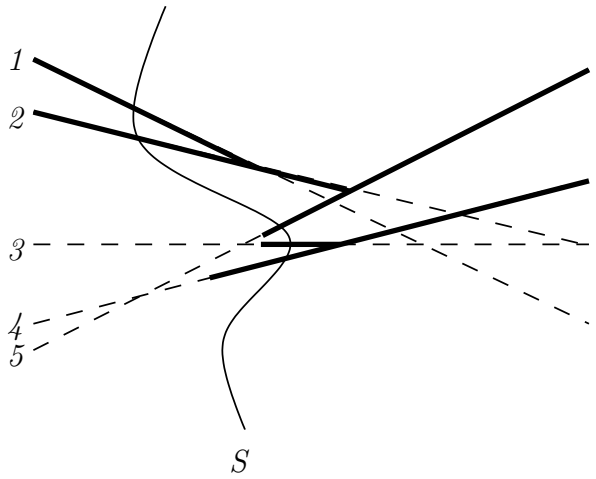
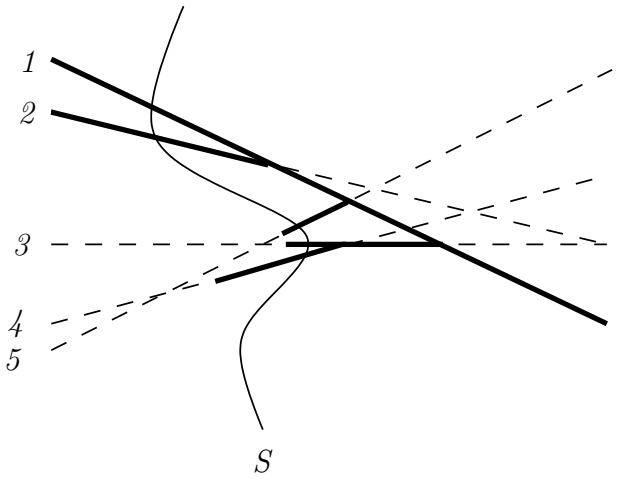
Offline Bewegungsplanung: Horizontbäume

Elmar Langetepe
University of Bonn

Datenstruktur Horizontbäume



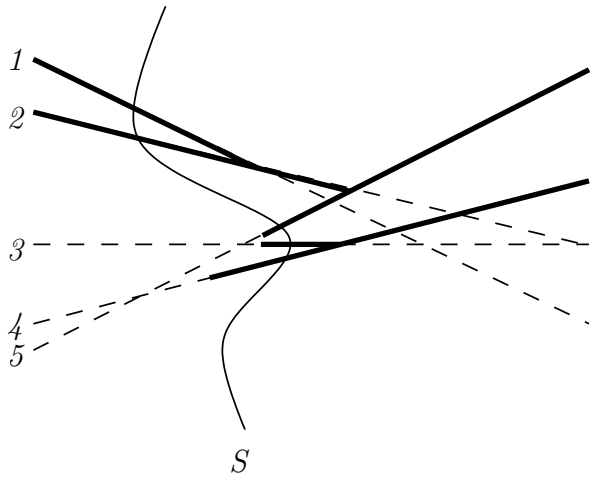
Oberer Horizontbaum



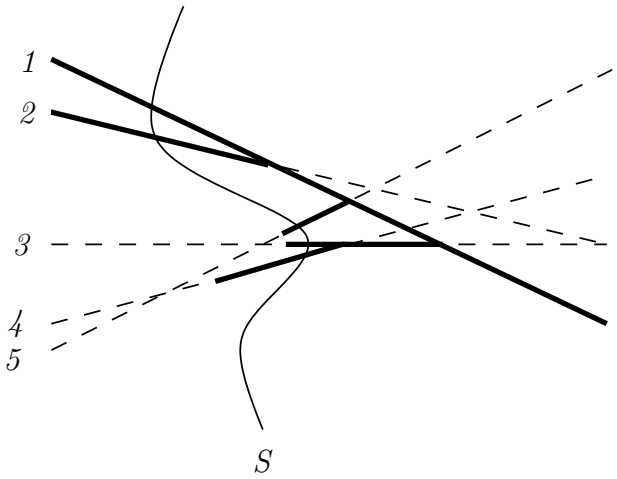
Unterer Horizontbaum

Datenstruktur Horizontbäume

- Für Pseudogerade mit Ordnung



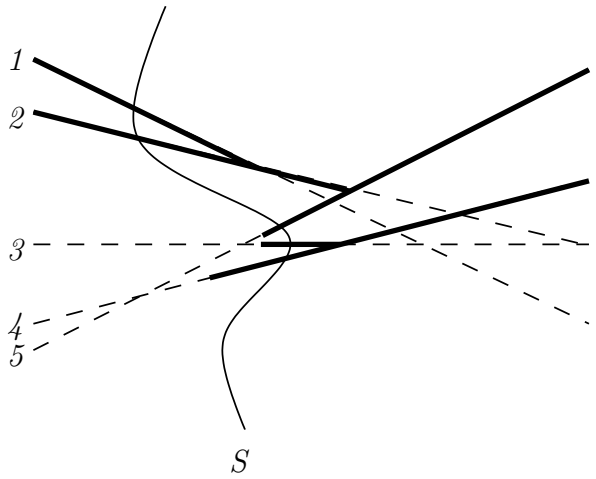
Oberer Horizontbaum



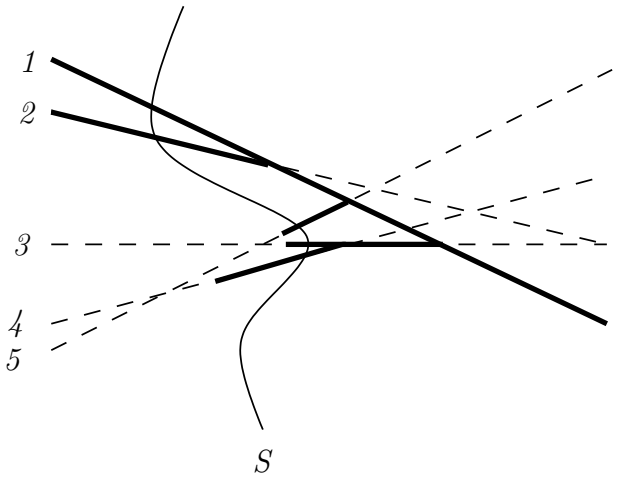
Unterer Horizontbaum

Datenstruktur Horizontbäume

- Für Pseudogerade mit Ordnung
- Oberer und Unterer Horizontbaum gegeben



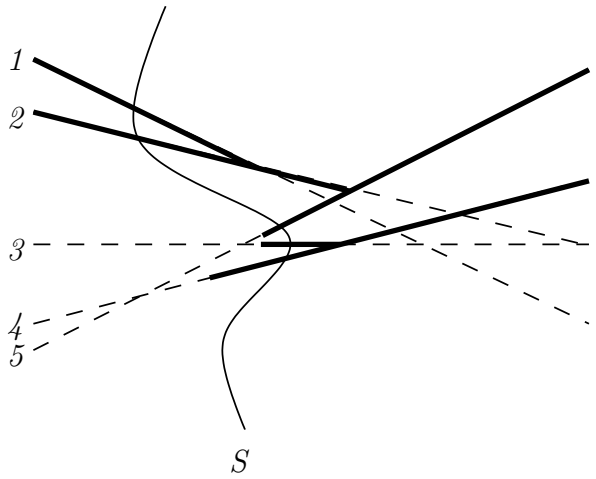
Oberer Horizontbaum



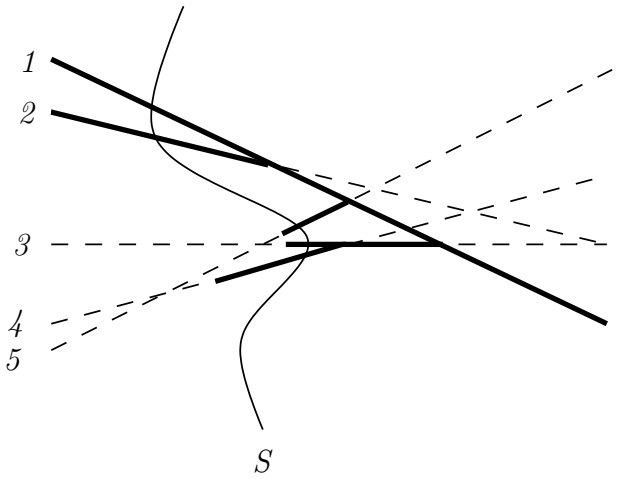
Unterer Horizontbaum

Datenstruktur Horizontbäume

- Für Pseudogerade mit Ordnung
- Oberer und Unterer Horizontbaum gegeben
- Hebe über nächsten erlaubten SP



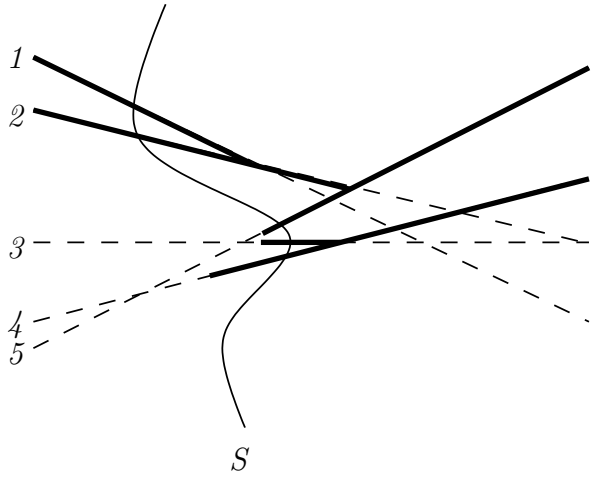
Oberer Horizontbaum



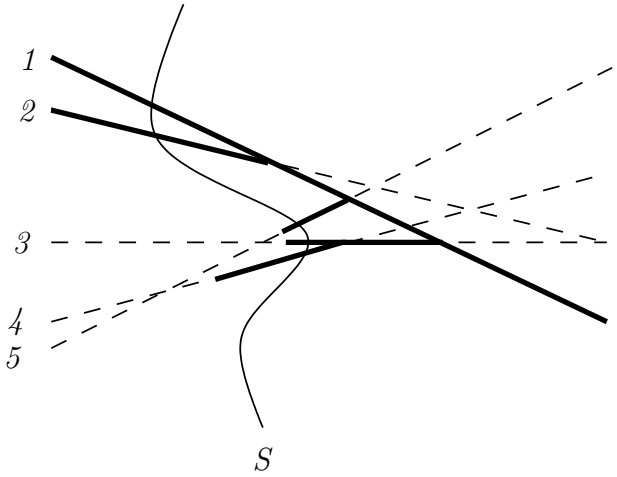
Unterer Horizontbaum

Datenstruktur Horizontbäume

- Für Pseudogerade mit Ordnung
- Oberer und Unterer Horizontbaum gegeben
- Hebe über nächsten erlaubten SP
- Aktualisiere die Bäume



Oberer Horizontbaum

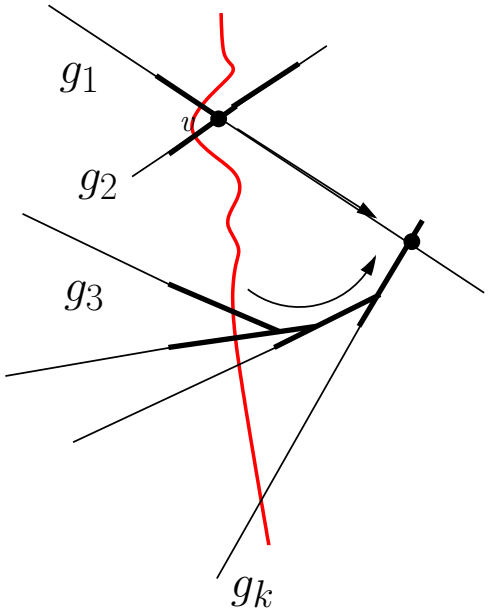


Unterer Horizontbaum

Kosten Aktualisierung T^o (T_u)

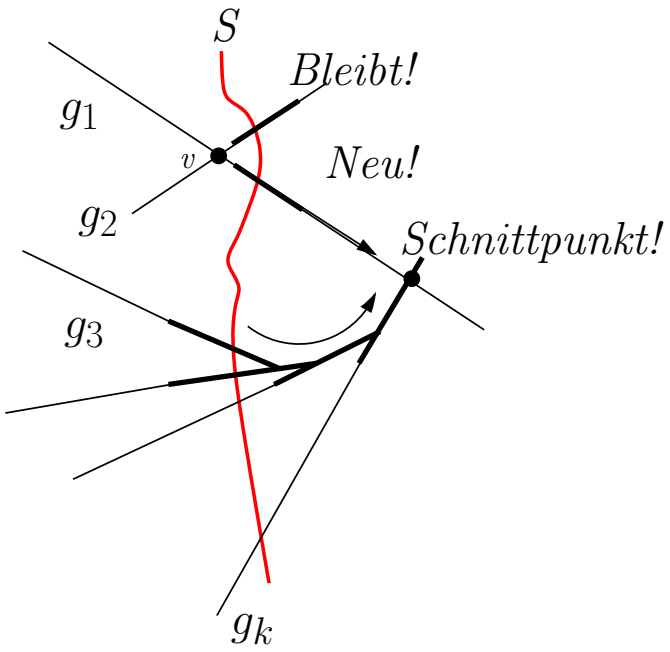
Kosten Aktualisierung T^o (T_u)

1. Nächster Knoten v!



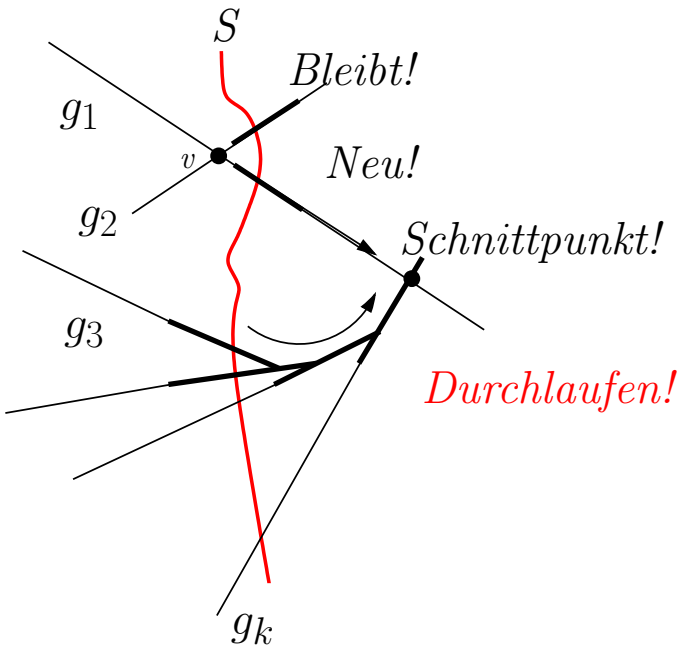
Kosten Aktualisierung T^o (T_u)

1. Nächster Knoten v!
2. Drüberheben



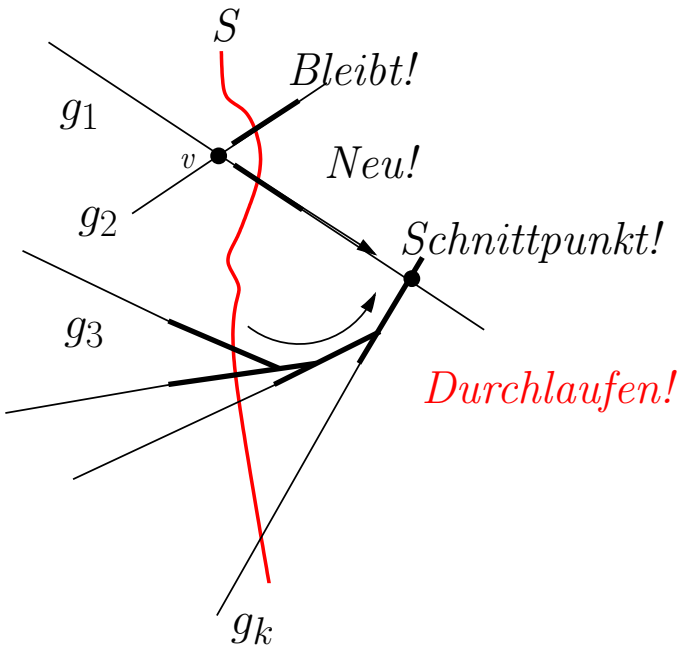
Kosten Aktualisierung T^o (T_u)

1. Nächster Knoten v!
2. Drüberheben
3. Schnittpunkt mit Kette darunter bestimmen



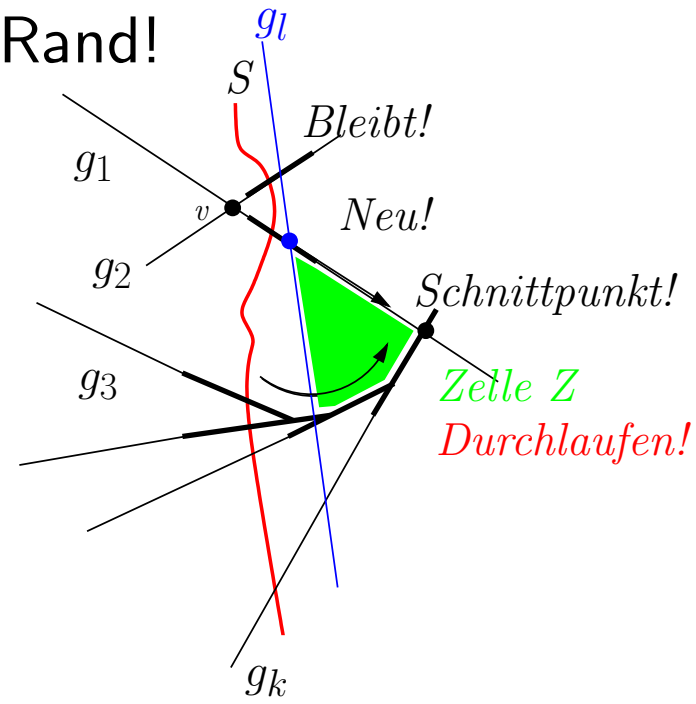
Kosten Aktualisierung T^o (T_u)

1. Nächster Knoten v!
2. Drüberheben
3. Schnittpunkt mit Kette darunter bestimmen
4. Einmal Kanten einer Zelle durchlaufen



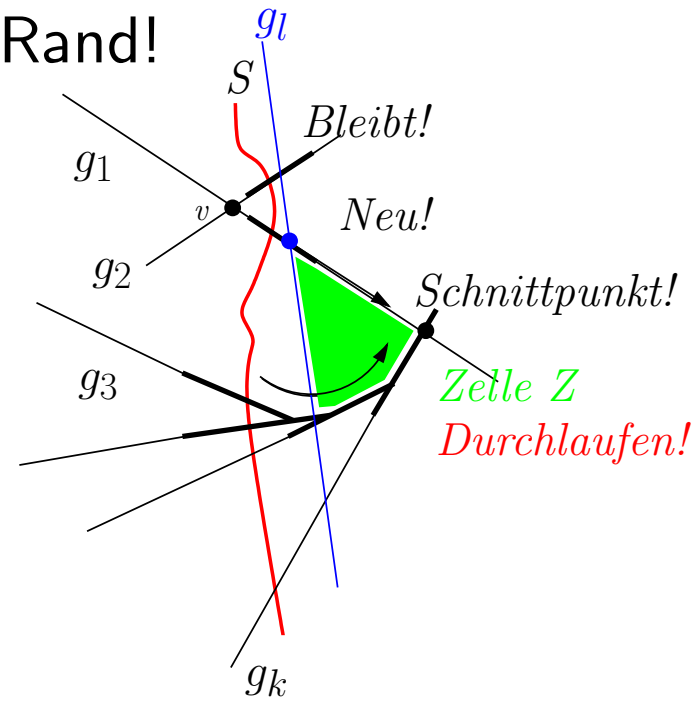
Kosten Aktualisierung T^o (T_u)

1. Nächster Knoten v!
2. Drüberheben
3. Schnittpunkt mit Kette darunter bestimmen
4. Einmal Kanten einer Zelle durchlaufen
5. Nicht alle auf dem Rand!



