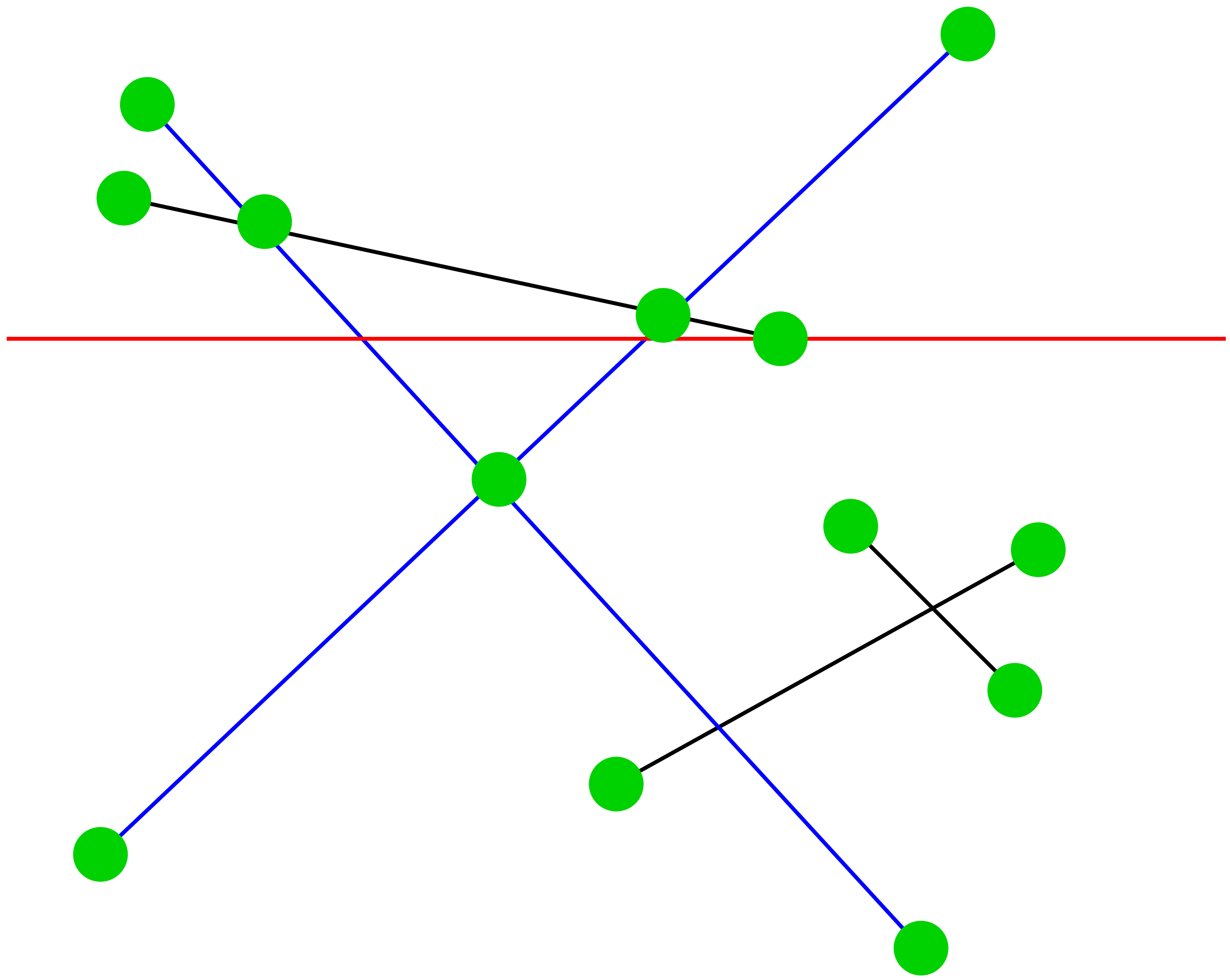
Verallgemeinerung – Beispiel



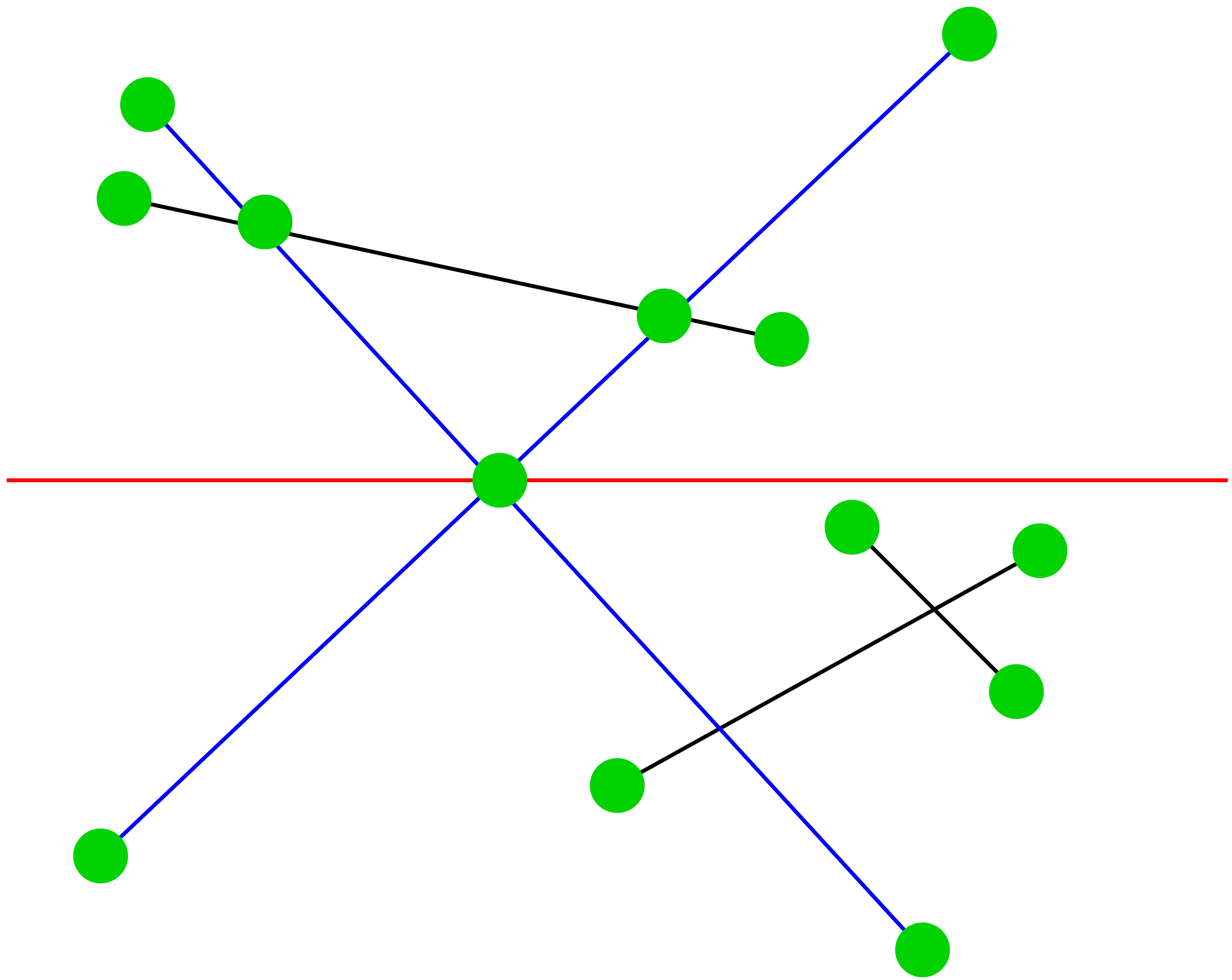
Verallgemeinerung – Beispiel



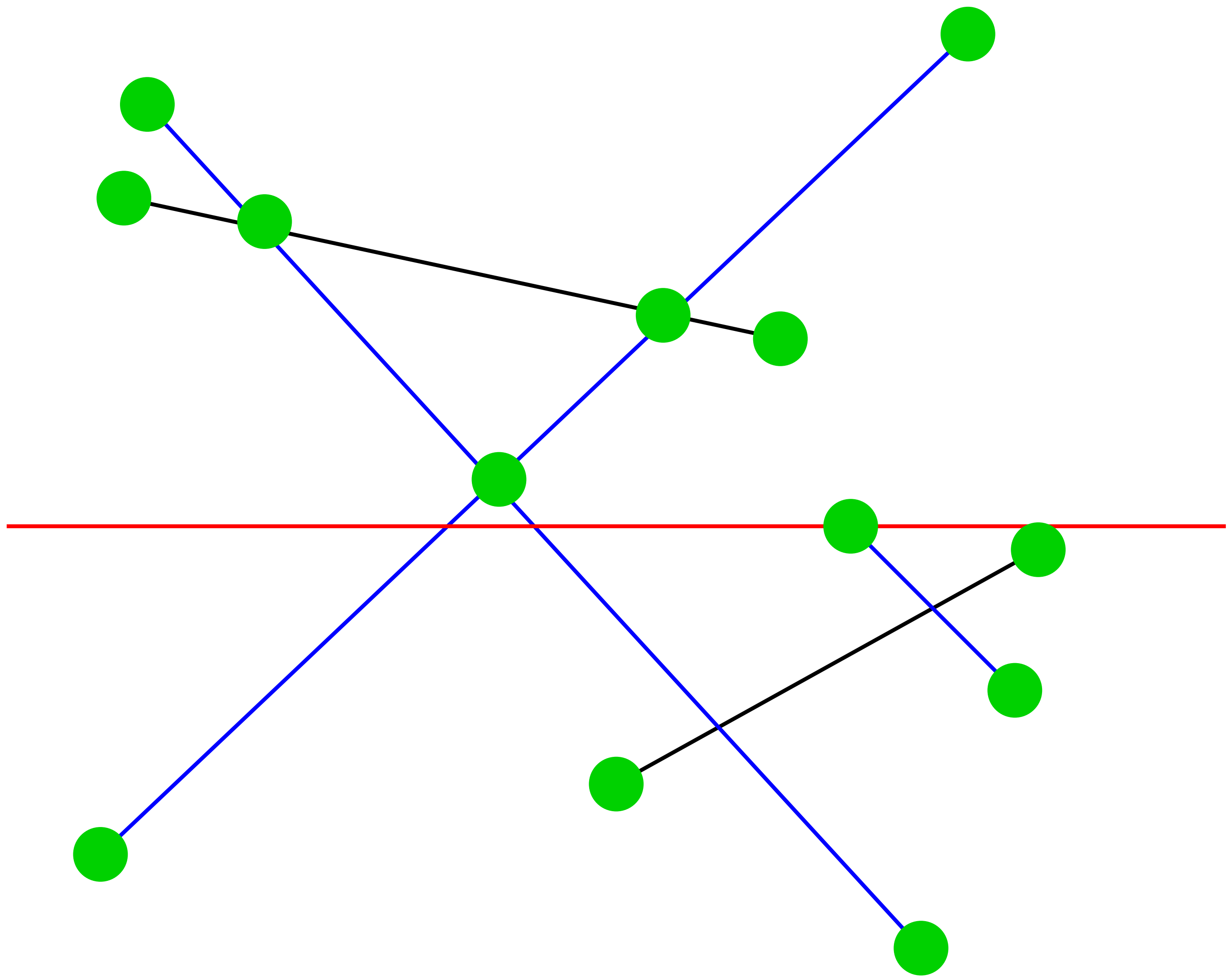
Verallgemeinerung – Beispiel



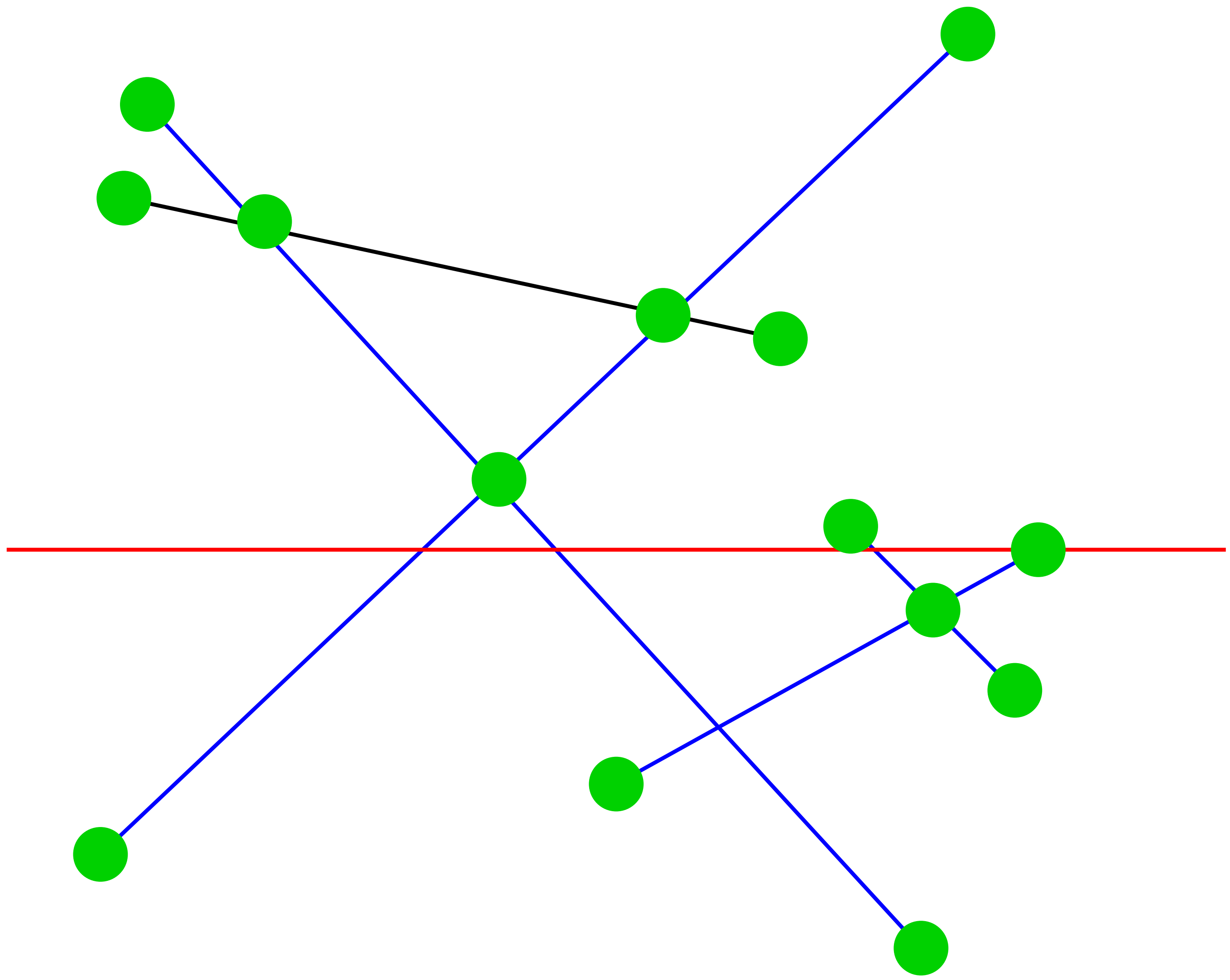
Verallgemeinerung – Beispiel



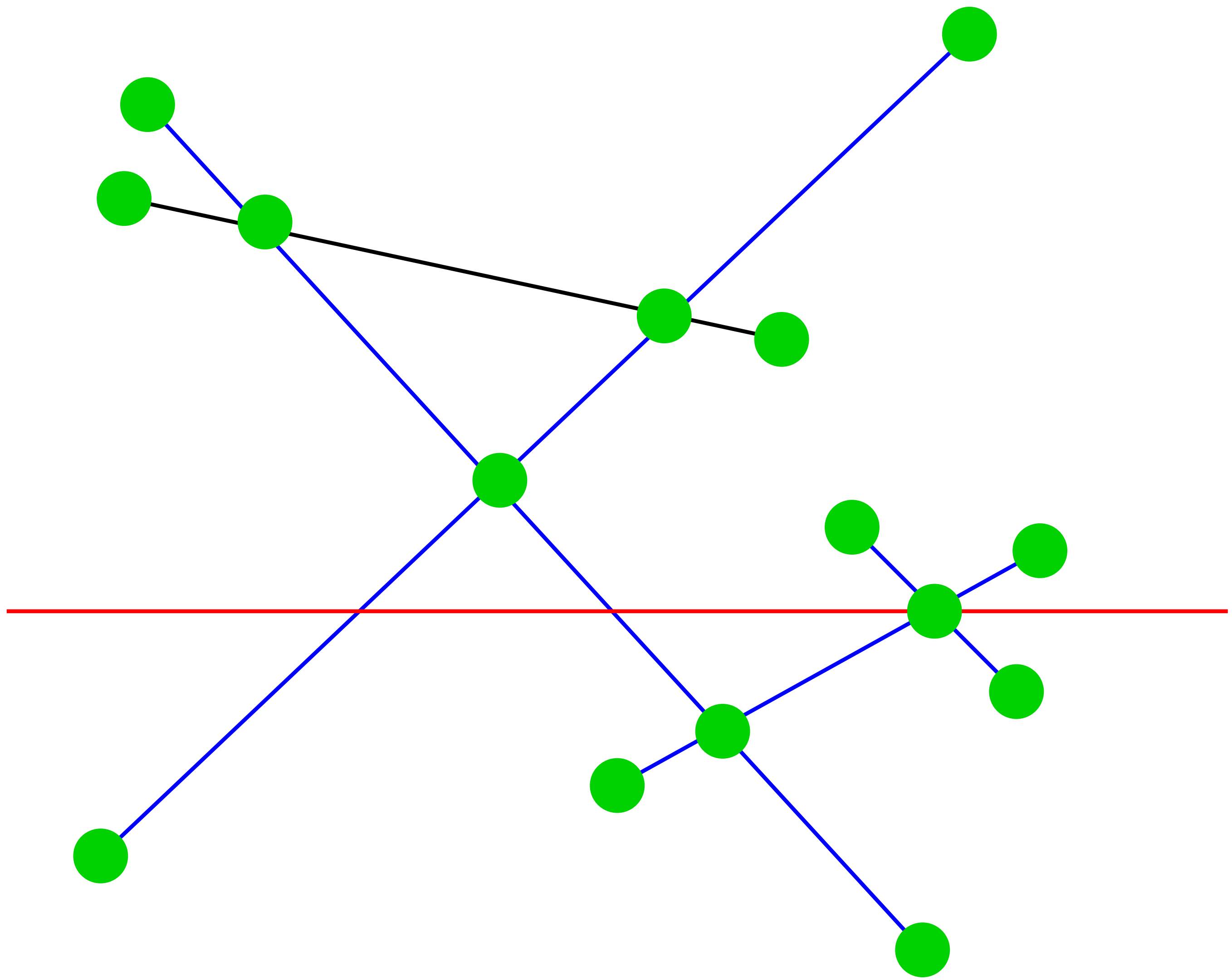
Verallgemeinerung – Beispiel



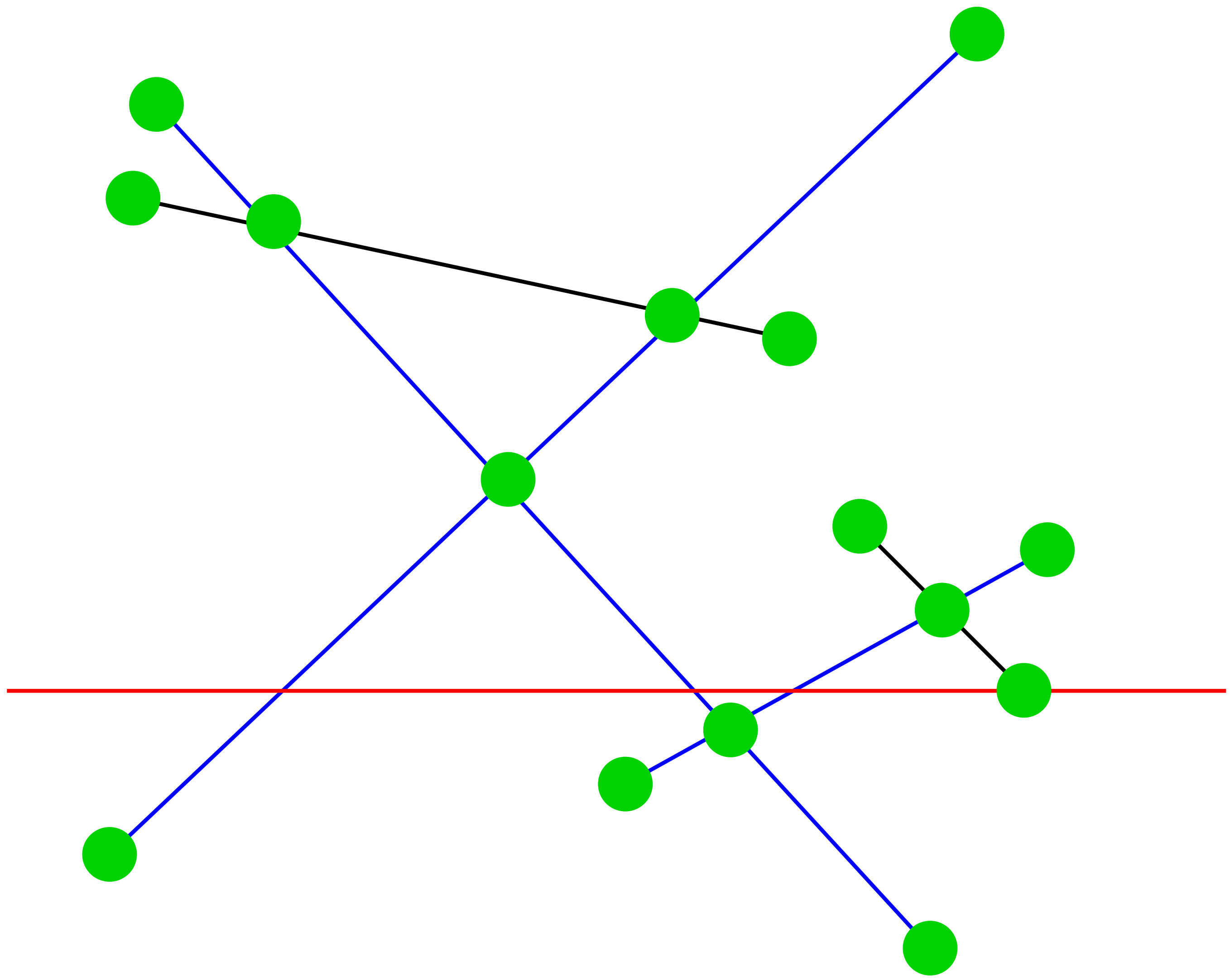
Verallgemeinerung – Beispiel



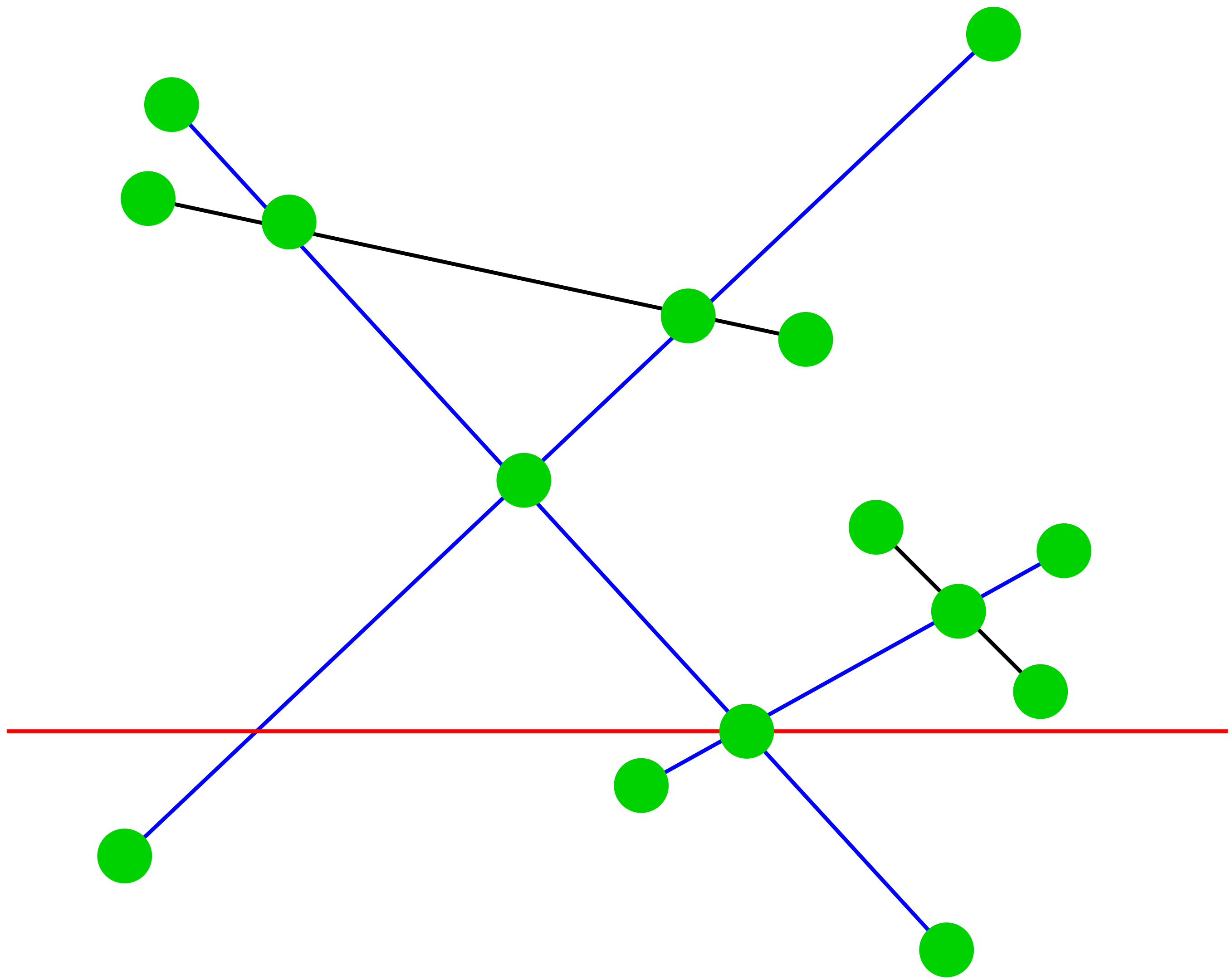
Verallgemeinerung – Beispiel



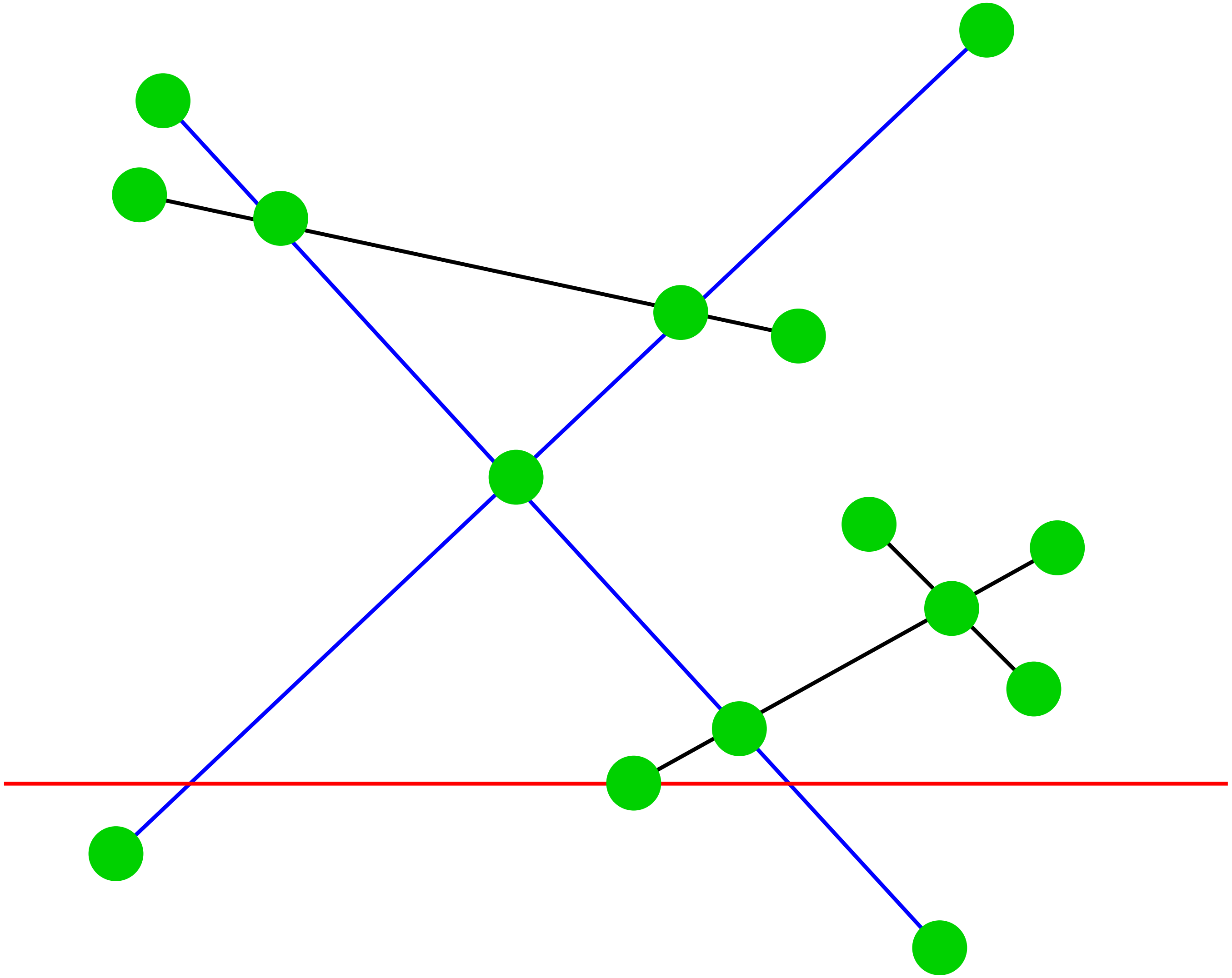
Verallgemeinerung – Beispiel



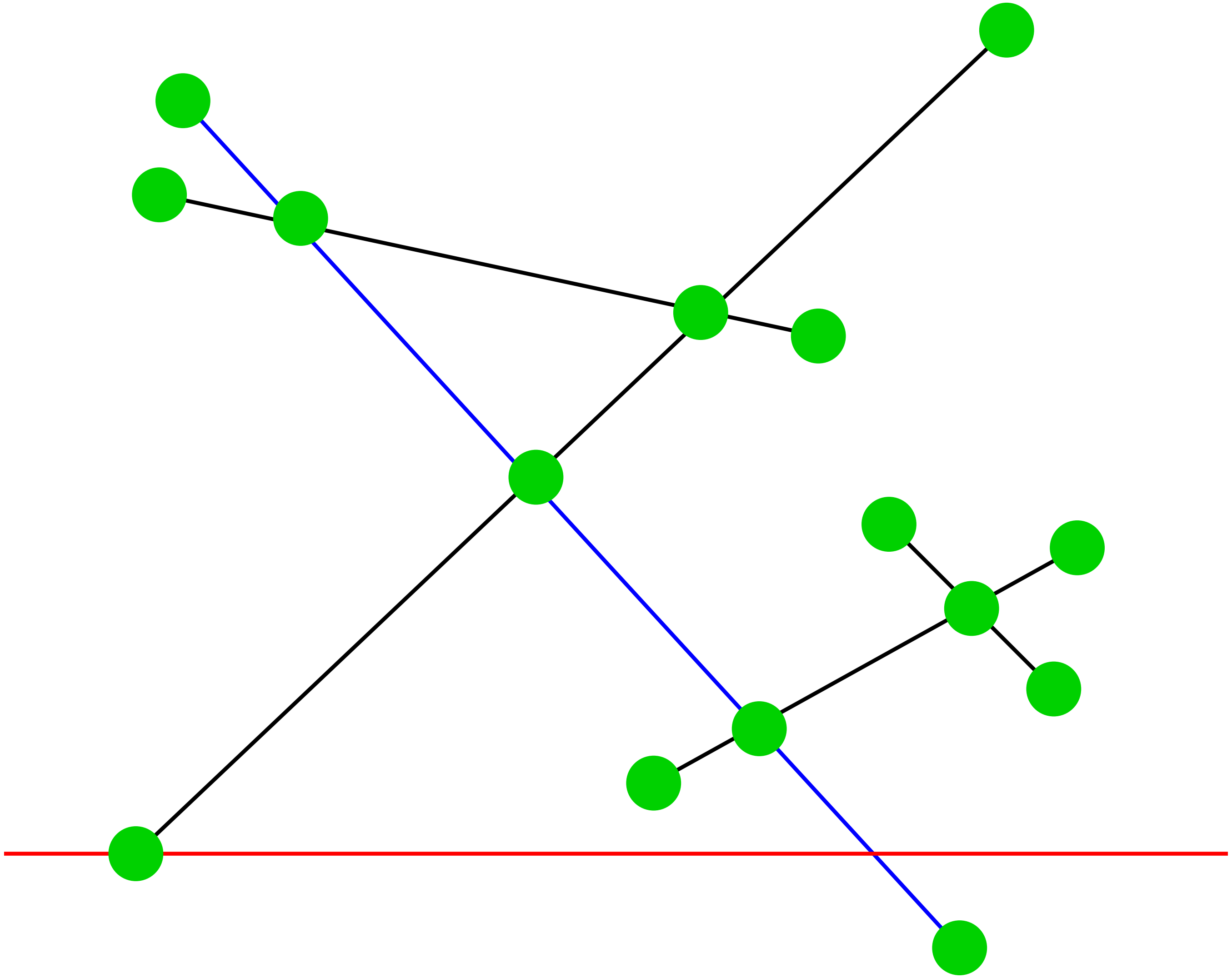
Verallgemeinerung – Beispiel



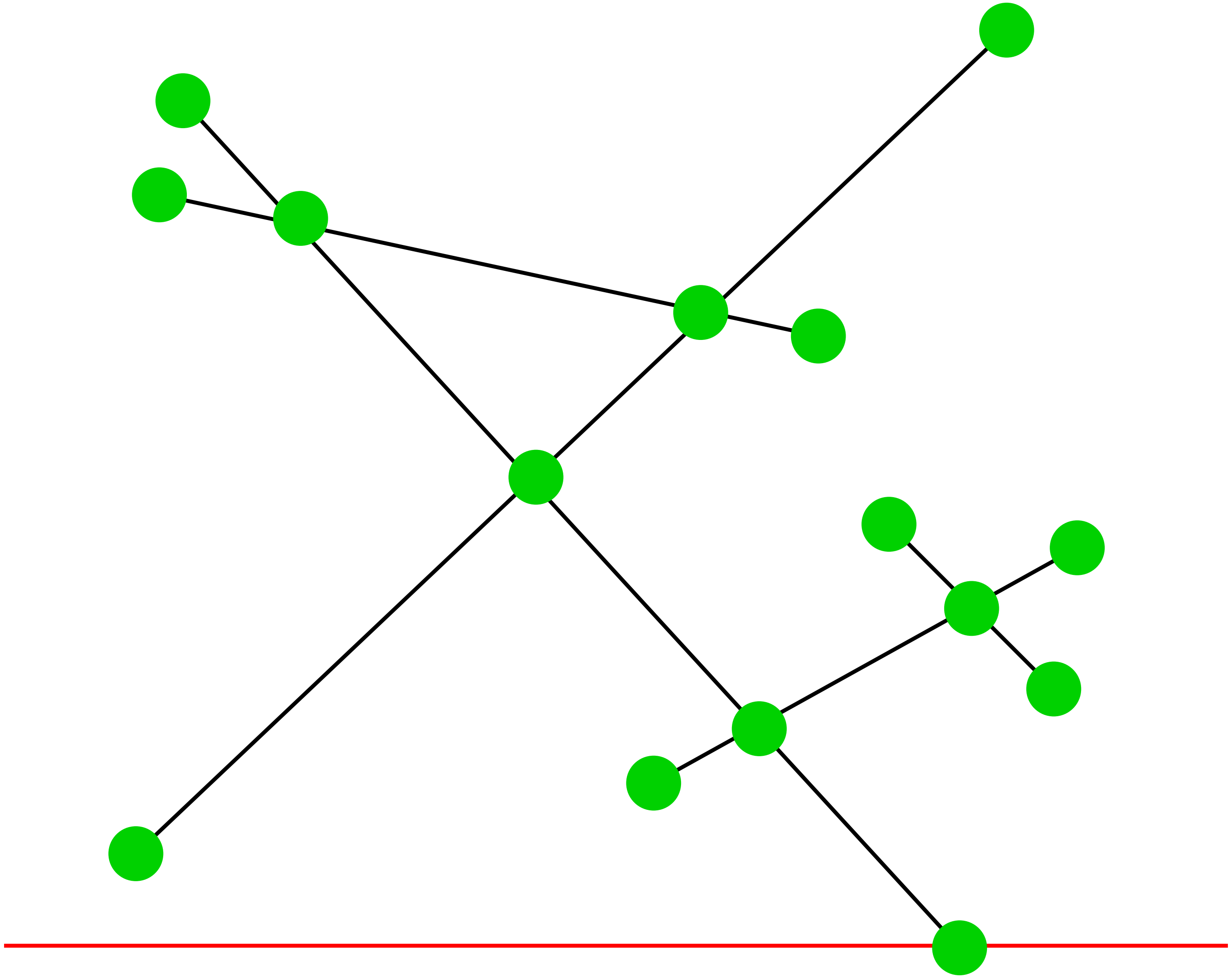
Verallgemeinerung – Beispiel



Verallgemeinerung – Beispiel



Verallgemeinerung – Beispiel



Verallgemeinerung – Analyse

Insgesamt: $\mathrm{O}((n+k)\log n)$

Verallgemeinerung – jetzt (fast) wirklich

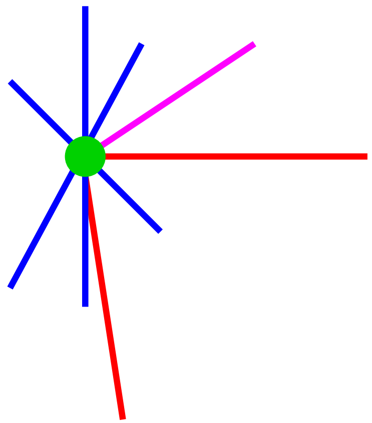
Verbleibende Annahme: Keine Überlappungen

Ordnung für $Q : (x,y) \prec (x',y') \Leftrightarrow y > y' \vee y = y' \wedge x < x'$

(verquere lexikographische Ordnung)

Interpretation: infinitesimal ansteigende Sweep-Line

Q speichert Mengen von Ereignissen mit gleichem (x,y)



handleEvent$(p = (x, y))$

U:= segments starting at p // from Q

C:= segments with p in their interior // from T

L:= segments finishing at p // from Q

if $|U| + |C| + |L| \geq 2$ **then** report intersection @ p

T.remove$(L \cup C)$