Kosten Aktualisierung T^o (T_u)

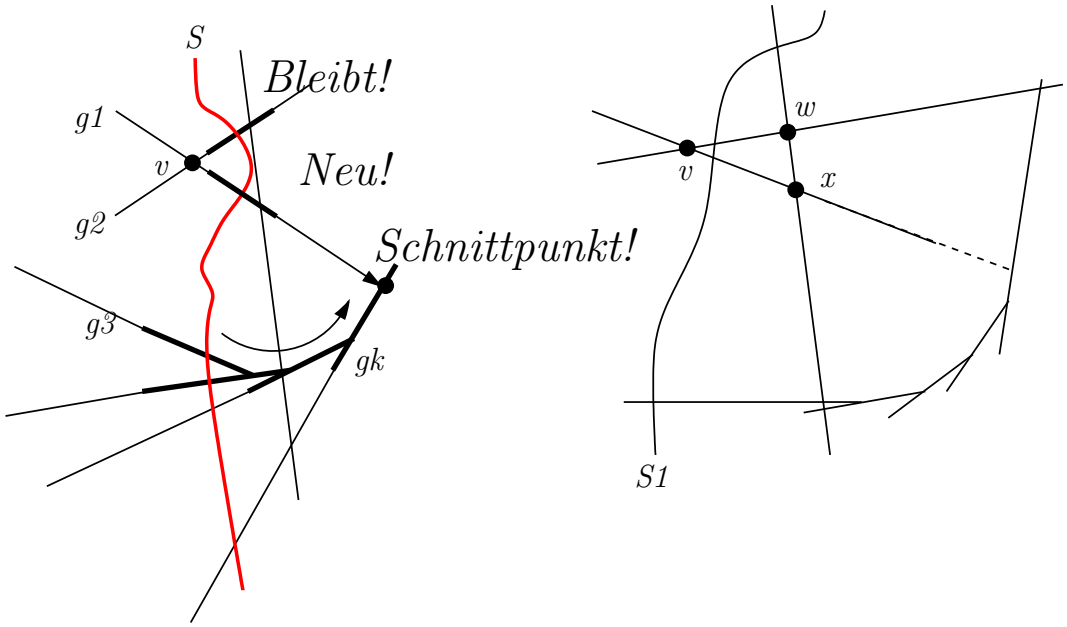
1. Nächster Knoten v!
2. Drüberheben
3. Schnittpunkt mit Kette darunter bestimmen
4. Einmal Kanten einer Zelle durchlaufen
5. Nicht alle auf dem Rand!



Gesamtkosten: Aktualisierung T^o (T_u)

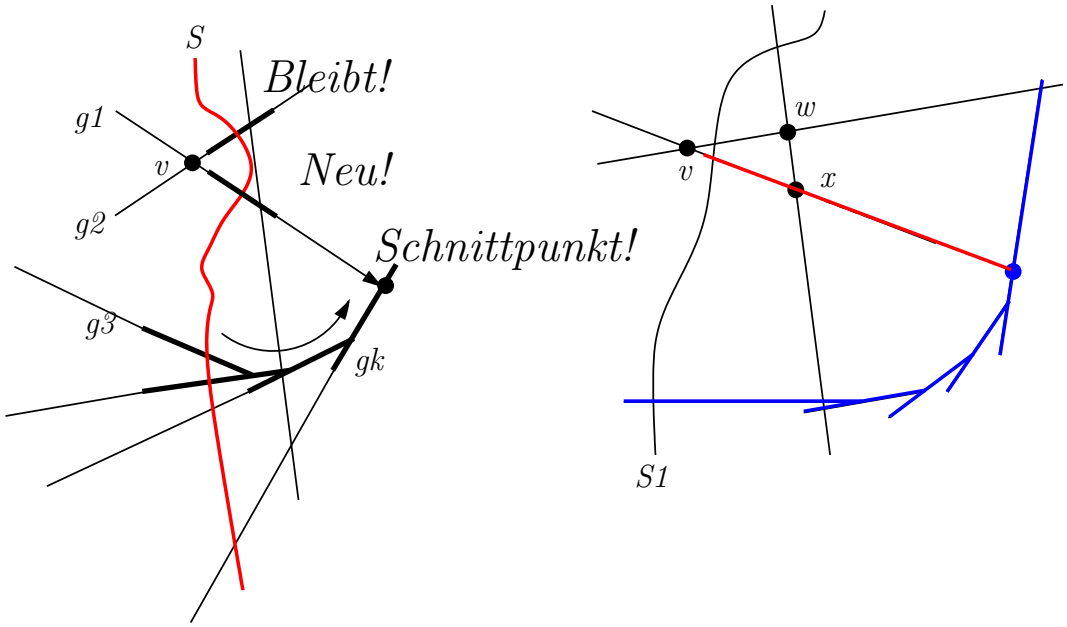
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung



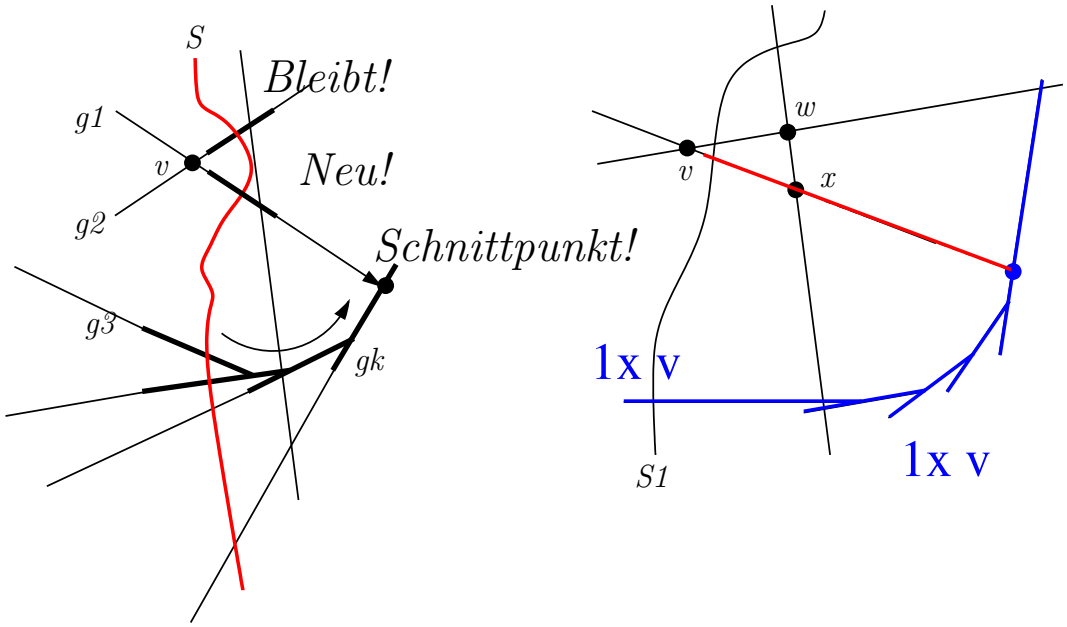
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle



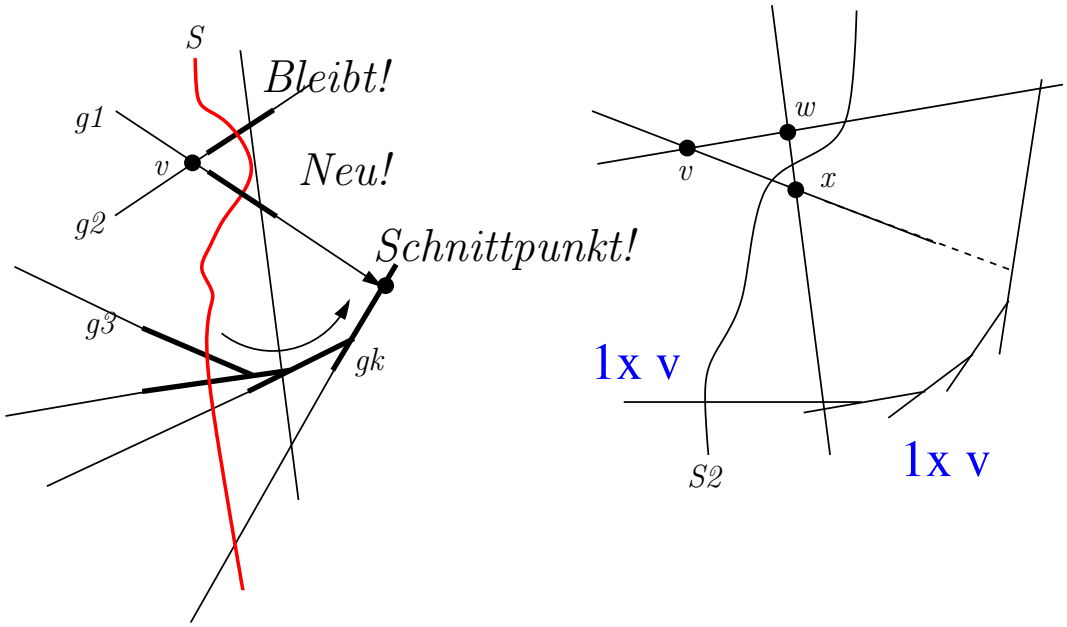
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



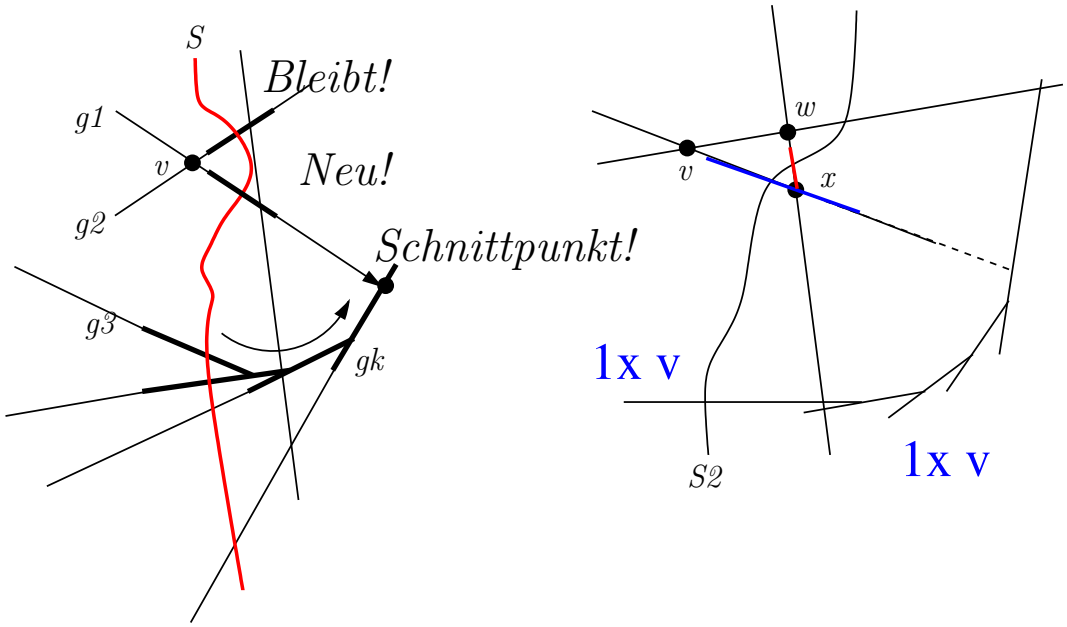
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



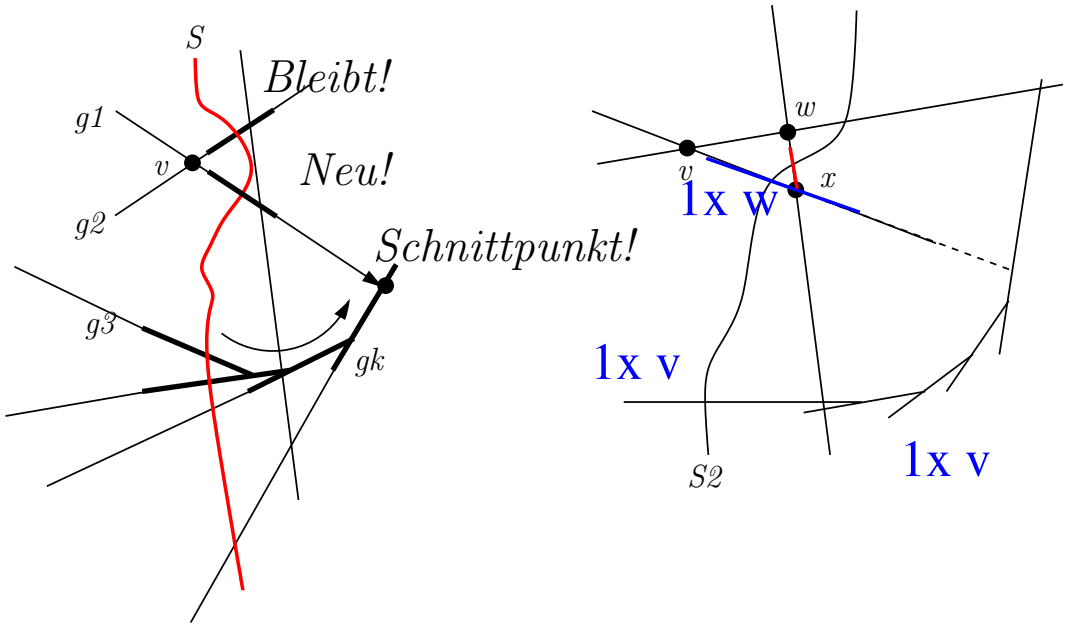
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



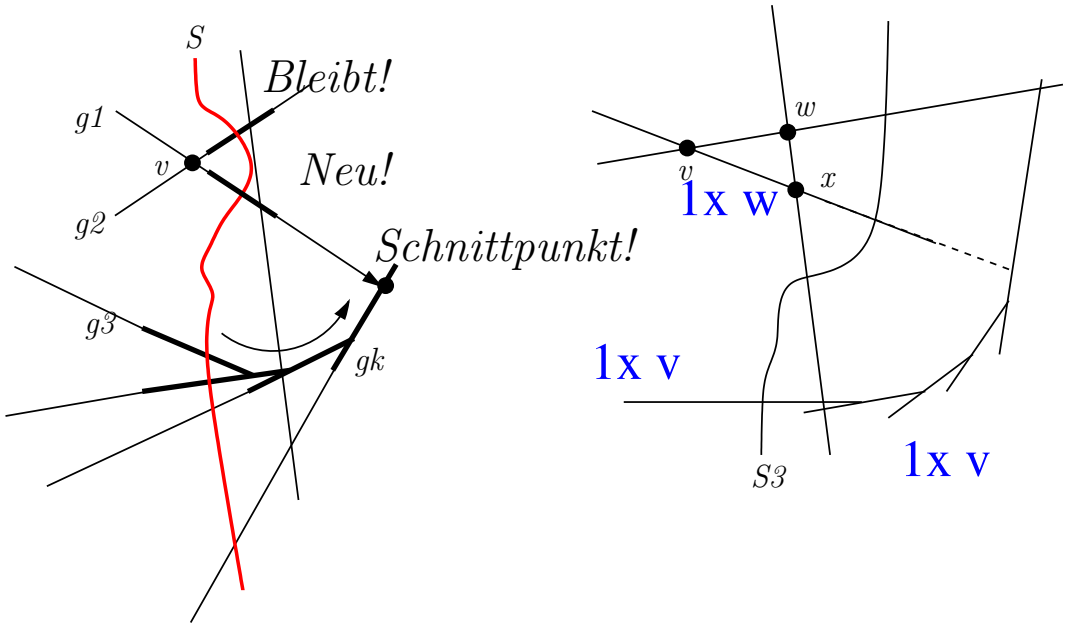
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



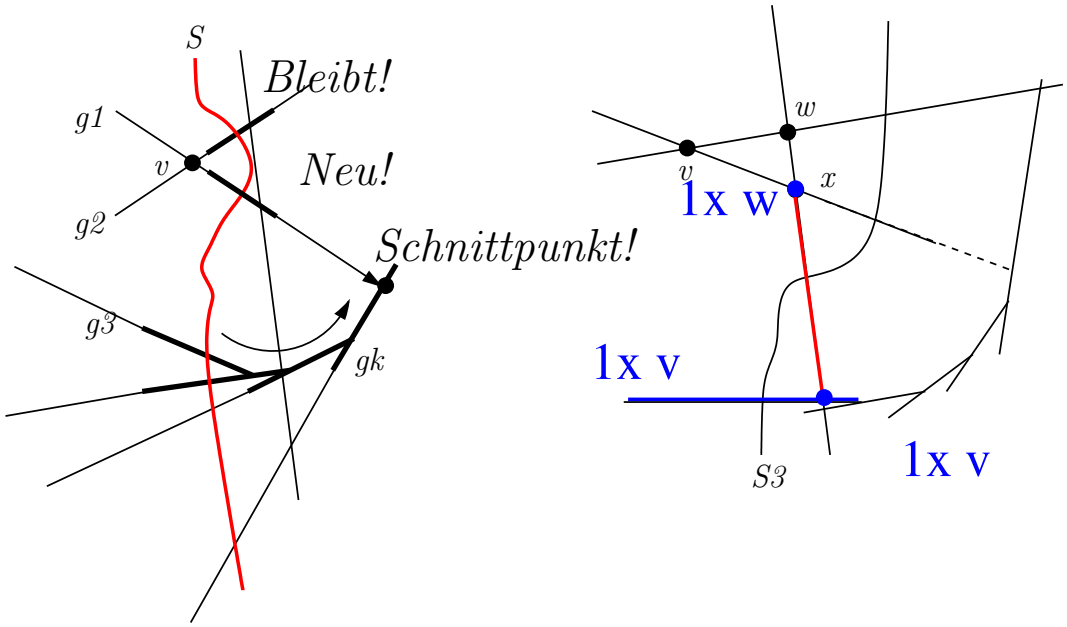
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



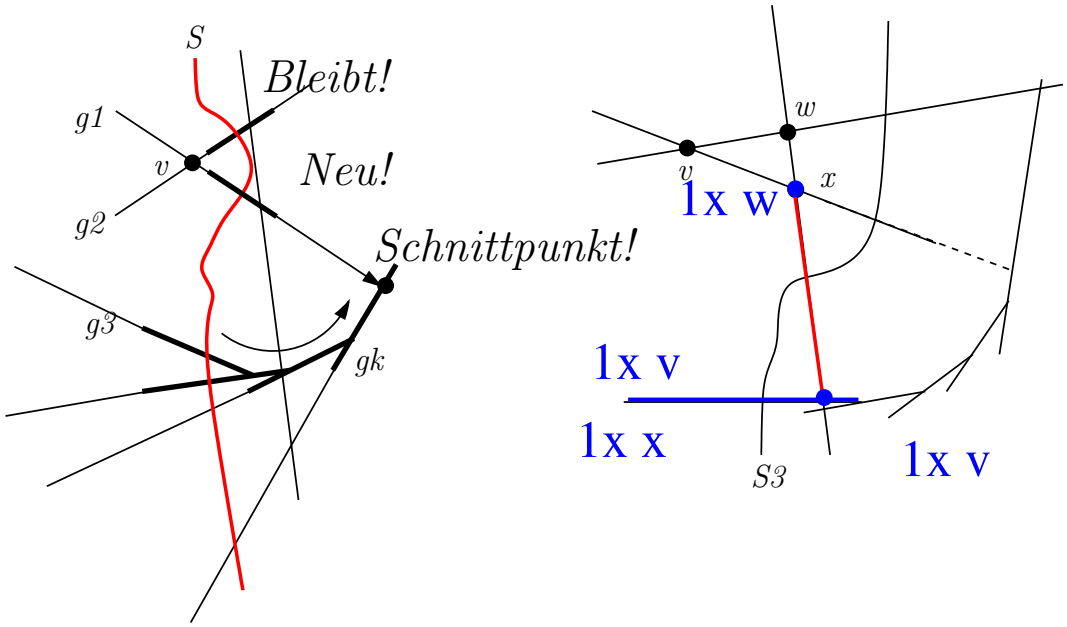
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



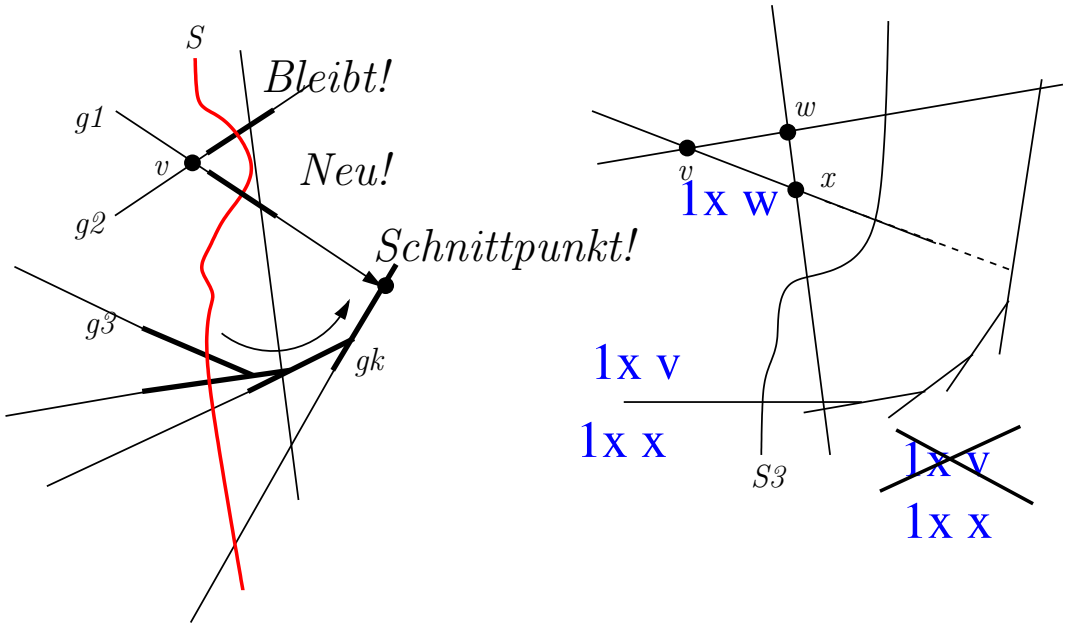
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



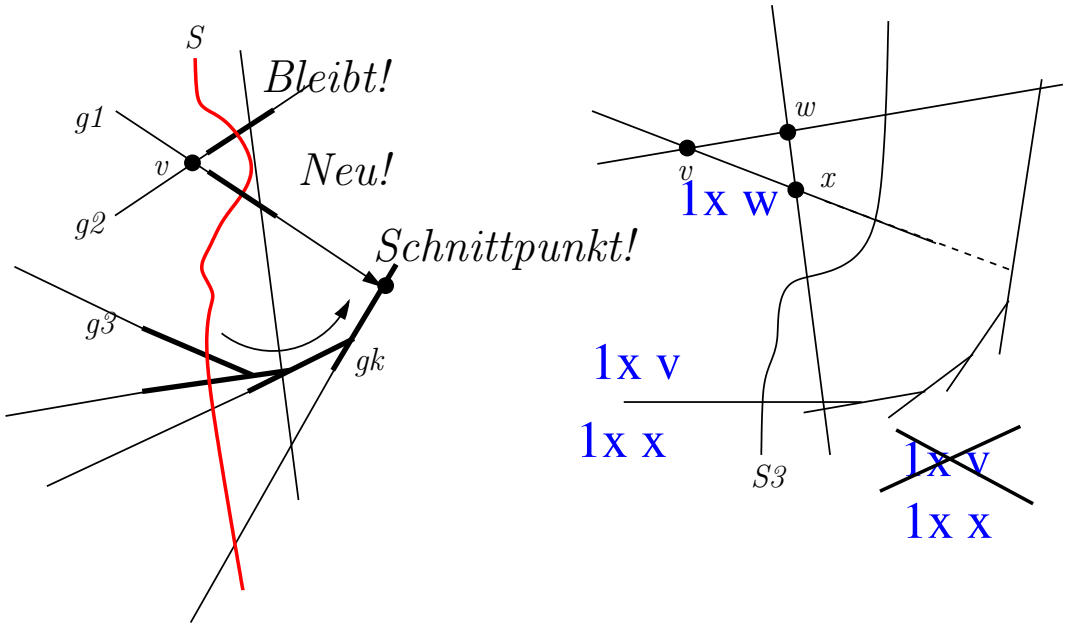
Gesamtkosten: Aktualisierung T^o (T_u)

1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



Gesamtkosten: Aktualisierung T^o (T_u)

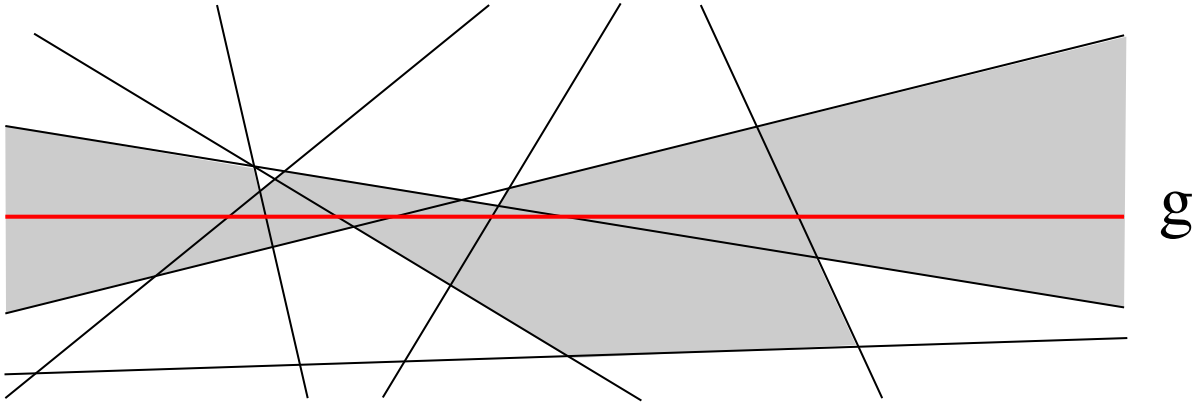
1. Geschickte Zuordnung
2. Gesamtzählweise: Pro Knoten alle Kanten einer Zelle
3. $\sum_{\text{Zelle } Z}(\text{Kanten } Z) \times (\text{Knoten } Z)$



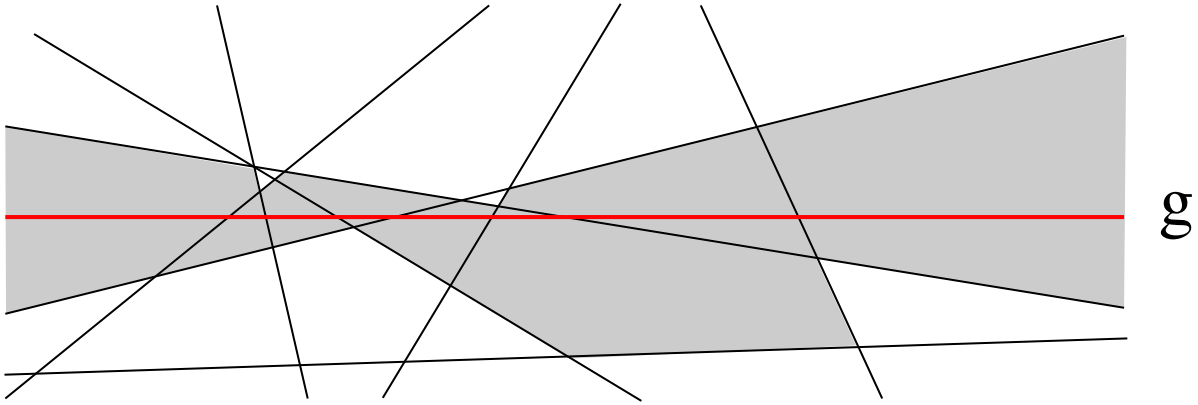
Andere Zählweise, bessere Zuordnung!

$$\sum_{\text{Zelle } Z} (\text{Kanten } Z) \times (\text{Knoten } Z) \leq$$

$$2 \sum_{\text{Gerade } g} \left(\sum_{g \text{ schneidet } Z} (\text{Kanten } Z) \right)$$

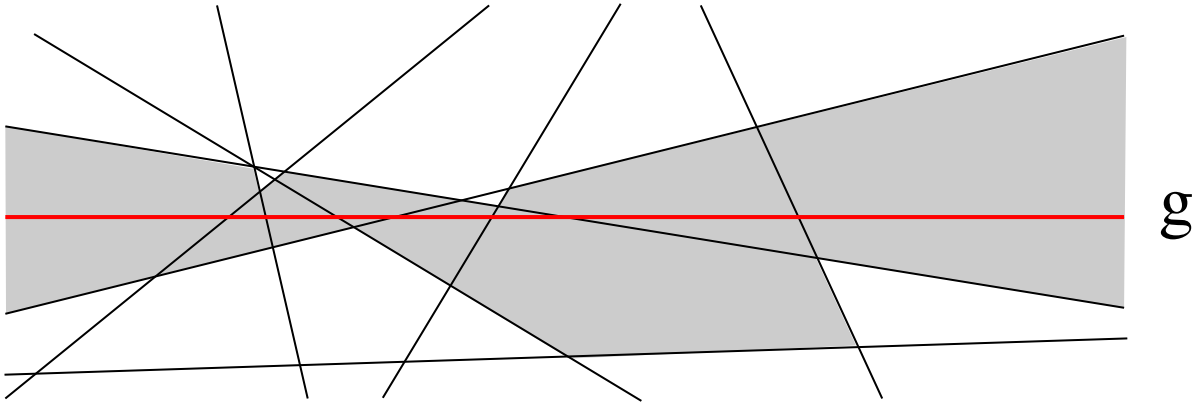


Komplexität Zone einer Geraden g



Komplexität Zone einer Geraden g

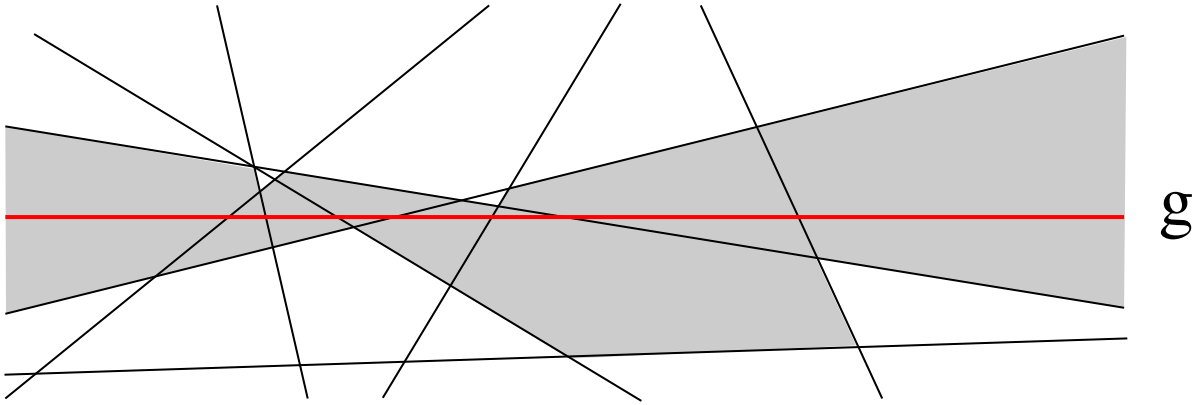
$$\left(\sum_{g \text{ schneidet } Z} (\text{Kanten } Z) \right)$$



Komplexität Zone einer Geraden g

$$\left(\sum_{g \text{ schneidet } Z} (\text{Kanten } Z) \right)$$

n-mal diese Kosten, jede Gerade



Zone(l) in O(n)

Zone(l) in O(n)

- n nicht-senkrechte Geraden, Horizontale l

Zone(l) in O(n)

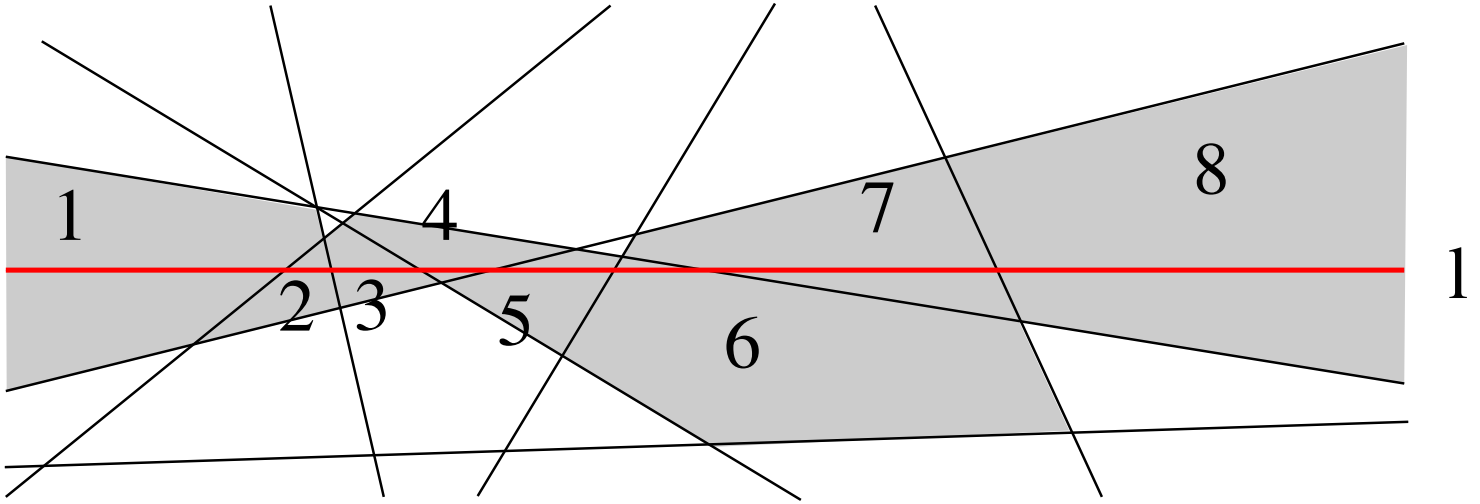
- n nicht-senkrechte Geraden, Horizontale l
- **Def. 1.3:** $\texttt{Zone}(l) := \{\text{Kanten der Zelle } Z | \overline{Z} \cap l \neq \emptyset\}$