T.insert$(C \cup U)$ such that order just below p is correct

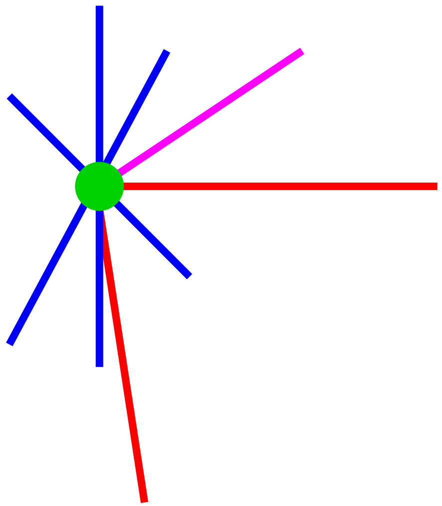
if $U \cup C = \emptyset$ **then**

findNewEvent$(T$.findPred$(p), T$.findSucc$(p), p)$

else

findNewEvent$(T$.findPred$(p), T$.findLeftmost$(p), p)$

findNewEvent$(T$.findRightmost$(p), T$.findSucc$(p), p)$



findNewEvent(s,t,p)

if s and t intersect at a point $p' \succ p$ **then**

if $p' \notin Q$ **then** Q.insert(p')

Überlappungen finden

Für jede Strecke s berechne die Gerade $g(s)$, auf der s liegt

Sortiere S nach $g(s)$

1D Überlappungsproblem für jede auftretende Gerade.

Platzverbrauch

Im Moment: $\Theta(n+k)$

Reduktion auf $\mathrm{O}(n)$:

lösche Schnittpunkte zwischen nicht benachbarten Strecken aus T.