Zone(l) in O(n)

- n nicht-senkrechte Geraden, Horizontale l
- **Def. 1.3:** $\texttt{Zone}(l) := \{\text{Kanten der Zelle } Z | \overline{Z} \cap l \neq \emptyset\}$
- **Theorem 1.4:** Komplexität von $\texttt{Zone}(l)$ liegt in $O(n)$

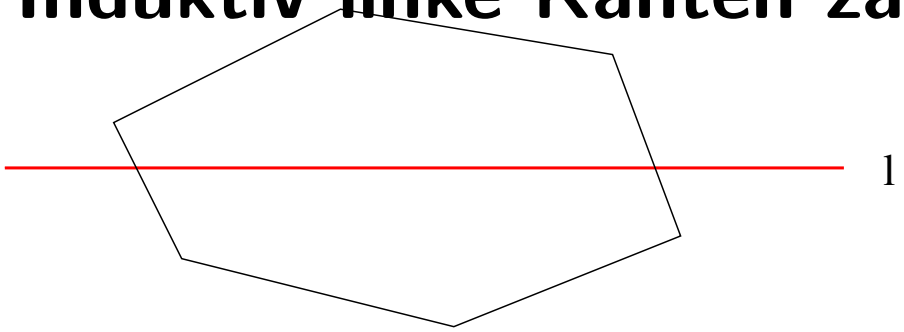
Zone(l) in O(n)

- n nicht-senkrechte Geraden, Horizontale l
- **Def. 1.3:** $\texttt{Zone}(l) := \{\text{Kanten der Zelle } Z | \overline{Z} \cap l \neq \emptyset\}$
- **Theorem 1.4:** Komplexität von $\texttt{Zone}(l)$ liegt in $O(n)$

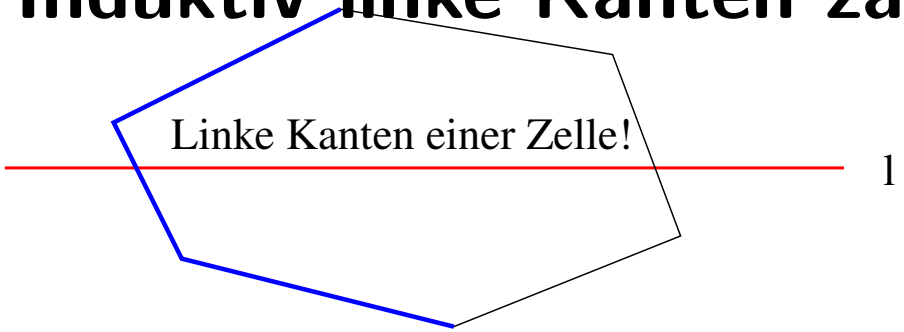


Induktiv linke Kanten zählen

Induktiv linke Kanten zählen

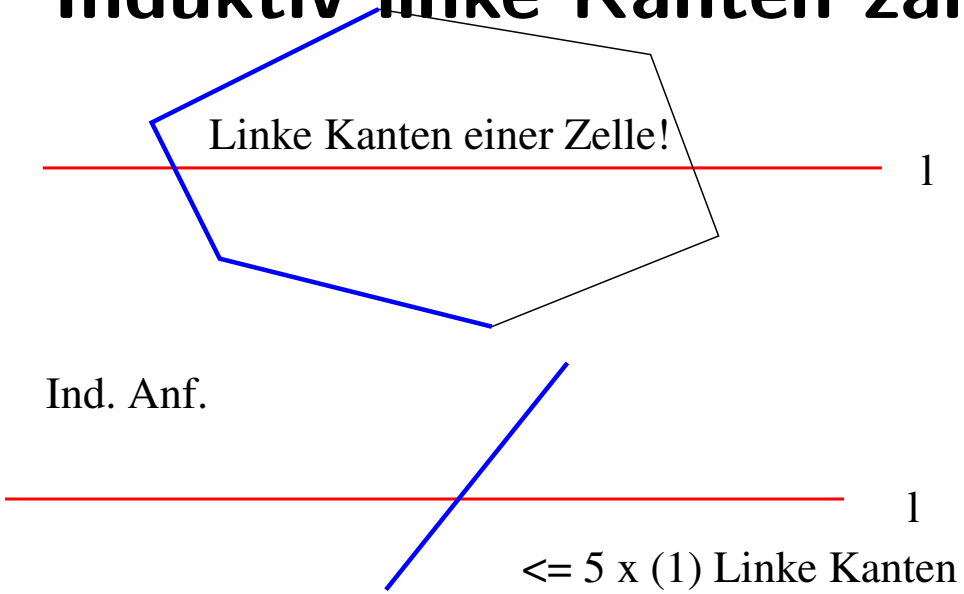


Induktiv linke Kanten zählen

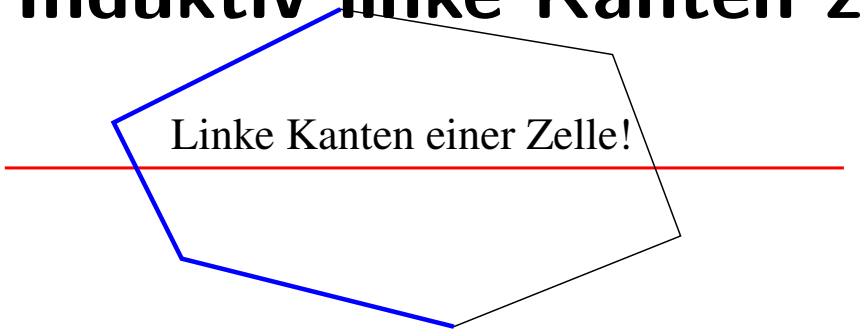


Induktiv linke Kanten zählen

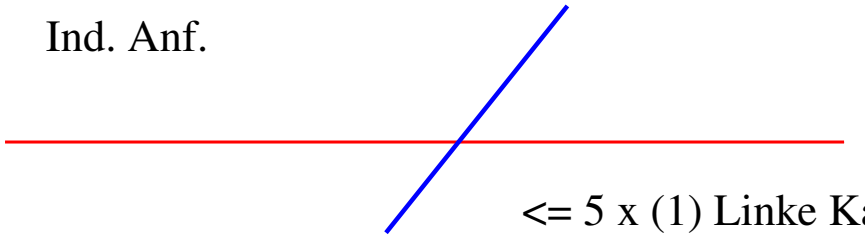
Induktiv linke Kanten zählen



Induktiv linke Kanten zählen

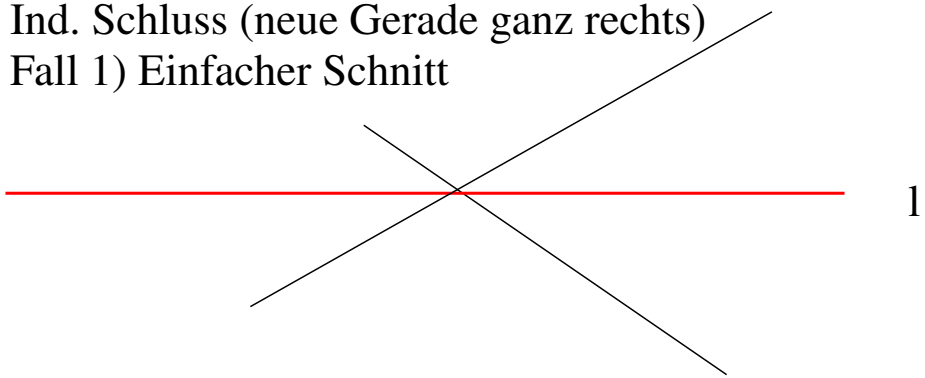


Ind. Anf.

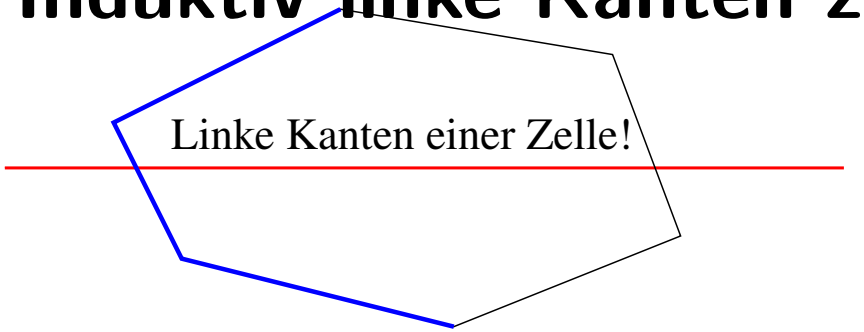


<= 5 x (1) Linke Kanten

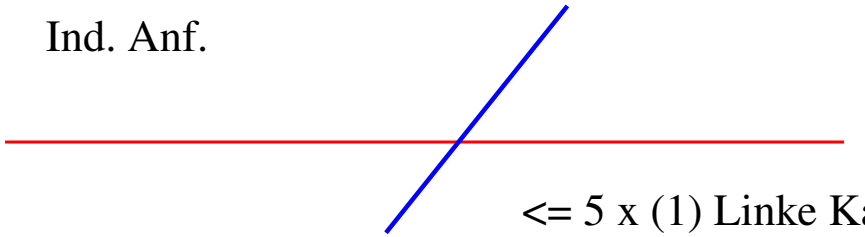
Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



Induktiv linke Kanten zählen

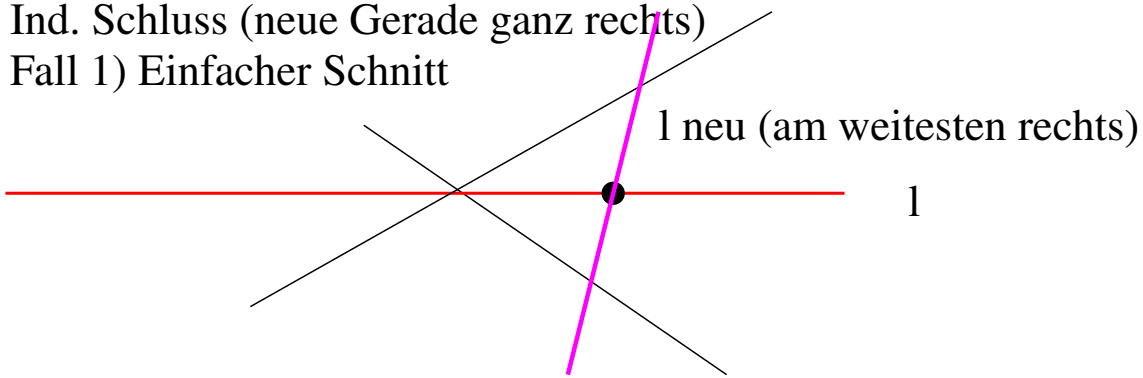


Ind. Anf.

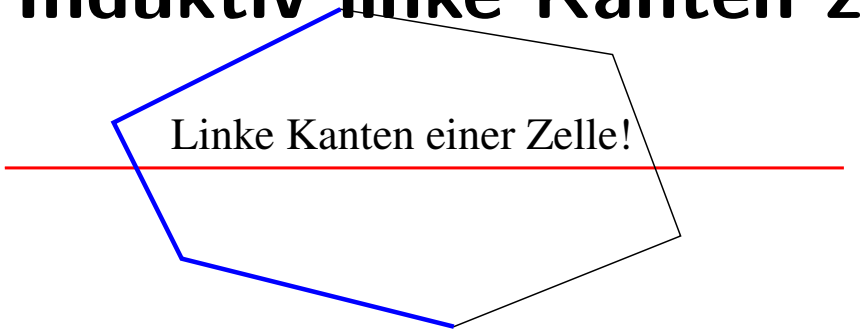


<= 5 x (1) Linke Kanten

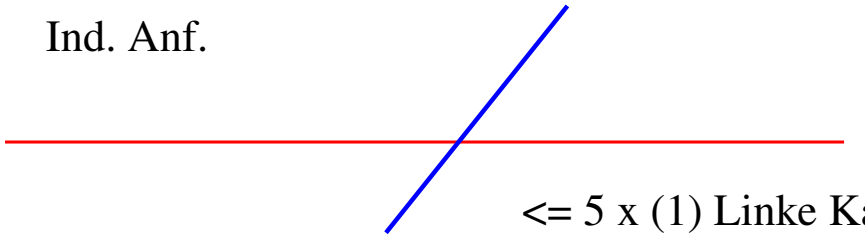
Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



Induktiv linke Kanten zählen

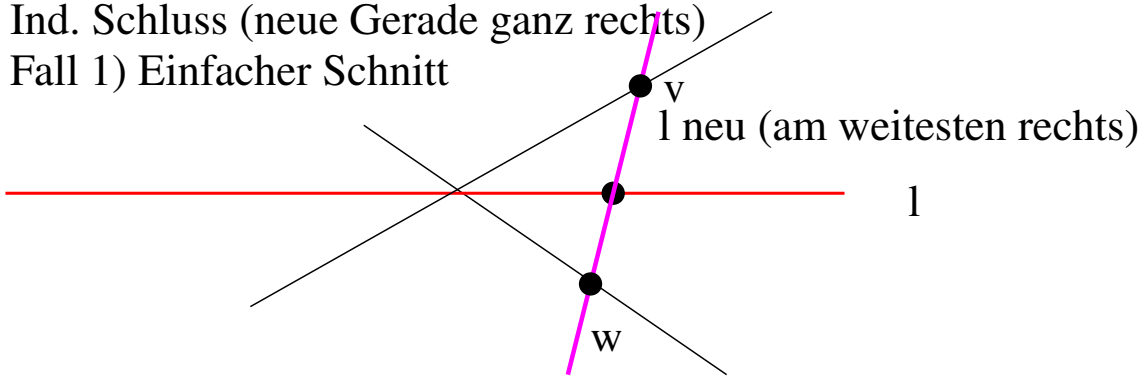


Ind. Anf.

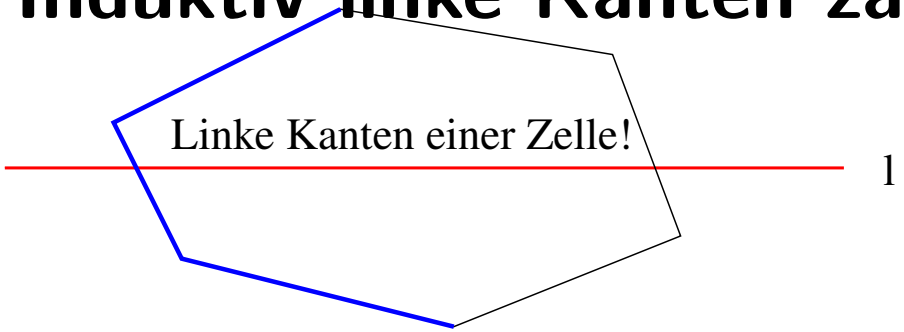


<= 5 x (1) Linke Kanten

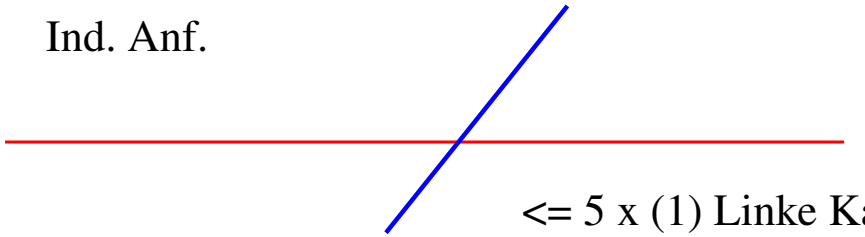
Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



Induktiv linke Kanten zählen

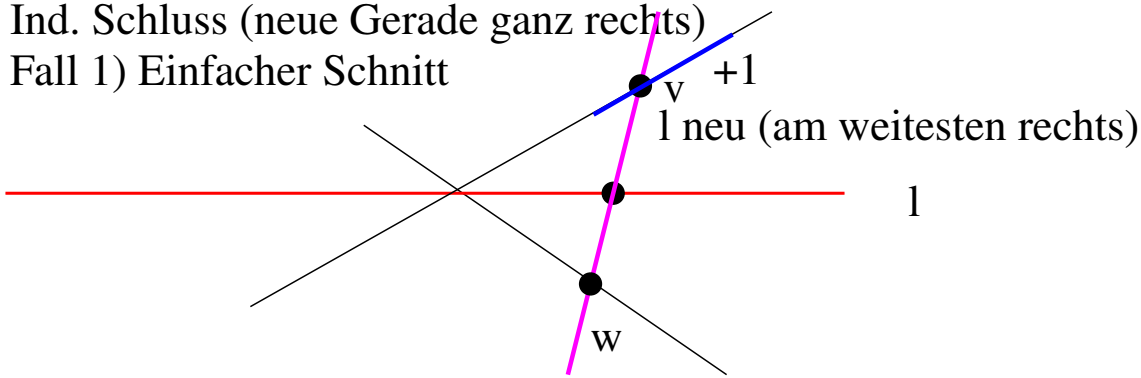


Ind. Anf.

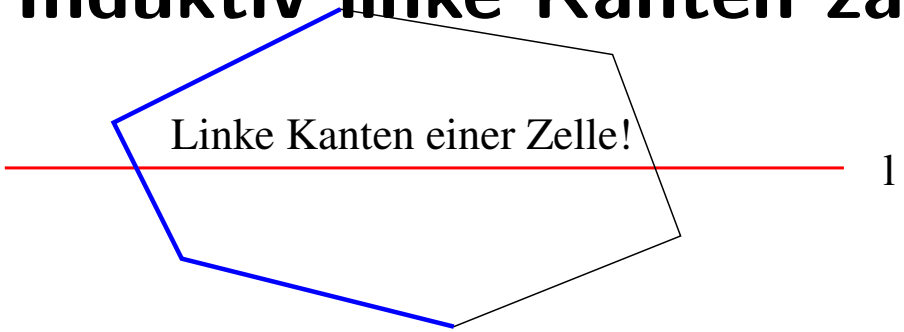


<= 5 x (1) Linke Kanten

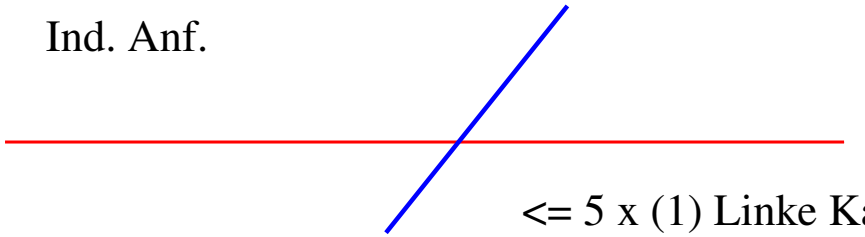
Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



Induktiv linke Kanten zählen

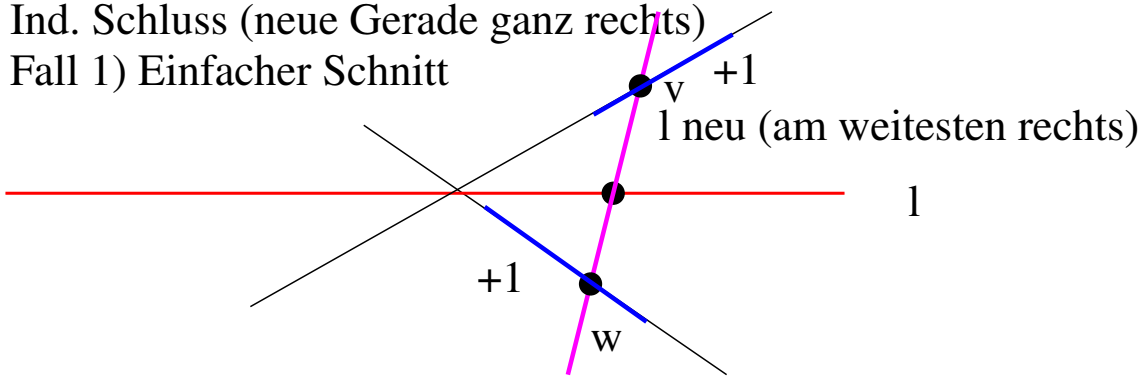


Ind. Anf.

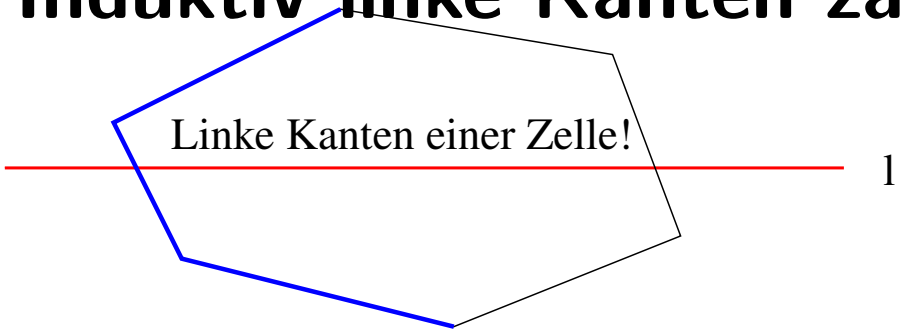


<= 5 x (1) Linke Kanten

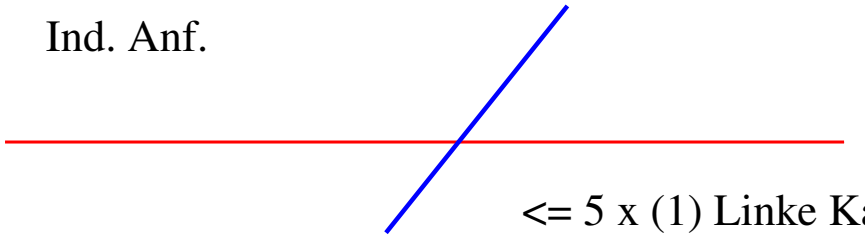
Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



Induktiv linke Kanten zählen

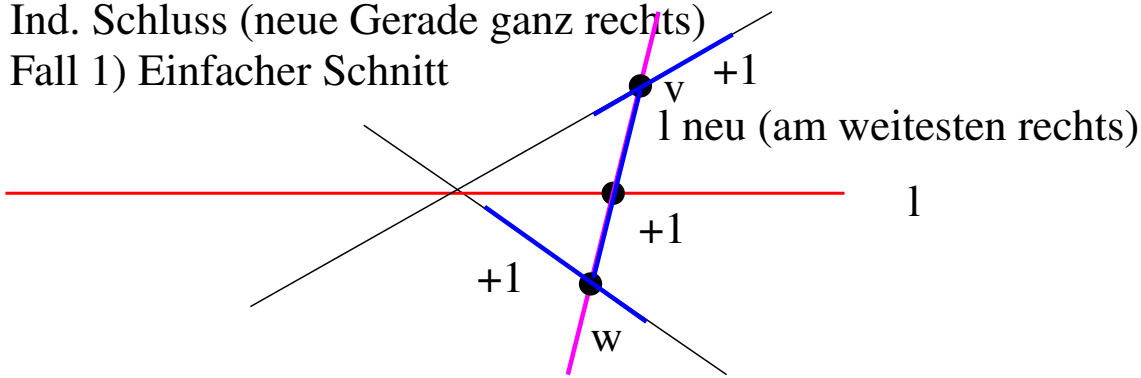


Ind. Anf.



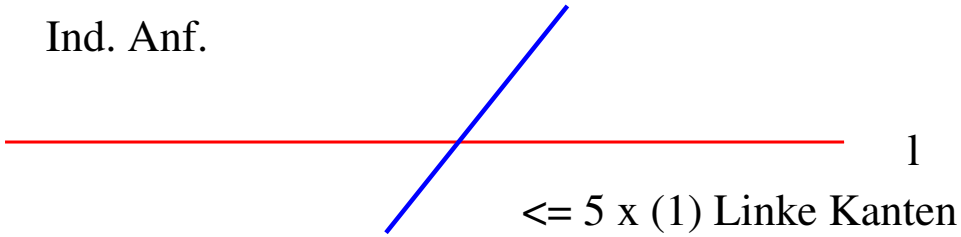
<= 5 x (1) Linke Kanten

Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



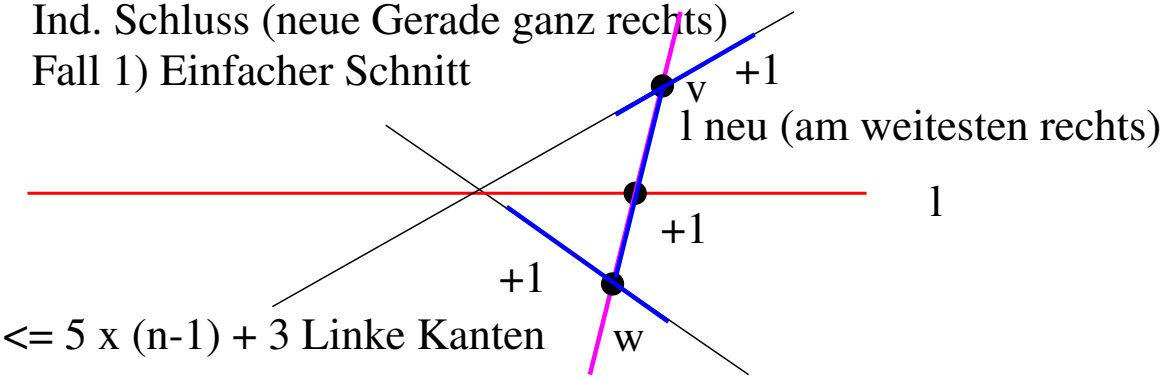
Induktiv linke Kanten zählen

Ind. Anf.



<= 5 x (1) Linke Kanten

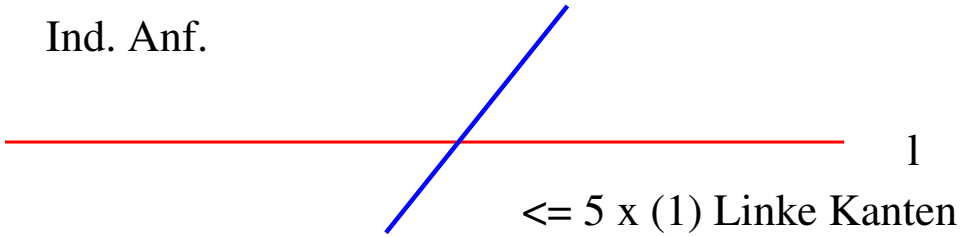
Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt



<= 5 x (n-1) + 3 Linke Kanten

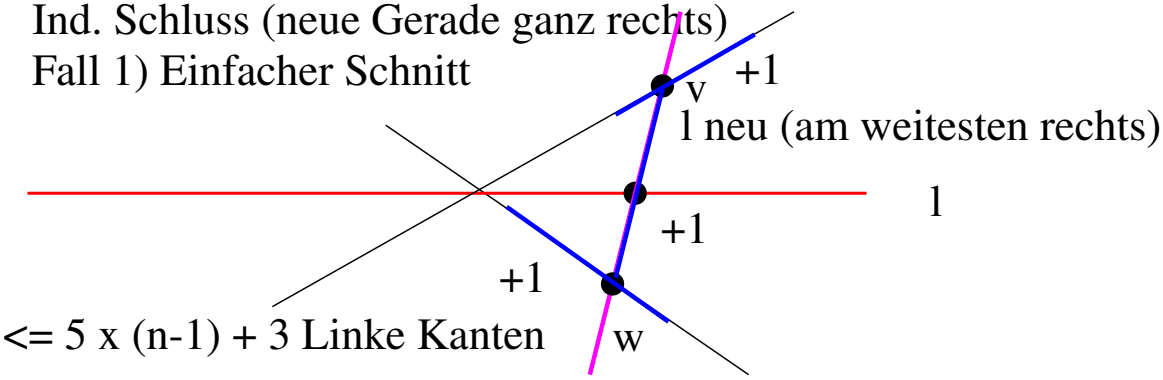
Induktiv linke Kanten zählen

Ind. Anf.



<= 5 x (1) Linke Kanten

Ind. Schluss (neue Gerade ganz rechts)
Fall 1) Einfacher Schnitt

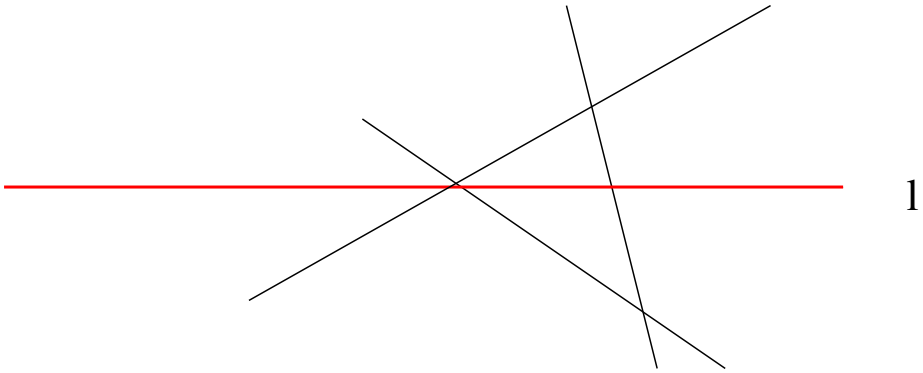


<= 5 x (n-1) + 3 Linke Kanten

Induktiv linke Kanten zählen

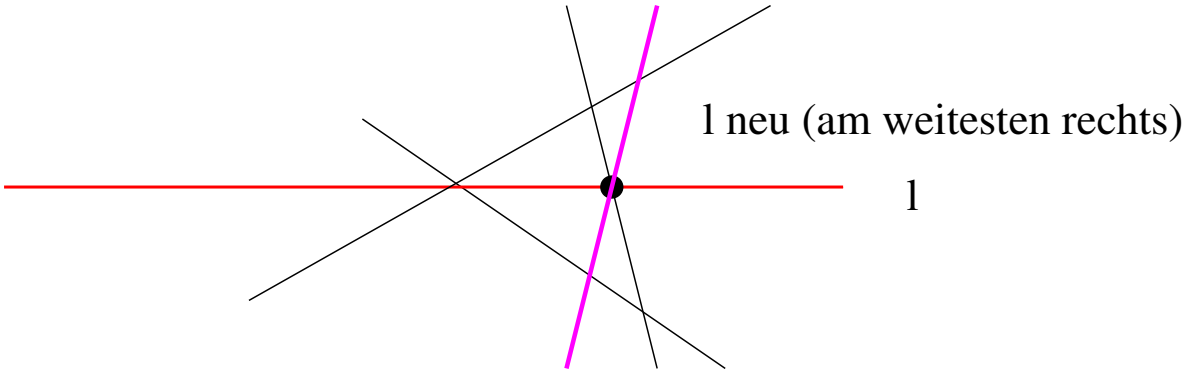
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



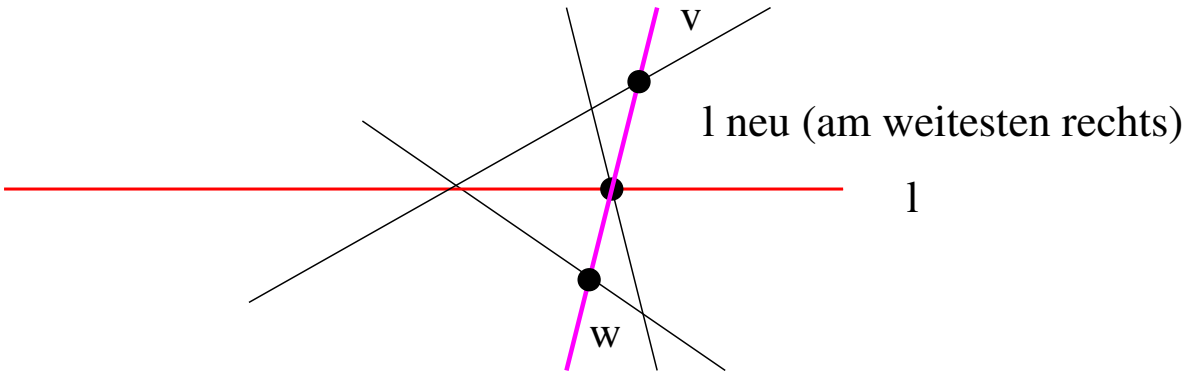
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



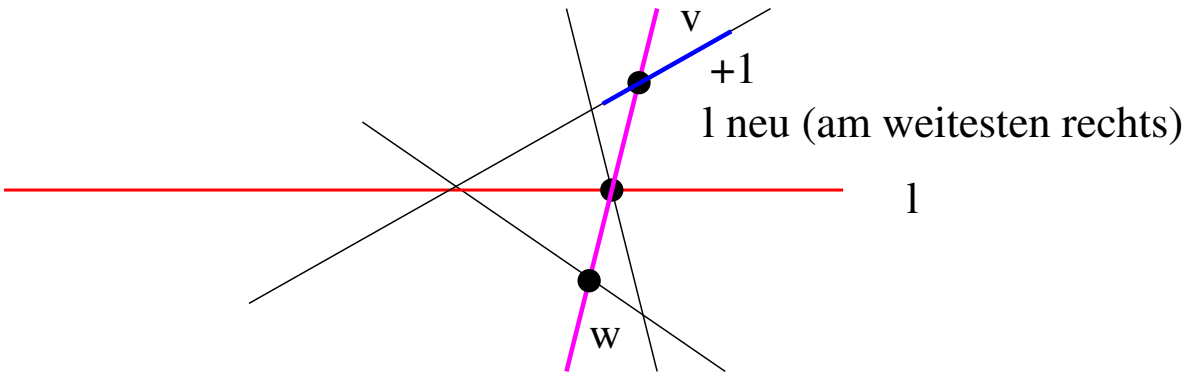
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



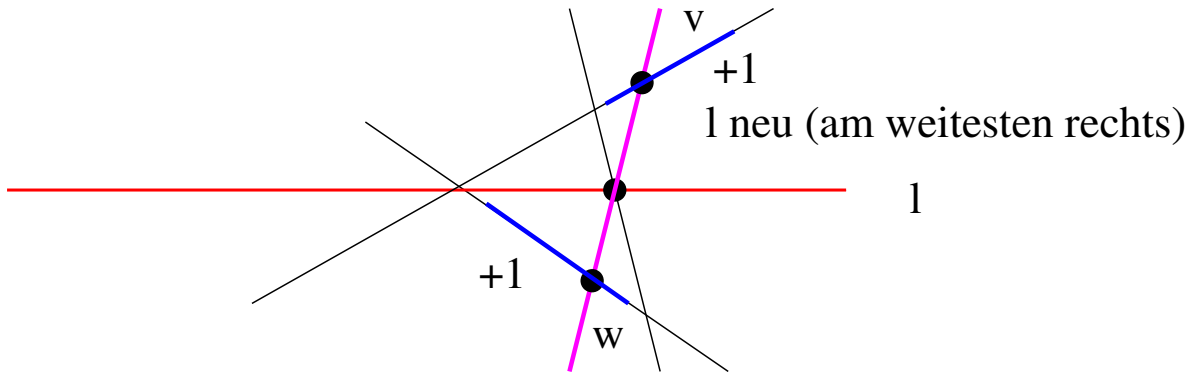
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