Die werden ohnehin wieder eingefügt wenn sie wieder benachbart werden.

Mehr Linienschnitt

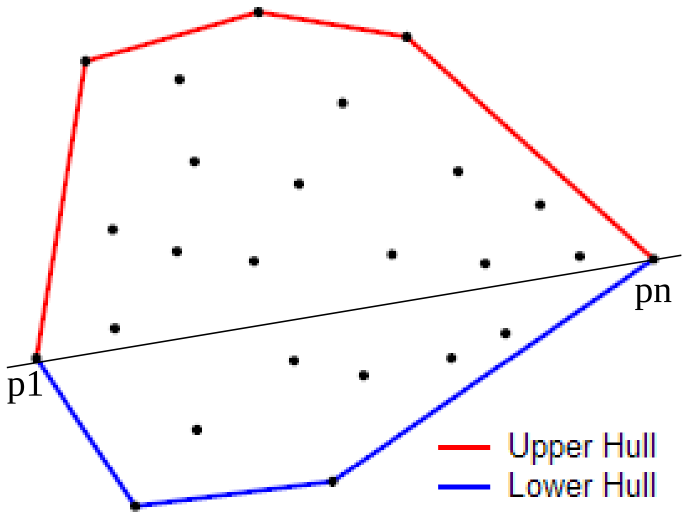
- [Bentley Ottmann 1979] Zeit $\mathrm{O}((n+k)\log n)$
- [Chazelle Edelsbrunner 1988] Zeit $\mathrm{O}(n\log n+k)$
- [Pach Sharir 1991] Zeit $\mathrm{O}((n+k)\log n)$, Platz $\mathrm{O}(n)$
- [Mulmuley 1988] erwartete Zeit $\mathrm{O}(n\log n+k)$, Platz $\mathrm{O}(n)$
- [Balaban 1995] Zeit $\mathrm{O}(n\log n+k)$, Platz $\mathrm{O}(n)$

12.2 2D Konvexe Hülle

Gegeben: Menge $P = \{p_1, \ldots, p_n\}$ von Punkten in $\mathbb{R}^2$

Gesucht: Konvexes Polygon K mit Eckpunkten aus P und $P \subseteq K$.

Wir geben einen einfachen Algorithmus, der in Zeit $\mathrm{O}(\mathrm{sort}(n))$ läuft.