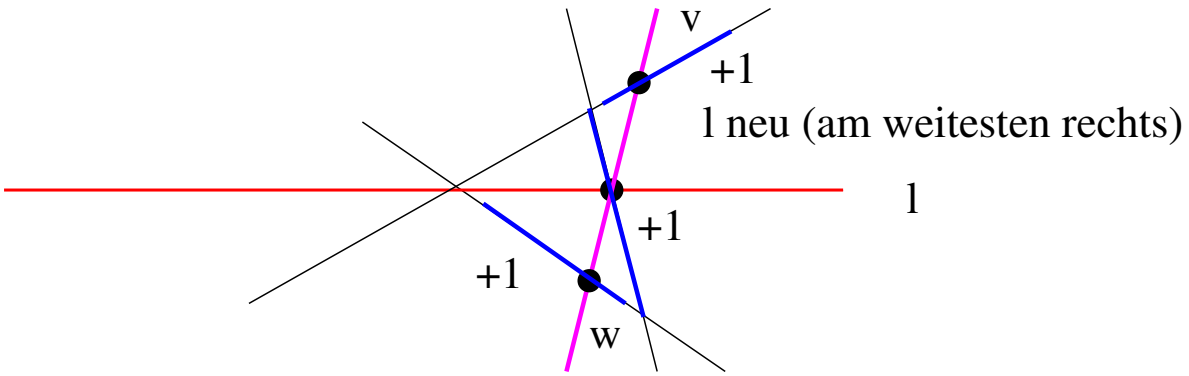
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



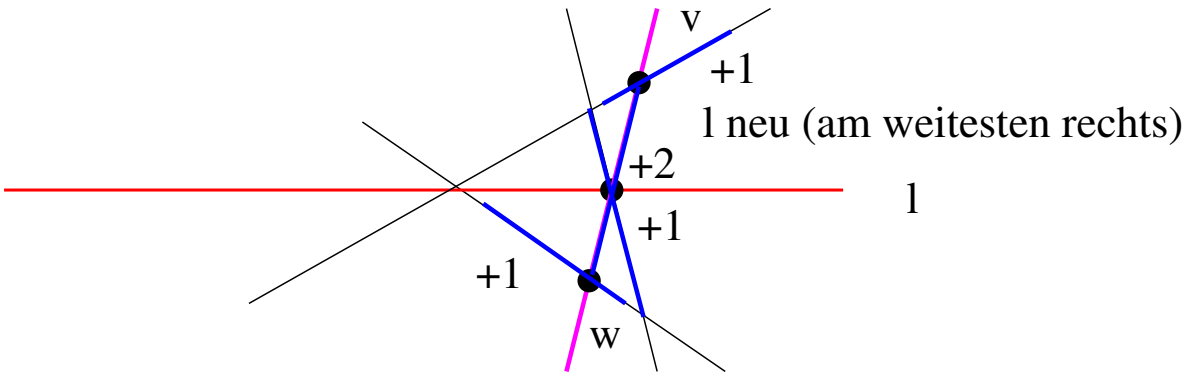
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



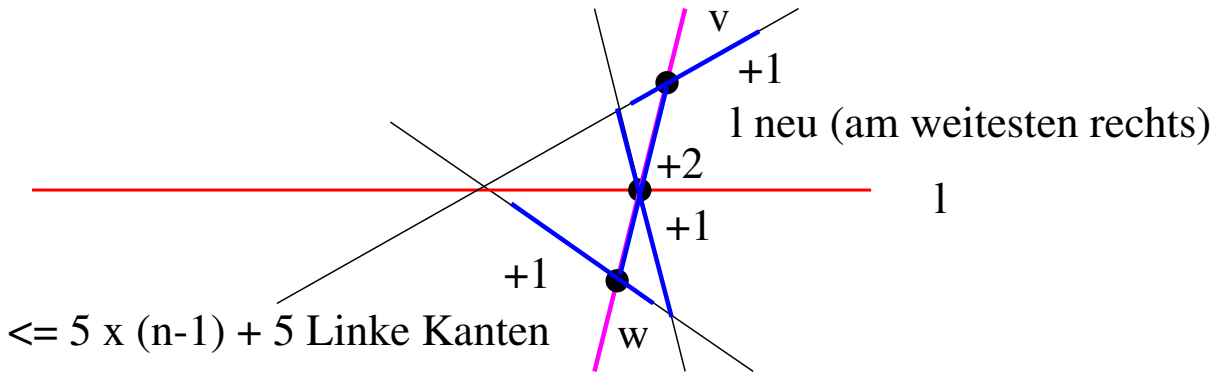
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



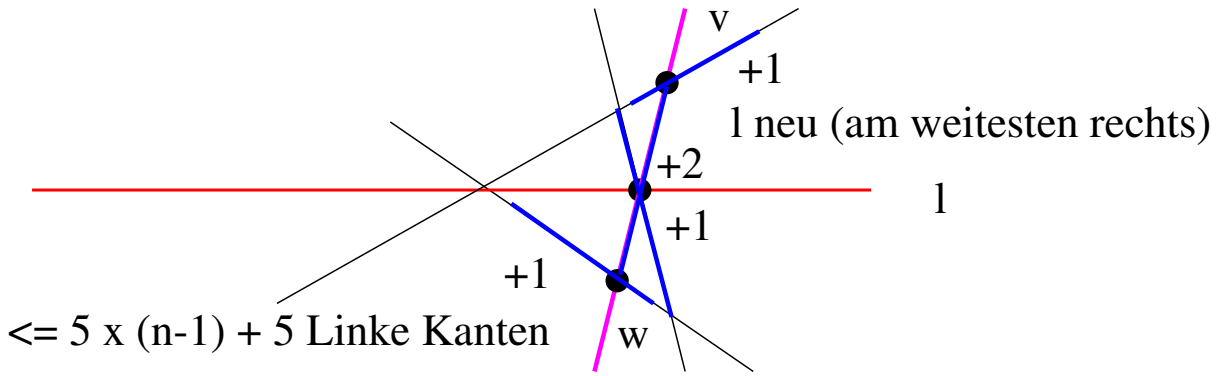
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



Induktiv linke Kanten zählen

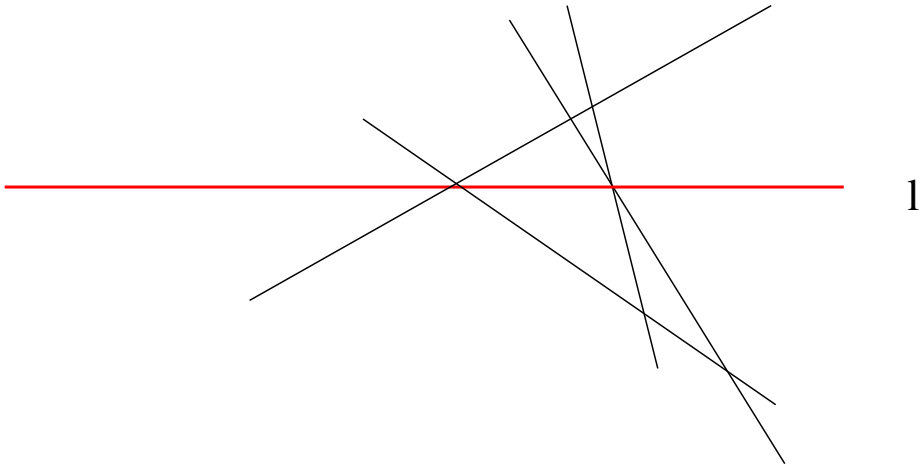
Ind. Schluss (WC)
Fall 2) Schnitt mit einer Kante



Induktiv linke Kanten zählen

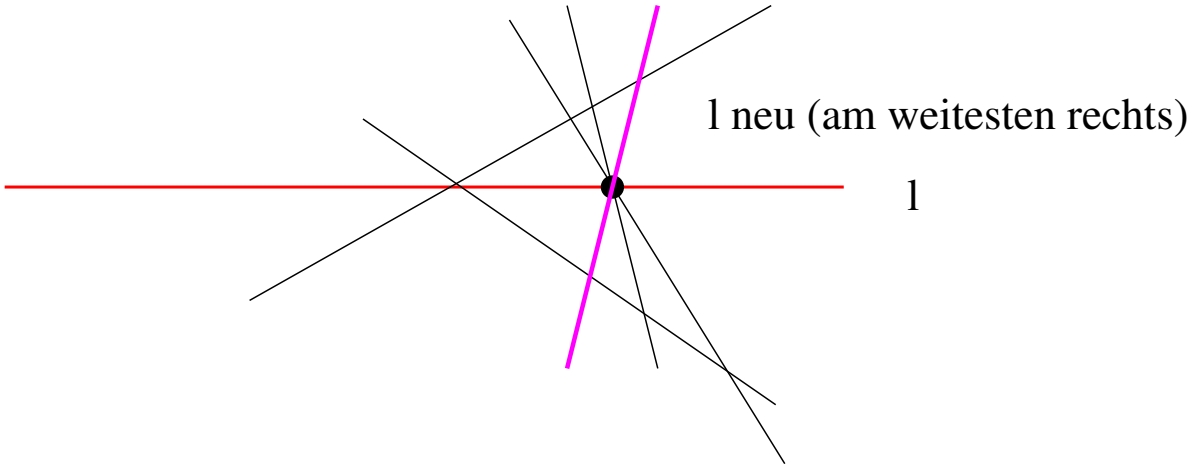
Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

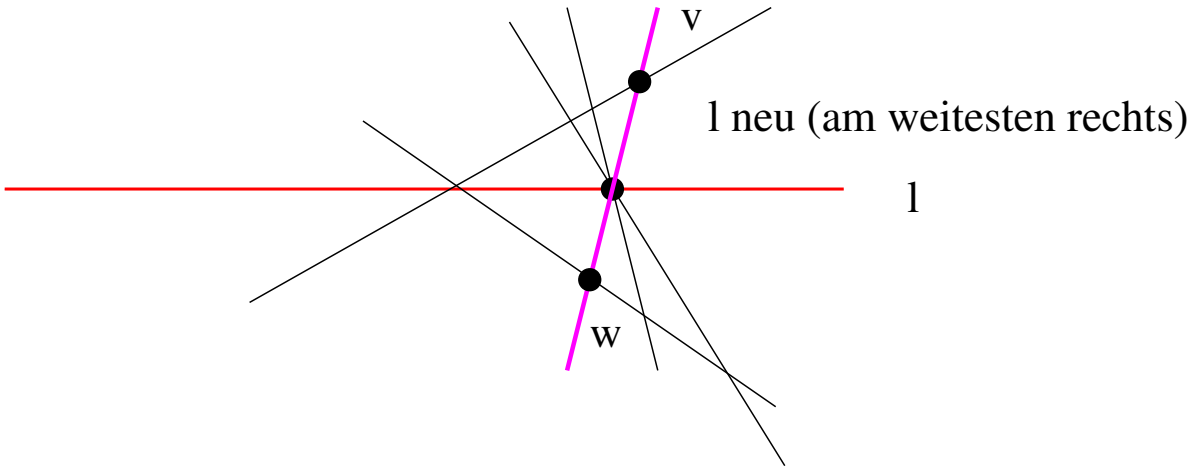
Ind. Schluss (WC)
Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

Ind. Schluss (WC)

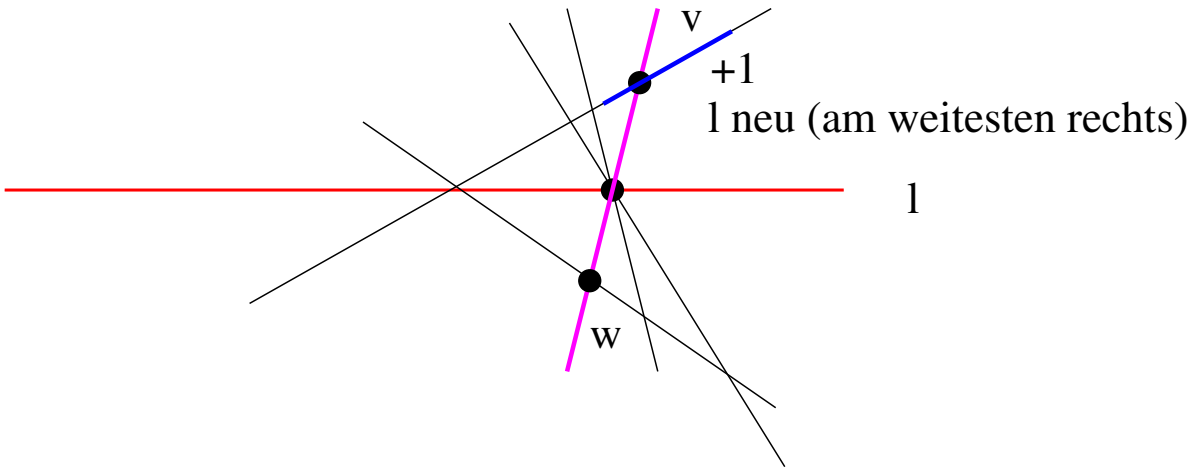
Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

Ind. Schluss (WC)

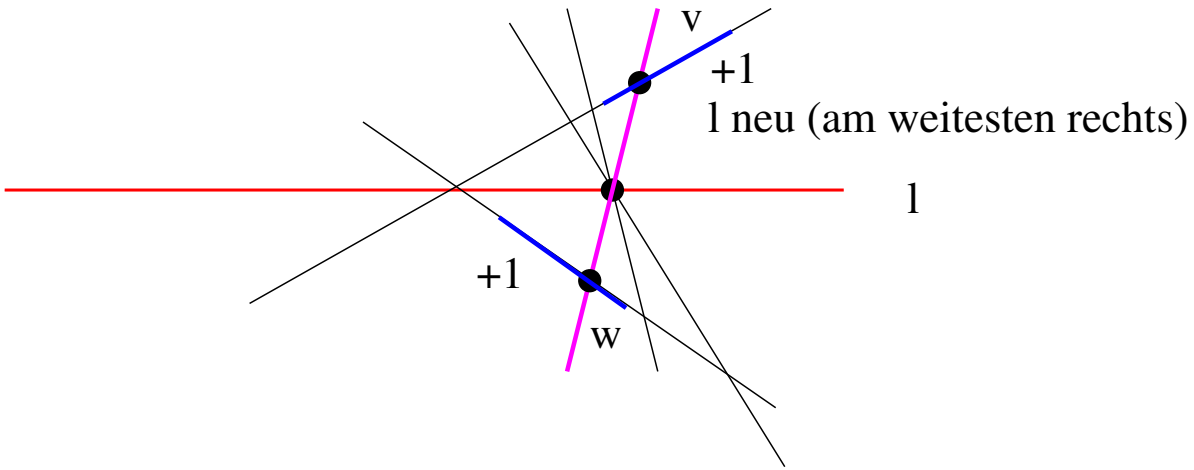
Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

Ind. Schluss (WC)

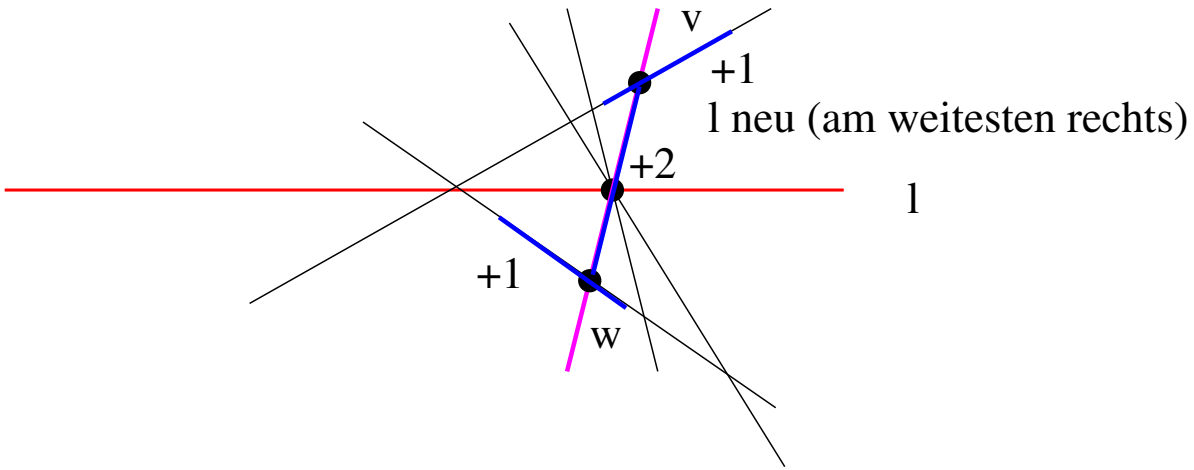
Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

Ind. Schluss (WC)

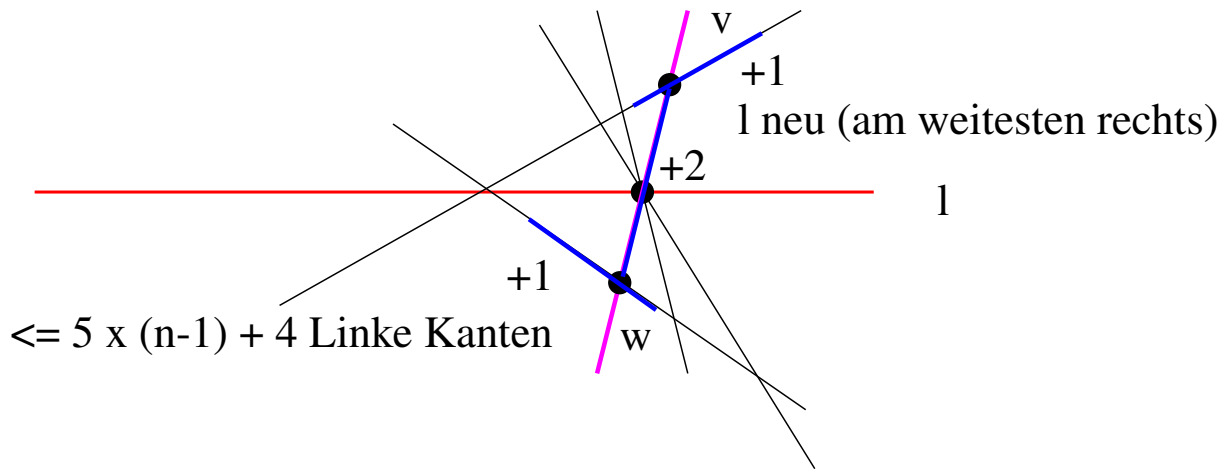
Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

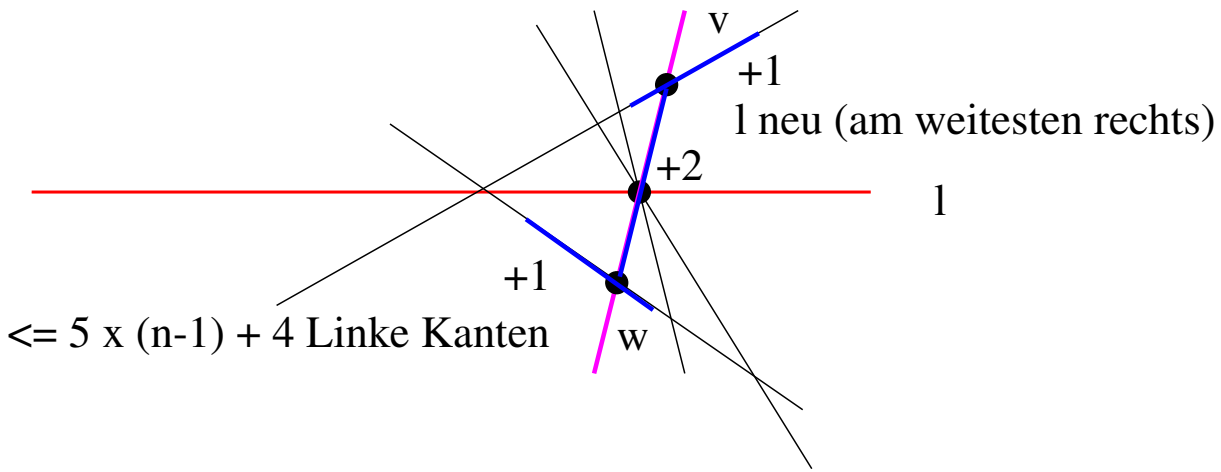
Ind. Schluss (WC)

Fall 3) Schnitt mit mahr als einer Kante



Induktiv linke Kanten zählen

Ind. Schluss (WC)
Fall 3) Schnitt mit mahr als einer Kante



Insgesamt nicht mehr als $5n$ linke Kanten!

Algorithmus Bearbeitungsreihenfolge

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume:

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$
- Kandidaten auswählen, Liste aktualisieren (Horizontbaum):

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$
- Kandidaten auswählen, Liste aktualisieren (Horizontbaum): $O(1)$

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$
- Kandidaten auswählen, Liste aktualisieren (Horizontbaum): $O(1)$
- Horizontbäume aktualisieren:

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$
- Kandidaten auswählen, Liste aktualisieren (Horizontbaum): $O(1)$
- Horizontbäume aktualisieren: $O(n^2)$

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$
- Kandidaten auswählen, Liste aktualisieren (Horizontbaum): $O(1)$
- Horizontbäume aktualisieren: $O(n^2)$
- Reihenfolge des Überschreitens: Bearbeitungsreihenfolge,

Algorithmus Bearbeitungsreihenfolge

- Init.: Nachbarliste, Kandidatenliste, Horizontbäume: $O(n^2)$
- Kandidaten auswählen, Liste aktualisieren (Horizontbaum): $O(1)$
- Horizontbäume aktualisieren: $O(n^2)$
- Reihenfolge des Überschreitens: Bearbeitungsreihenfolge, $O(n^2)$

Zusammenfassung Gesamtalgorithmus

- Horizontbäume aktualisieren: $O(n^2)$
- Topol. Sweep: $O(n^2)$
- Bearbeitungsreihenfolge: $O(n^2)$
- Sweep Sichtbarkeitsgraph: $O(n^2)$
- **Theorem 1.5::** Sichtbarkeitsgraph von n Liniensegmenten in $O(n^2)$ berechenbar. Speicherbedarf $O(n)$

Kürzeste Pfade in gewichteten Graph (Dijkstra, 1959)

Gegeben: Graph $G = (V, E)$ zusammenhängend, $|V| = n$, Kantengewichtung $g : E \longrightarrow R^{\geq 0}$, Knoten $s, t \in V$.

Gesucht: Kantenfolge von s nach t mit minimalem Gesamtgewicht.

Solange $A \neq \emptyset$:

- Entnehme A ein p mit minimalen $d(p)$.
- $W := W \cup \{p\}$.
- Für alle direkten Nachbarn q von p in W^C:
  - $d(q) := \min \left\{ d(q),\ d(p) + g(p, q) \right\}$
  - Wenn $q \notin A$ dann $A := A \cup \{q\}$.

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W,

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n$)

- Welle W, Ausläufer A

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n$)

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n$)

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n$)

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$

Dijkstra auf (V, E)**,** $|E| = m \in O(n^2)$**,** $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$

Dijkstra auf (V, E)**,** $|E| = m \in O(n^2)$**,** $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n$)

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps
- 1) in $O(1)$,

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps
- 1) in $O(1)$, 2) in $O(\log n)$,

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps
- 1) in $O(1)$, 2) in $O(\log n)$, 3) in $O(1)$

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps
- 1) in $O(1)$, 2) in $O(\log n)$, 3) in $O(1)$
- Laufzeit: $O(n \log n + m)$,

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps
- 1) in $O(1)$, 2) in $O(\log n)$, 3) in $O(1)$
- Laufzeit: $O(n \log n + m)$, Optimal?

Dijkstra auf (V, E), $|E| = m \in O(n^2)$, $|V| = n)$

- Welle W, Ausläufer A
- Priority-Queue für A mit $|A| \in O(n)$
- DS für A:

  1. Entnahme Min: $n - mal$
  2. Entferne Min: $n - mal$
  3. Anpassen Schlüssel: $m - mal$

- Beste Wahl: Fibbonacci Heaps
- 1) in $O(1)$, 2) in $O(\log n)$, 3) in $O(1)$
- Laufzeit: $O(n \log n + m)$, Optimal?
- Andere DS, z.B. Baum: $O((n + m) \log n)$, 3) in $O(\log n)$

Ergebnis!!

Ergebnis!!

Theorem: Kürzester Weg in polygonaler Szene P mit $|P| = n$ von s nach t kann in Zeit $O(n^2)$ berechnet werden.

Ergebnis!!

Theorem: Kürzester Weg in polygonaler Szene P mit $|P| = n$ von s nach t kann in Zeit $O(n^2)$ berechnet werden.

Beweis:

Ergebnis!!

Theorem: Kürzester Weg in polygonaler Szene P mit $|P| = n$ von s nach t kann in Zeit $O(n^2)$ berechnet werden.

Beweis: Sichtbarbeitsgraph $O(n^2)$,
Dijkstra: $O(n \log n + n^2) \in O(n^2)$

Untere Schranke!

Untere Schranke!

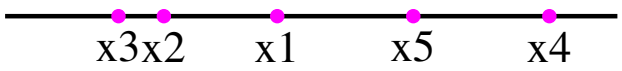
- Sortieren reduzieren auf kürzesten Weg berechnen!

Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene

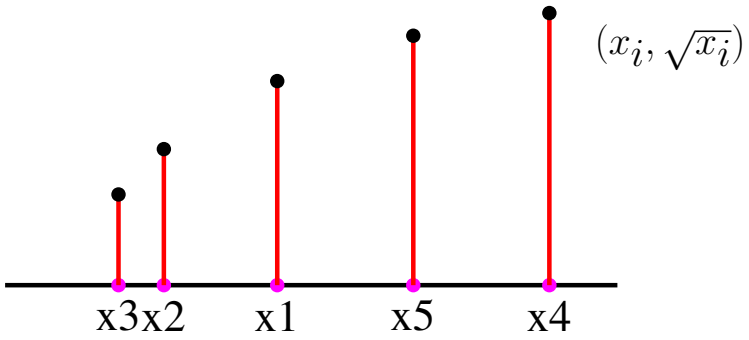
Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



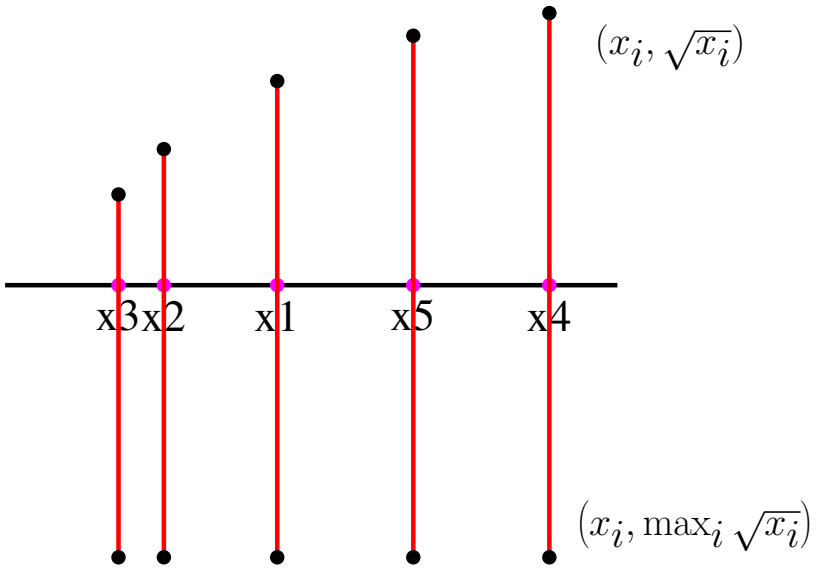
Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



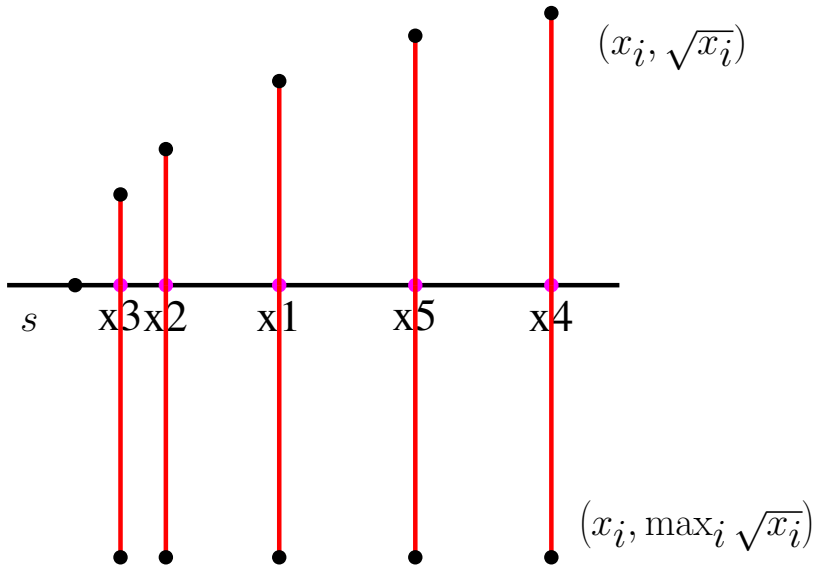
Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



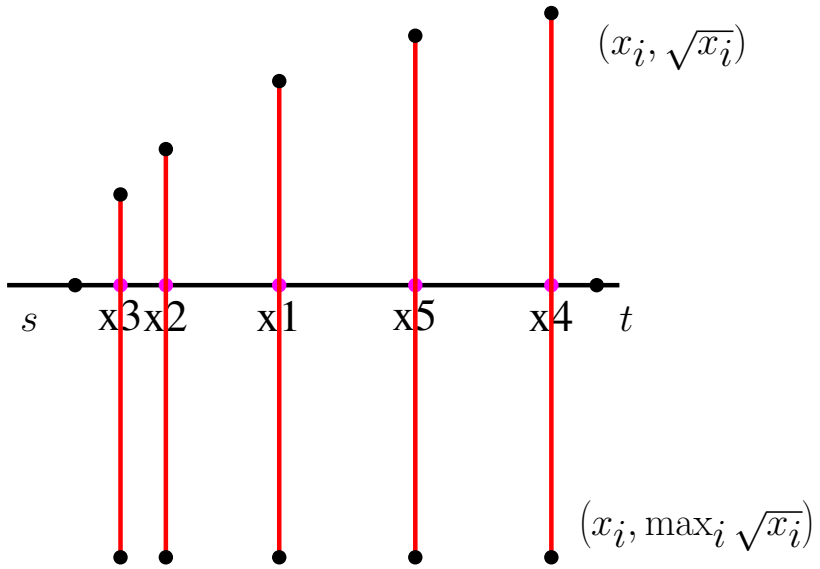
Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



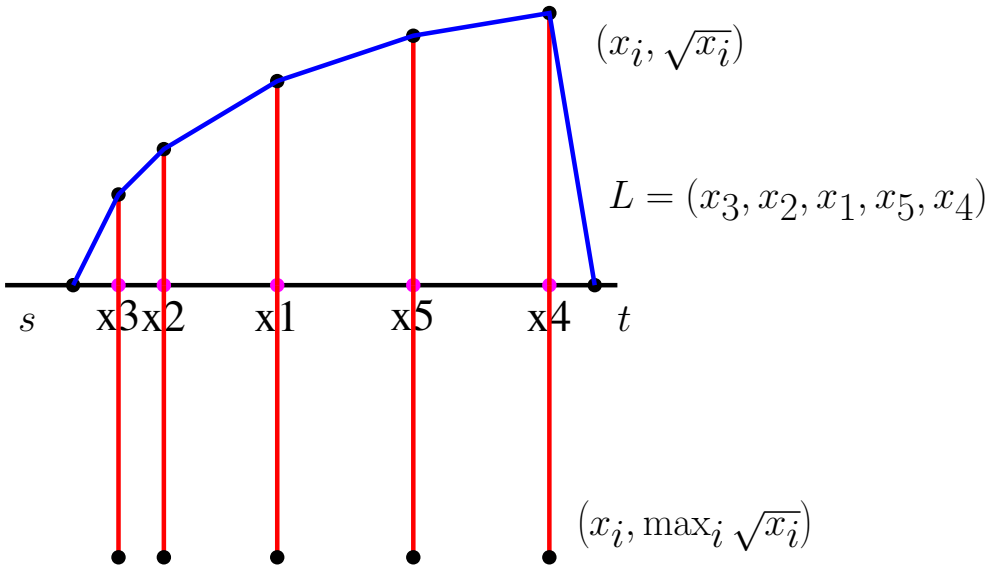
Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



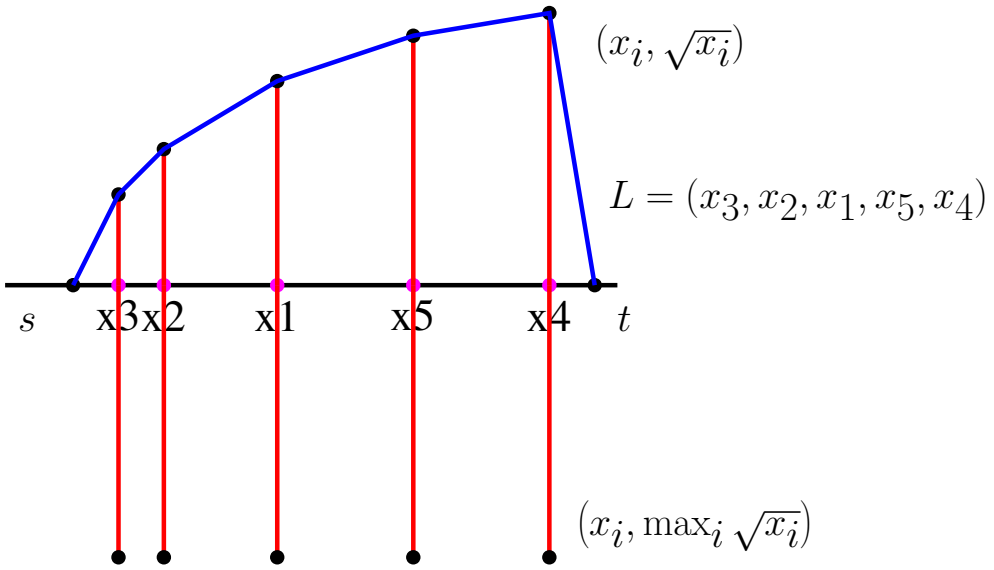
Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



Untere Schranke!

- Sortieren reduzieren auf kürzesten Weg berechnen!
- $x_1, x_2, \ldots, x_n$ in $O(n)$ auf polygonale Szene
- $\Omega(n \log n)$



Andere Ansätze

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$
2. Shortest Path Map Verfahren

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$
2. Shortest Path Map Verfahren

   - Locus approach:

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$
2. Shortest Path Map Verfahren

   - Locus approach: Klassen gleicher Antwort!