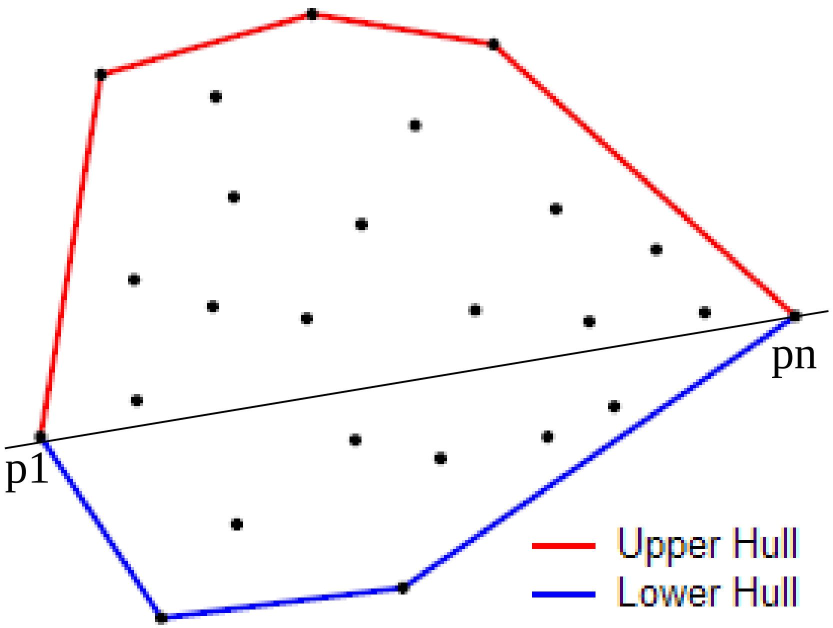
Konvexe Hülle

sortiere P lexikographisch nach (x, y), d.h., ab jetzt

$p_1 < p_2 < \cdots < p_n$

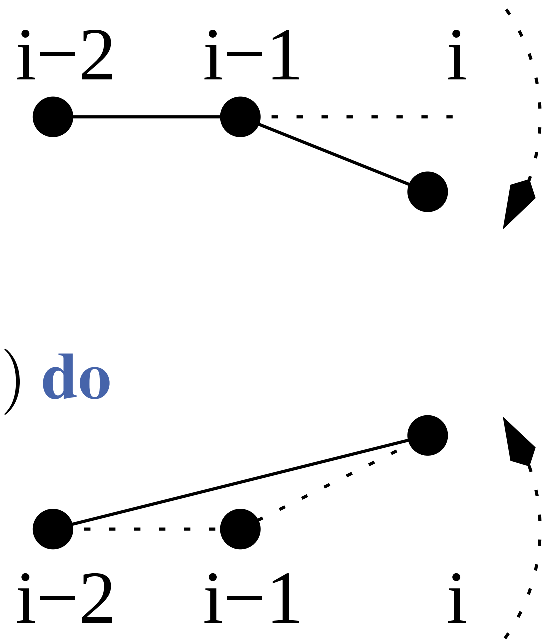
OBdA:

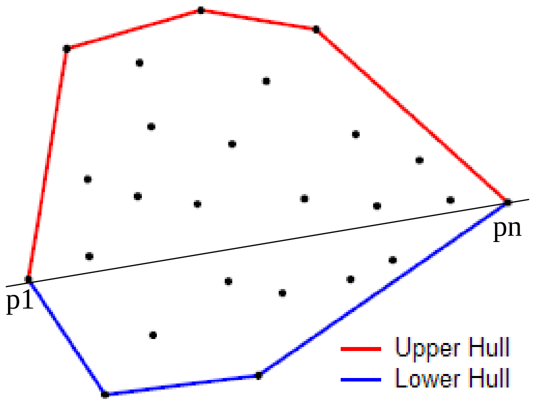
Wir berechnen nur die obere Hülle von Punkten oberhalb von $\overline{p_1 p_n}$



Graham's Scan [Graham 1972, Andrew 1979]

Function upperHull$(p_1, \dots, p_n)$
    $L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point
    invariant L is the upper hull of $\langle p_n, p_1, \dots, p_i \rangle$
    for $i := 3$ **to** n **do**
        while ¬rightTurn(L.secondButlast, L.last, p_i) **do**
            L.pop
        $L := L \circ \langle p_i \rangle$
    return L





Graham's Scan – Beispiel

Function upperHull$(p_1, \ldots, p_n)$

    $L = \langle p_n, p_1, p_2 \rangle$: Stack **of** Point

    invariant L is the upper hull of $\langle p_n, p_1, \ldots, p_i \rangle$

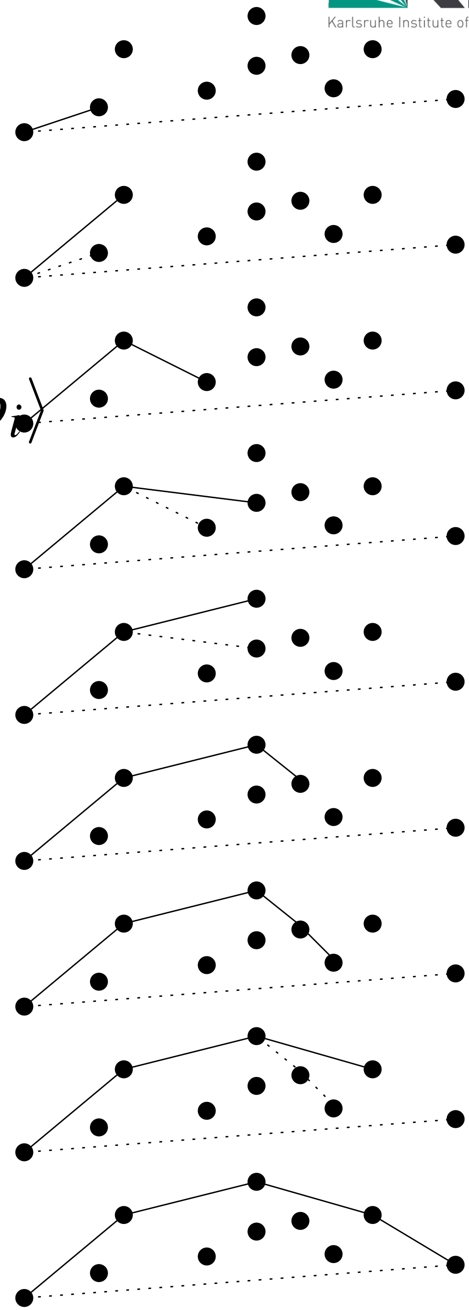
    for $i := 3$ **to** n **do**

        while $\neg$rightTurn(L.secondButlast, L.last, p_i) **do**

            L.pop

        $L := L \circ \langle p_i \rangle$

    return L



Graham's Scan – Analyse

Function upperHull$(p_1,\ldots,p_n)$

  $L{=}\langle p_n, p_1, p_2\rangle$: Stack **of** Point

  invariant L is the upper hull of $\langle p_n, p_1, \ldots, p_i\rangle$

  for $i := 3$ **to** n **do**

    while $\neg$rightTurn$(L$.secondButlast$, L$.last$, p_i)$ **do**

      L.pop

    $L{:=}\, L \circ \langle p_i\rangle$

  return L

Sortieren $+\mathrm{O}(n)$

Wieviele Iterationen der While-Schleife insgesamt?

3D Konvexe Hülle

Geht in Zeit $\mathrm{O}(n \log n)$ [Preparata Hong 1977]

Konvexe Hülle, $d \geq 4$

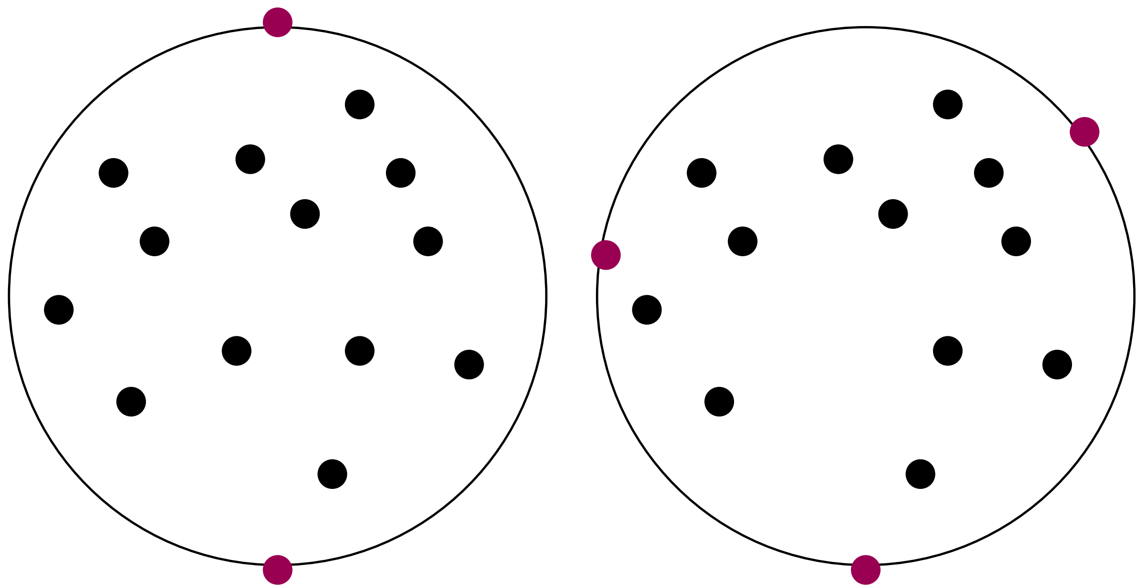
Ausgabekomplexität $\mathrm{O}\left(n^{\lfloor d/2 \rfloor}\right)$

12.3 Kleinste einschließende Kugel

Gegeben: Menge $P = \{p_1, \ldots, p_n\}$ von Punkten in $\mathbb{R}^d$

Gesucht: Kugel K mit minimalem Radius, so dass $P \subseteq K$.

Wir geben einen einfachen Algorithmus, der in erwarteter Zeit $\mathrm{O}(n)$ läuft. [Welzl 1991].



Function smallestEnclosingBallWithPoints(P,Q)

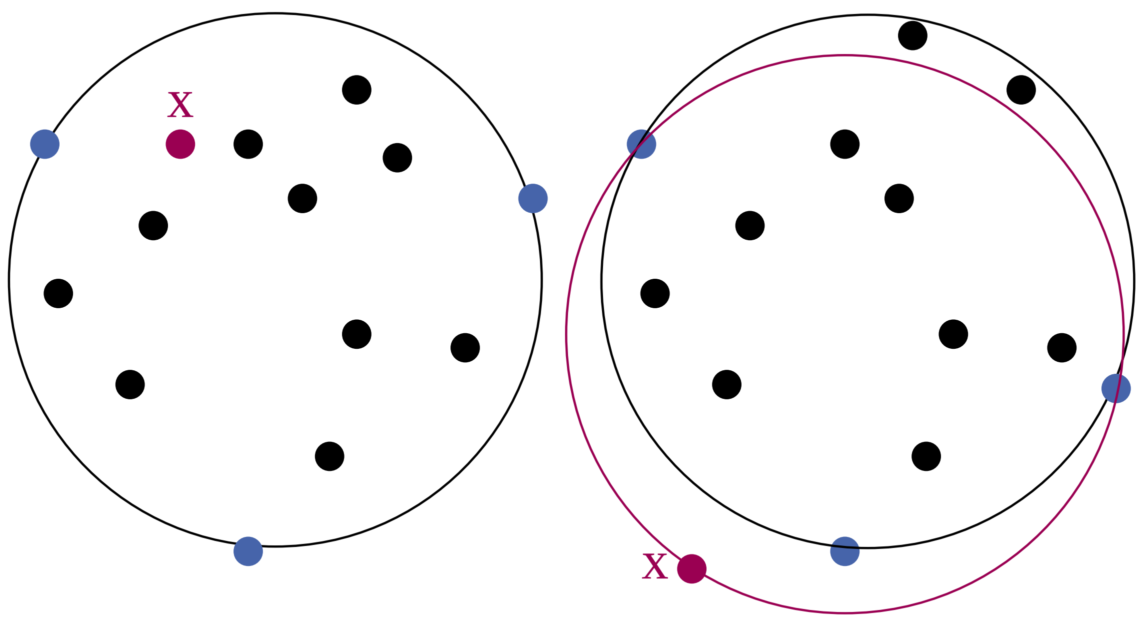
if $|P| = 0 \vee |Q| = d+1$ **then return** ball(Q)

pick random $x \in P$

B:= smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q)$

if $x \in B$ **then return** B

return smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q \cup \{x\})$



Kleinste einschließende Kugel – Korrektheit

Function smallestEnclosingBallWithPoints(P, Q)
  if $|P| = 0 \vee |Q| = d+1$ **then return** ball(Q)
  pick random $x \in P$
  B:= smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q)$
  if $x \in B$ **then return** B
  return smallestEnclosingBallWithPoints$(P \setminus \{x\}, Q \cup \{x\})$

z.Z.: $x \notin B \rightarrow x$ ist auf dem Rand von sEB(P)

Wir zeigen Kontraposition:

x nicht auf dem Rand von sEB(P)

$\longrightarrow$ sEB(P) = sEB$(P \setminus \{x\}) = B$ z.Z.: sEBs sind eindeutig!

Also $x \in B$

Lemma: sEB(P) ist eindeutig bestimmt.

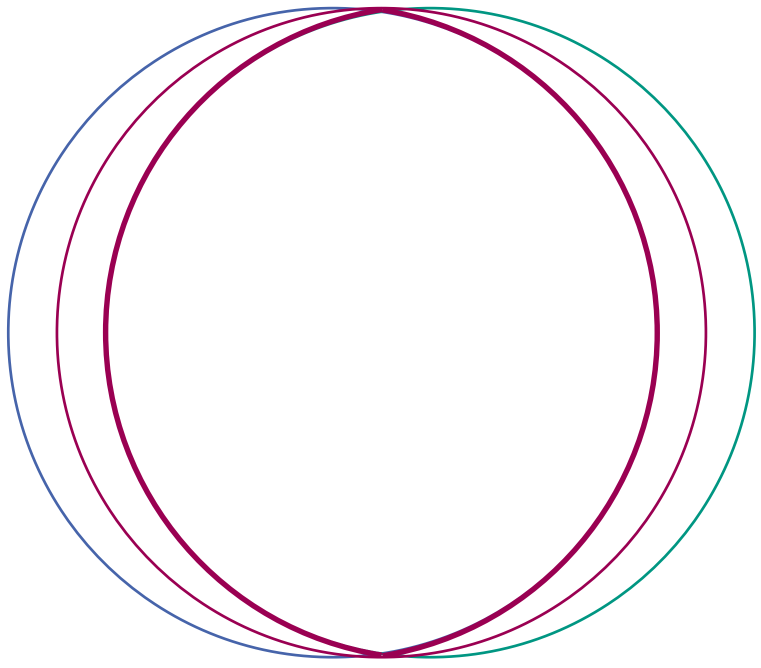
Beweis: Annahme, $\exists$sEBs $B_1 \neq B_2$

$\longrightarrow P \subseteq B_1 \wedge P \subseteq B_2$

$\longrightarrow P \subseteq B_1 \cap B_2 \subseteq \text{sEB}(B_1 \cap B_2) =: B$

Aber dann ist radius$(B) <$ radius(B_1)

Widerspruch zur Annahme, dass B_1 eine sEB ist. $\square$



Kleinste einschließende Kugel – Analyse

Wir zählen die erwartete Anzahl der Tests $x \in B$, $T(p,q)$.

$$T(p,d+1) = T(0,p) = 0 \qquad \text{Basis der Rekurrenz}$$

$$\begin{aligned} T(p,q) &\leq 1 + T(p-1,q) + \mathbb{P}[x \notin B]\,T(p,q+1) \\ &\leq 1 + T(p-1,q) + \frac{d+1-q}{p} T(p,q+1) \end{aligned}$$

Kleinste einschließende Kugel – Analyse, $d = 2$

$T(p, d+1) = T(0, q) = 0$

$$T(p,q) \leq 1 + T(p-1,q) + \frac{d+1-q}{p} T(p,q+1)$$

$$T(p,2) \leq 1 + T(p-1,2) + \frac{1}{p} T(p,3) \leq 1 + T(p-1,2) \leq p$$

$$T(p,1) \leq 1 + T(p-1,1) + \frac{2}{p} T(p,2)$$

$$\leq 1 + T(p-1,1) + \frac{2}{p} p = 3 + T(p-1,1) \leq 3p$$

$$T(p,0) \leq 1 + T(p-1,0) + \frac{3}{p} T(p,1)$$

$$\leq 1 + T(p-1,0) + \frac{3}{p} 3p = 10 + T(p-1,0) \leq 10p$$

Kleinste einschließende Kugel – Analyse

d	$T(p,0)$
1	$3n$
2	$10n$
3	$41n$
4	$206n$

Allgemein $T(p,0) \geq d!n$

Ähnliche Randomisierte Linearzeitalgorithmen

- Lineare Programmierung mit konstantem d [Seidel 1991]
- Kleinster einschließender Ellipsoid, Kreisring,...
- Support-Vector-Machines (maschinelles Lernen)
- Alles wo (LP-type problem [Sharir Welzl 1992])
  - $\mathrm{O}(1)$ Objekte das Optimum festlegen
  - Objekt x hinzufügen
    $\longrightarrow$ Lösung bleibt gleich oder ist an Lösungsdef. beteiligt

12.4 2D Bereichssuche (range search)

Daten: $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

Anfragen: achsenparallele Rechtecke $Q = [x, x'] \times [y, y']$

finde $P \cap Q$ (range reporting)

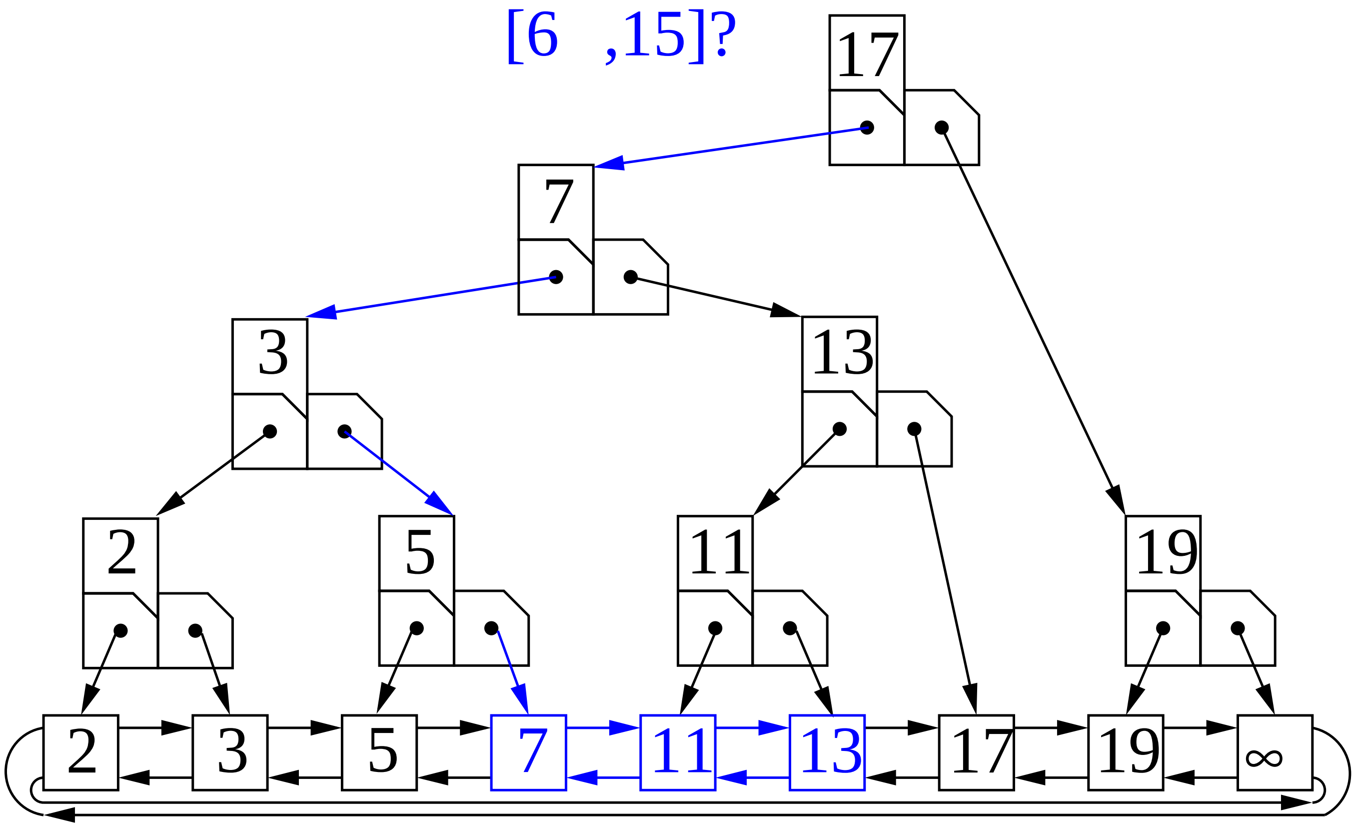
oder $k = |P \cap Q|$ (range counting)

Vorverarbeitung erlaubt.

- Vorverarbeitungszeit? $\mathrm{O}(n \log n)$
- Platz? $\mathrm{O}(n)$?
- Anfragebearbeitung?
  - Counting: $\mathrm{O}(\log n)$
  - Reporting: $\mathrm{O}(k + \log n)$ oder wenigstens $\mathrm{O}(k \cdot \log n)$

1D Bereichssuche

Suchbaum



Zählanfragen: Teilbaumgrößen speichern

Sogar dynamisch !

Reduktion auf $1..n \times 1..n$

vereinfachende Annahme: Koordinaten paarweise verschieden.

Ersetze Koordinaten $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ durch ihren Rang:

$x_i \rightarrow$ Rang von x_i in $\{x_1, \ldots, x_n\}$

$y_i \rightarrow$ Rang von y_i in $\{y_1, \ldots, y_n\}$

Reduktion auf $1..n \times 1..n$

Ersetze Koordinaten $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ durch ihren Rang:

P_x:= sort$(\langle x_1, \ldots, x_n \rangle)$; P_y:= sort$(\langle y_1, \ldots, y_n \rangle)$

P:= $\{$(binarySearch(x, P_x), binarySearch$(y, P_y)) : (x, y) \in P\}$

Function rangeQuery$([x, x'] \times [y, y'])$

  x:= binarySearchSucc(x, P_x); x':= binarySearchPred(x', P_x)

  y:= binarySearchSucc(y, P_y); y':= binarySearchPred(y', P_y)

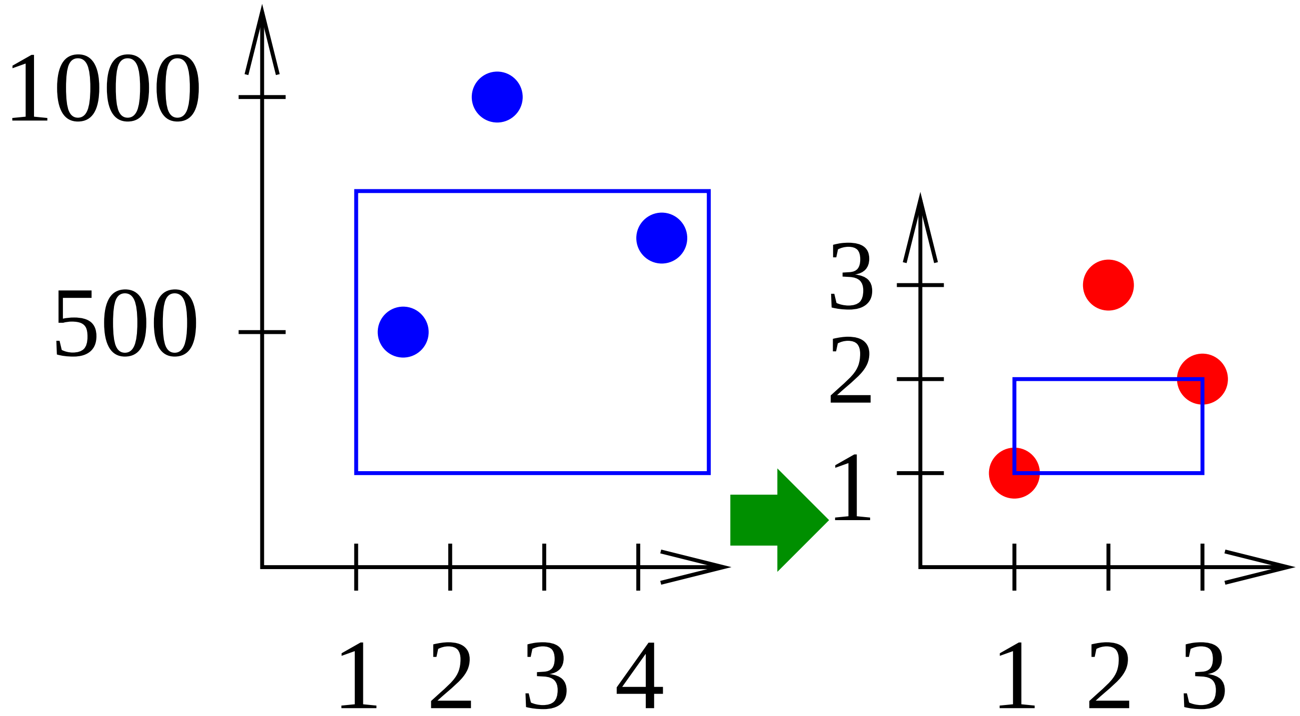
  R:= intRangeQuery$([x, x'] \times [y, y'])$

  return $\{(P_x[x], P_y[y]) : (x, y) \in R\}$

A Konstruktionszeit $\mathrm{O}(n \log n)$, Anfragezeit $\mathrm{O}(\log n) + T_{\text{intRangeQuery}}(n)$

Beispiel

$$\{(2.5,1000),(1.4,500),(4.2,700)\} \rightarrow \{(2,3),(1,1),(3,2)\}$$



Wavelet Tree

[Chazelle 1988, Grossi/Gupta/Vitter 2003, Mäkinen/Navarro 2007]

Class WaveletTree$(X = \langle x_1, \ldots, x_n \rangle)$// represents $(x_1, 1), \ldots, (x_n, n)$

 //Constructor:

 if $n < n_0$ **then** store X directly

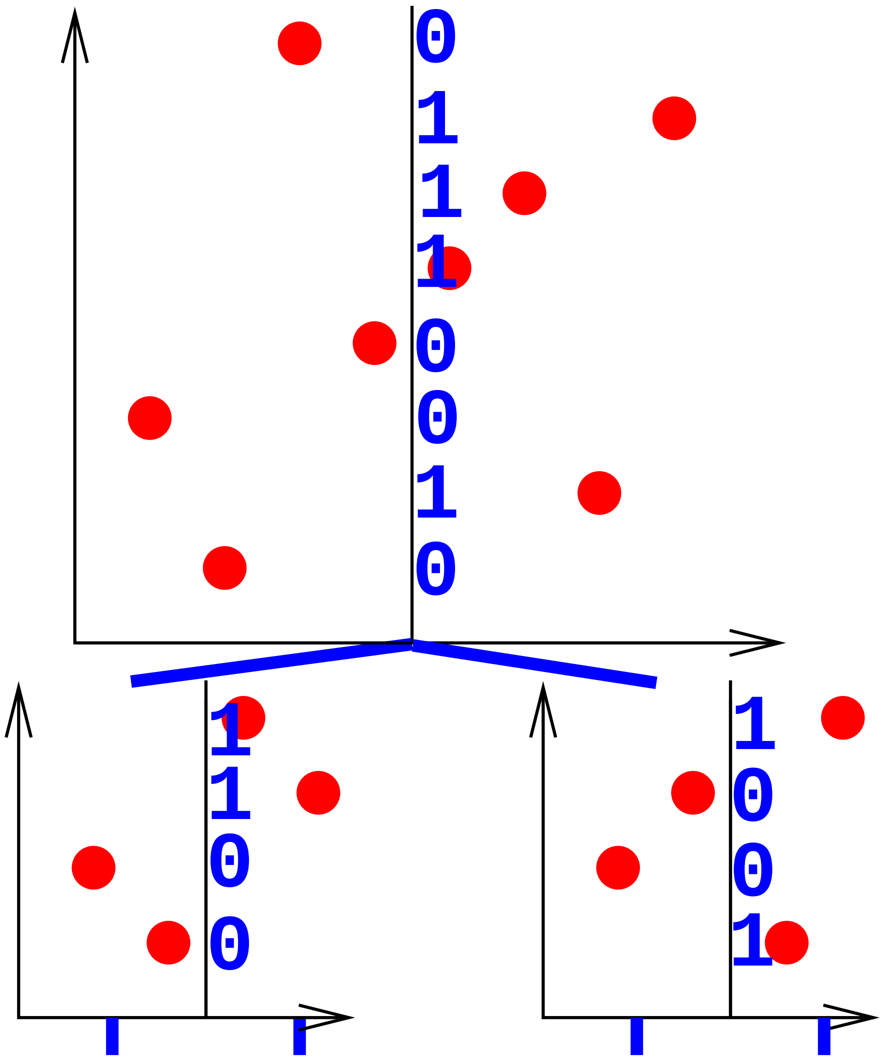
 else

  store bitvector b with $b[i] = 1$ iff $x_i > \lfloor n/2 \rfloor$

  ℓ:= WaveletTree$(\langle x_i : x_i \leq \lfloor n/2 \rfloor \rangle)$

  r:= WaveletTree$(\langle x_i - \lfloor n/2 \rfloor : x_i > \lfloor n/2 \rfloor \rangle)$

Beispiel



$\langle 2,1\rangle$ $\langle 2,1\rangle$ $\langle 1,2\rangle$ $\langle 1,2\rangle$

Wavelet Tree Counting Query

Function intRangeCount$([x,x'] \times [y,y'])$

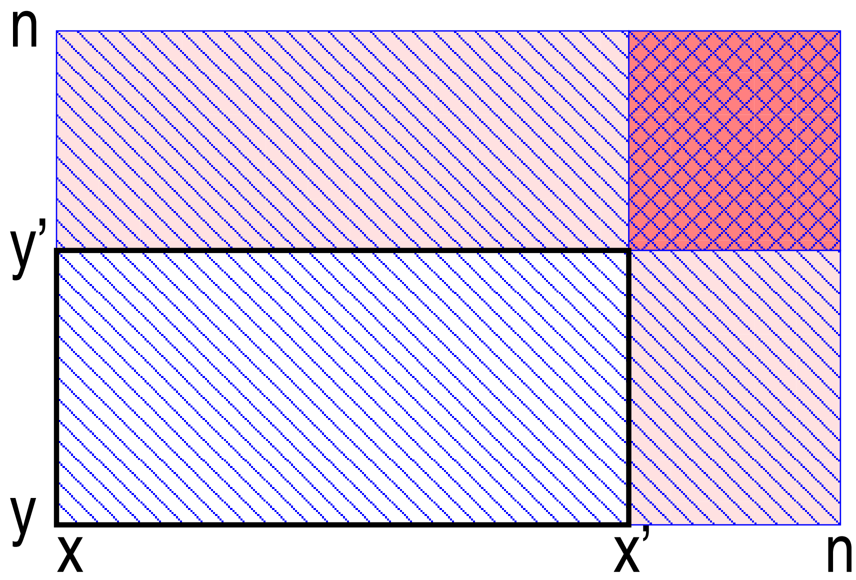
return

intDominanceCount$(x,y)-$

intDominanceCount$(x',y)-$

intDominanceCount$(x,y')+$

intDominanceCount(x',y')



Wavelet Tree Dominance Counting Query

Function intDominanceCount(x, y) // $|[x,n] \times [y,n] \cap P|$

if $n \leq n_0$ **then return** $|[x,n] \times [y,n] \cap P|$ // brute force

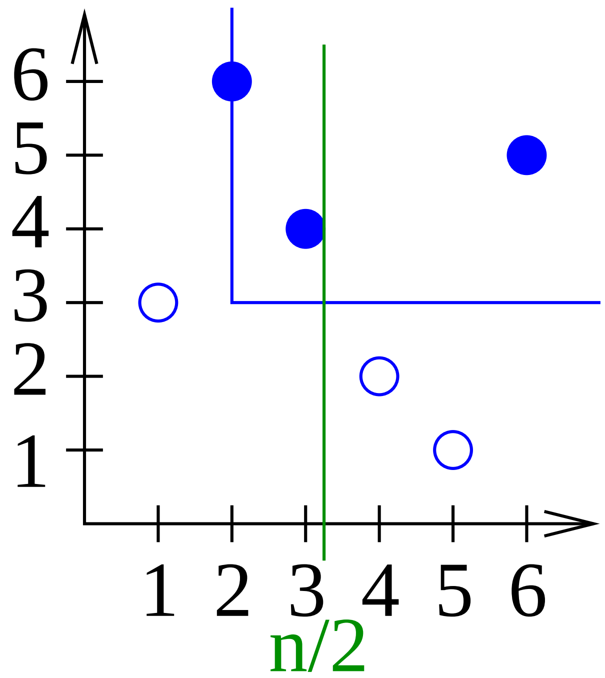
y_r:= b.rank(y) // Number of els $\leq y$ in right half

if $x \leq \lfloor n/2 \rfloor$ **then**

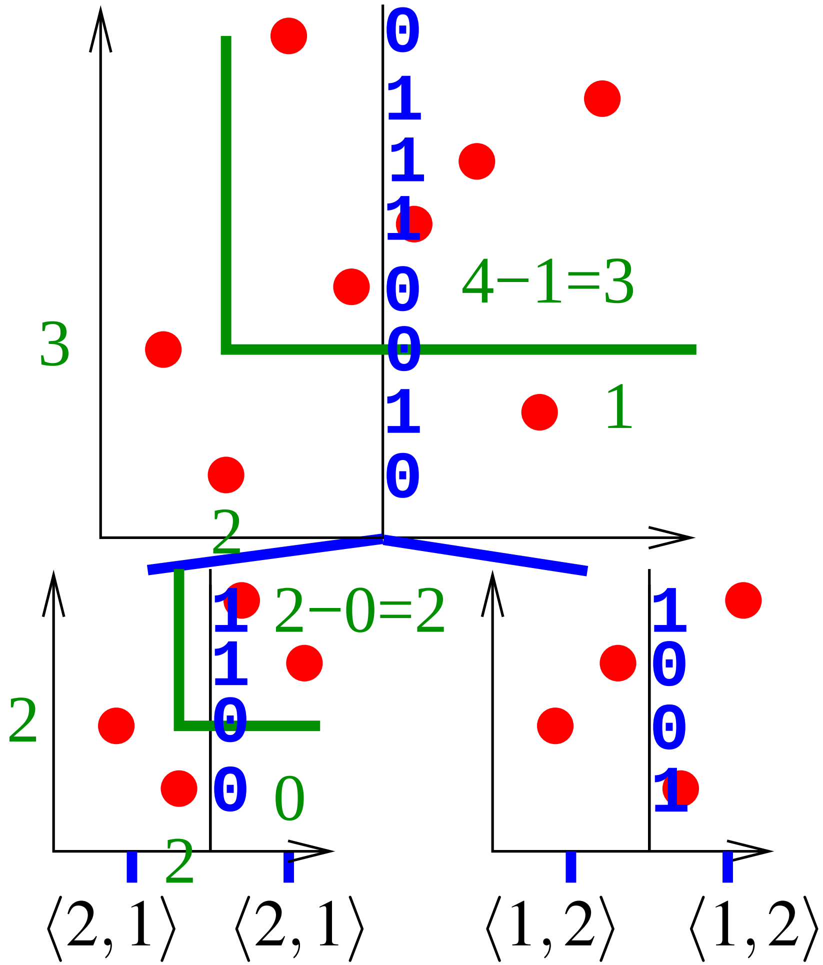
return ℓ.intDominanceCount$(x, y - y_r) + \lceil n/2 \rceil - y_r$

else

return r.intDominanceCount$(x - \lfloor n/2 \rfloor, y_r)$



Beispiel



Analyse

Nur ein rekursiver Aufruf.

Rekursionstiefe $\mathrm{O}(\log n)$.

rank in konstanter Zeit (s.u.)

Zeit $\mathrm{O}(\log n)$

Wavelet Tree Dominance Reporting Query

Function intDominanceReporting(x, y) // $[x, n] \times [y, n] \cap P$

  if $n \leq n_0$ **then return** $[x, n] \times [y, n] \cap P$ // brute force

  $R := \emptyset$ // Result

  $y_r := b.\mathrm{rank}(y)$ // Number of els $\leq y$ in right half

  if $x \leq \lfloor n/2 \rfloor$ **then** // Both halves interesting

    if $y - y_r < \frac{n}{2}$ **then** $R := R \cup \ell.\mathrm{intDominanceReporting}(x, y - y_r)$

    if $y_r < \frac{n}{2}$ **then** $R := R \cup r.\mathrm{oneSidedReporting}(y_r)$

  else if $y_r < \frac{n}{2}$ **then** $R := R \cup r.\mathrm{intDominanceReporting}(x - \lfloor n/2 \rfloor, y_r)$

  return R

Function oneSidedReporting(y) // $[1,n] \times [y,n] \cap P$

  if $n \leq n_0$ **then return** $[1,n] \times [y,n] \cap P$ // brute force

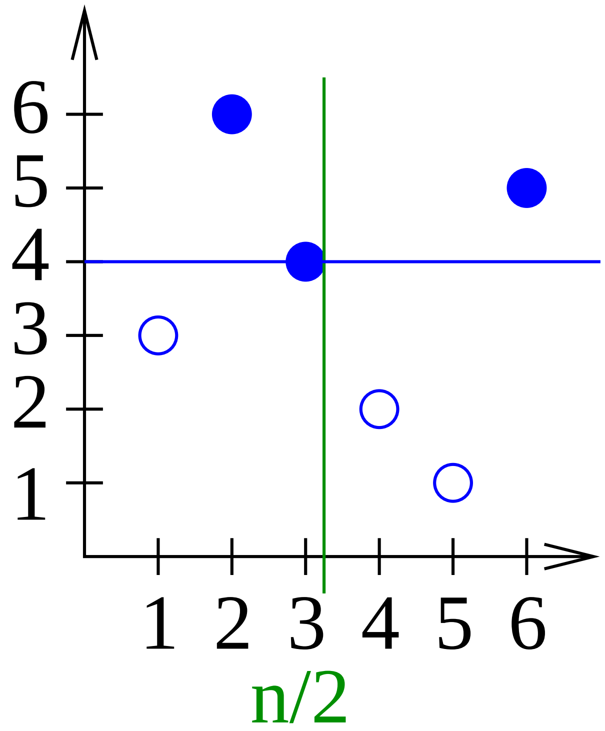
  $y_r := b.\mathrm{rank}(y)$ // Number of els $\leq y$ in right half

  $R := \emptyset$

  if $y_r < \frac{n}{2}$ **then** $R := R \cup r.\mathrm{oneSidedReporting}(y_r)$

  if $y - y_r < \frac{n}{2}$ **then** $R := R \cup \ell.\mathrm{oneSidedReporting}(y - y_r)$

  return R



Analyse

Rekurrenz

$T(n_0,0) = \mathrm{O}(1)$

$T(n,0) = T(n/2,0) + O(1) \Longrightarrow T(n,0) = \mathrm{O}(\log n)$

$T(n,k) = T(n/2,k') + T(n/2,k-k') + O(1)$ also

$T(n,k) \le ck'\log\frac{n}{2} + c(k-k')\log\frac{n}{2} + c \le ck\log n$

Zeit $\mathrm{O}(k+\log n)$ braucht zusätzlichen Faktor $\log n$ Platz.

Z.B. komplette Listen auf allen Ebenen speichern

Übungsaufgabe?

Allgemeine Reporting Query

4-seitig

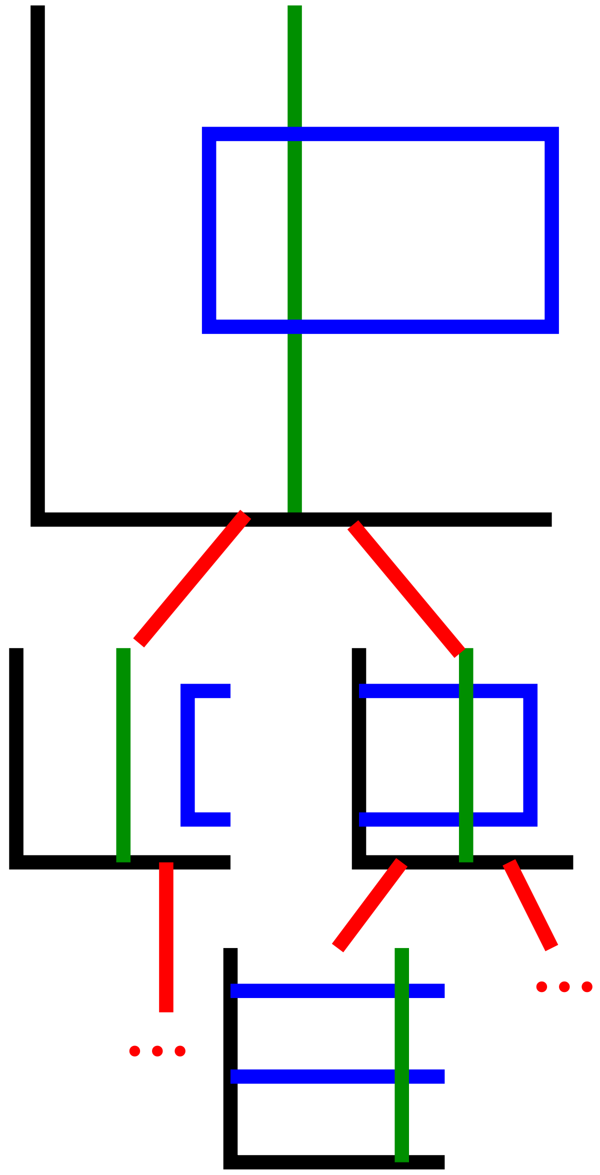
$\rightsquigarrow$

3-seitig (2 Varianten)

$\rightsquigarrow$

y-range

Analog oneSidedReporting (zwei Ranks statt einem)



Bitvektoren v mit rank in $\mathrm{O}(1)$

Wähle $B = \Theta(\log n)$.

Vorberechnung bRank$[i]$:= v.rank(iB)

Zeit $\mathrm{O}(n)$, Platz $\mathrm{O}(n)$ bits

Reduktion auf logarithmische Eingabegröße:

Function rank(j) **return** bRank$[j$ **div** $B]$ + rank$(v[B(j$ **div** $B)..j])$

Logarithmische Größe:

Maschinenbefehl (population count) oder Tabellenzugriff (z.B. Größe $\sqrt{n}$ Zahlen)

0 4 8 11

10010011 10001110 00001110 10100010

rank=8+2=10

Mehr zu Bitvektoren

- weitere wichtige Operation b.select(i):= Position des i-ten 1-bits. Ebenfalls $\mathrm{O}(1)$
- Informationstheoretisch asympt. optimaler Platz $n + o(n)$ bits möglich.
- Grundlage für weitere succinct data structures
- Bespiel: Baum mit Platz $2n + o(n)$ bits und Navigation in konstanter Zeit.

Geometry Summary

- Theoretical algorithms often make assumption like “general position” that need to be removed in practice
- Geometric predicates like *leftTurn* or *inCircle* package the numerically complicated issues
- High-dimensional problems particularly interesting/difficult
- Randomization and data structures once more important