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$
2. Shortest Path Map Verfahren

   - Locus approach: Klassen gleicher Antwort!
   - Fester Startpunkt s

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$
2. Shortest Path Map Verfahren

   - Locus approach: Klassen gleicher Antwort!
   - Fester Startpunkt s
   - Ebene einteilen: Zellen kombinatorisch gleich kürzeste Wege

Andere Ansätze

1. Sichtbarkeitsgraph, output-sensitiv: $O(n \log n + k)$
2. Shortest Path Map Verfahren

   - Locus approach: Klassen gleicher Antwort!
   - Fester Startpunkt s
   - Ebene einteilen: Zellen kombinatorisch gleich kürzeste Wege
   - Zelle ist Klasse mit gleicher Antwort

Shortest Path Map Theorem 1.9

Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden

Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra,

Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise

Shortest Path Map Theorem 1.9

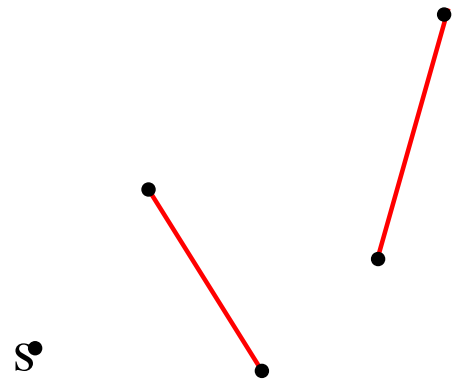
- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$,

Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$

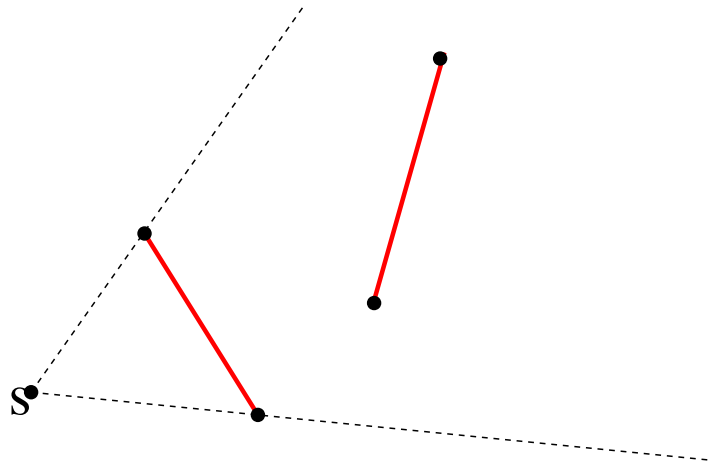
Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



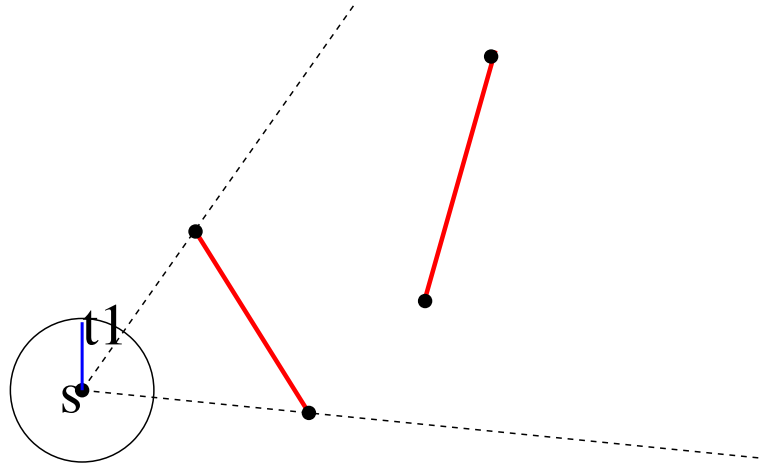
Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



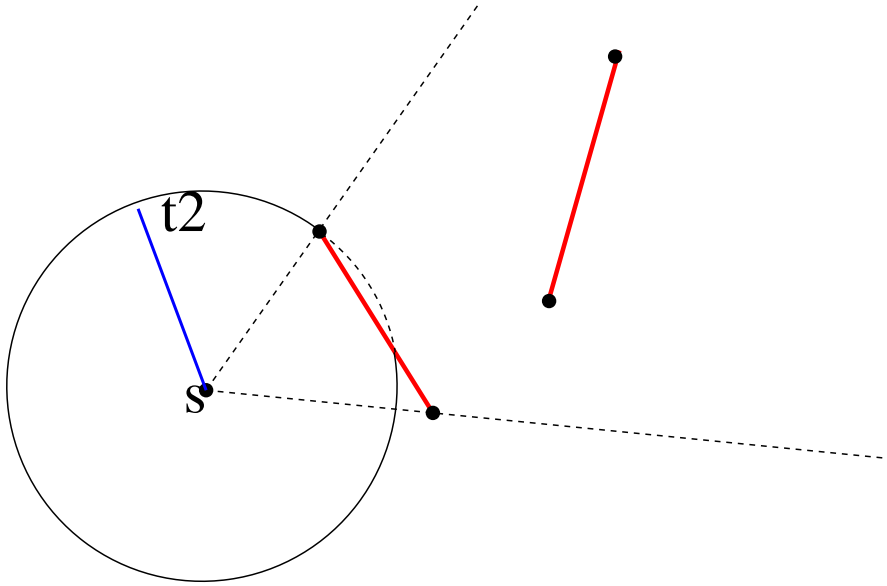
Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



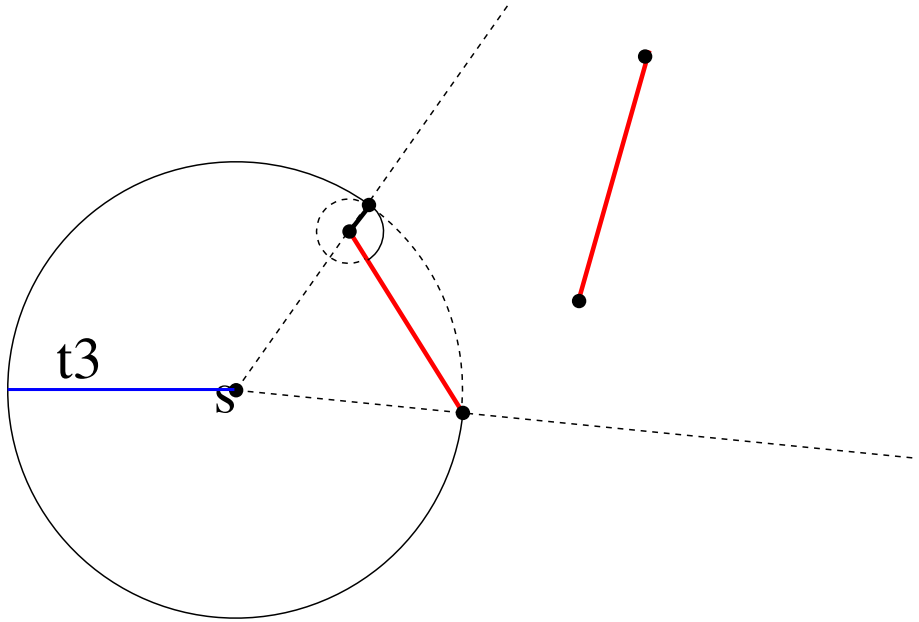
Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



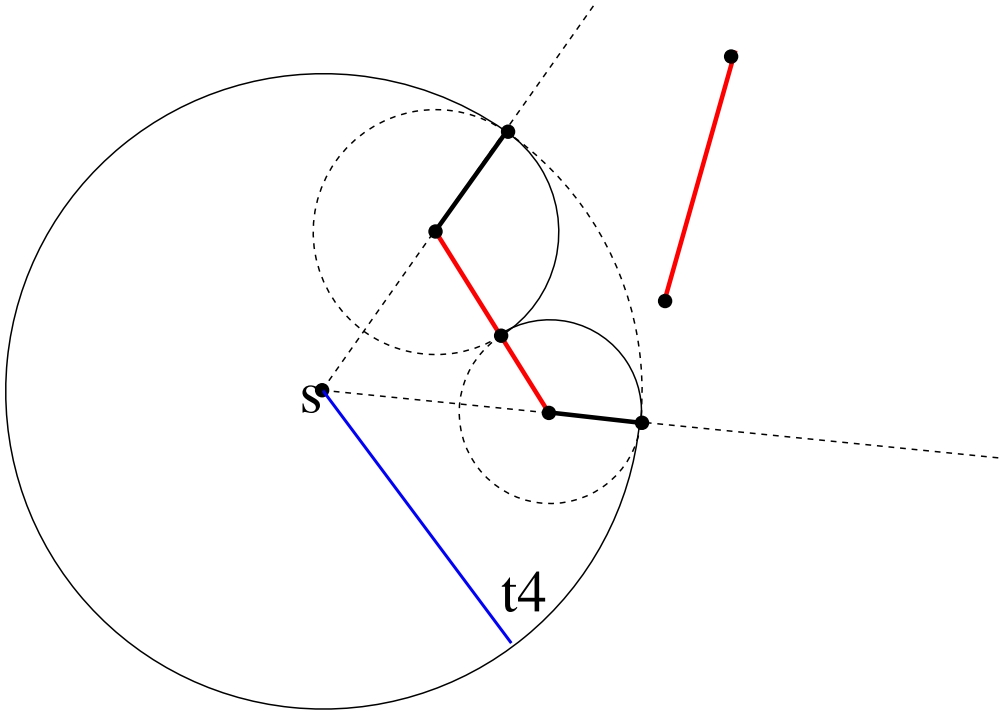
Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



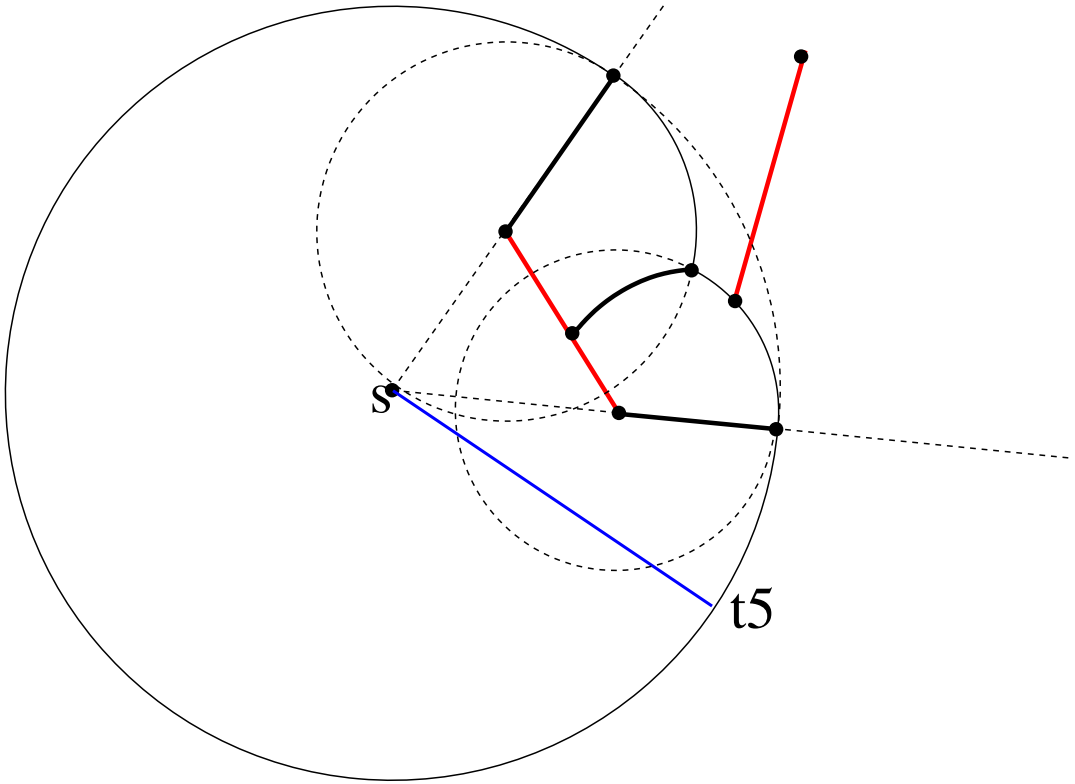
Shortest Path Map Theorem 1.9

- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



Shortest Path Map Theorem 1.9

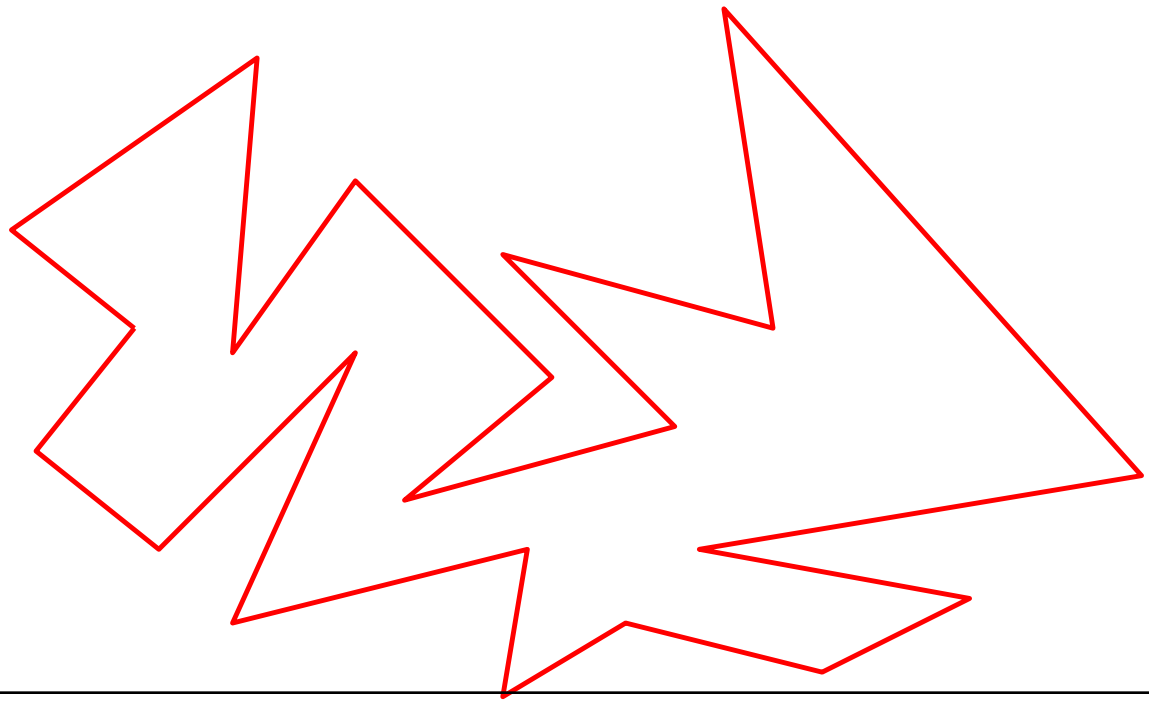
- Bisektoren: Hyperbelstücke, Geraden
- Berechnung: Continous Dijkstra, Simultan wachsende Kreise
- Laufzeit: Ber. $O(n \log n)$, Query $O(\log n)/O(\log n + k)$



Shortest Path in einfachen Polygonen

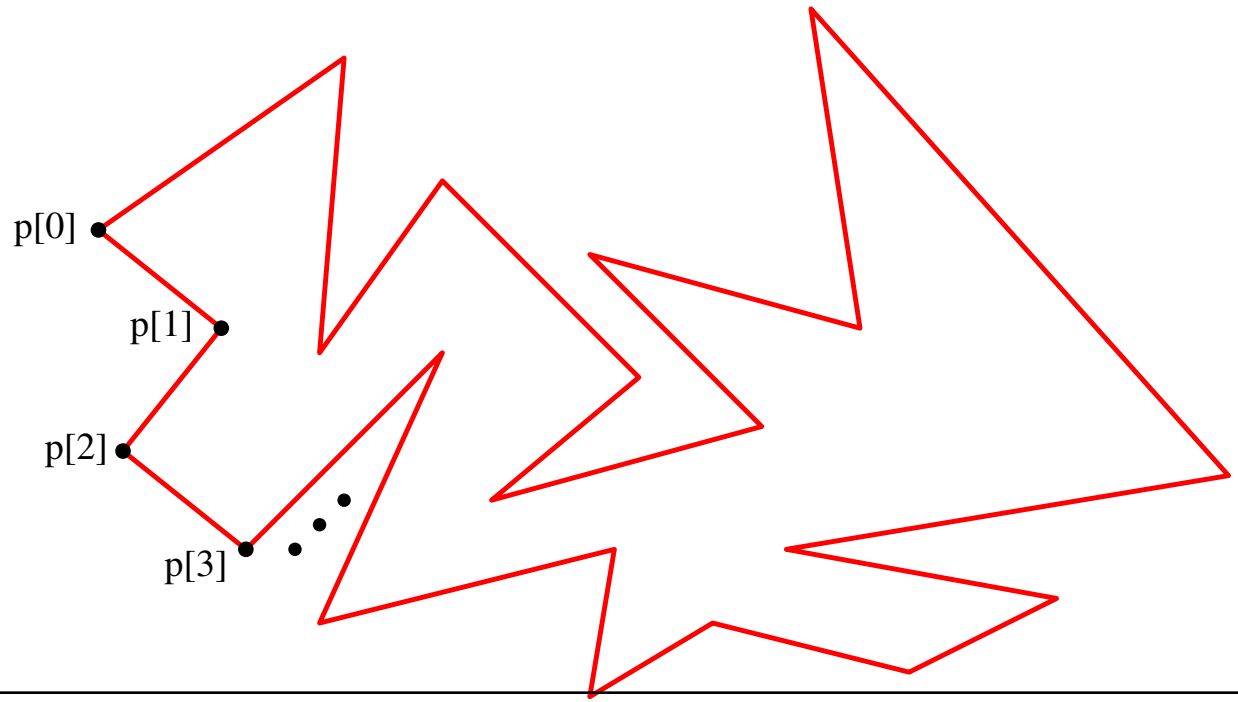
Shortest Path in einfachen Polygonen

- Einfaches Polygon P,



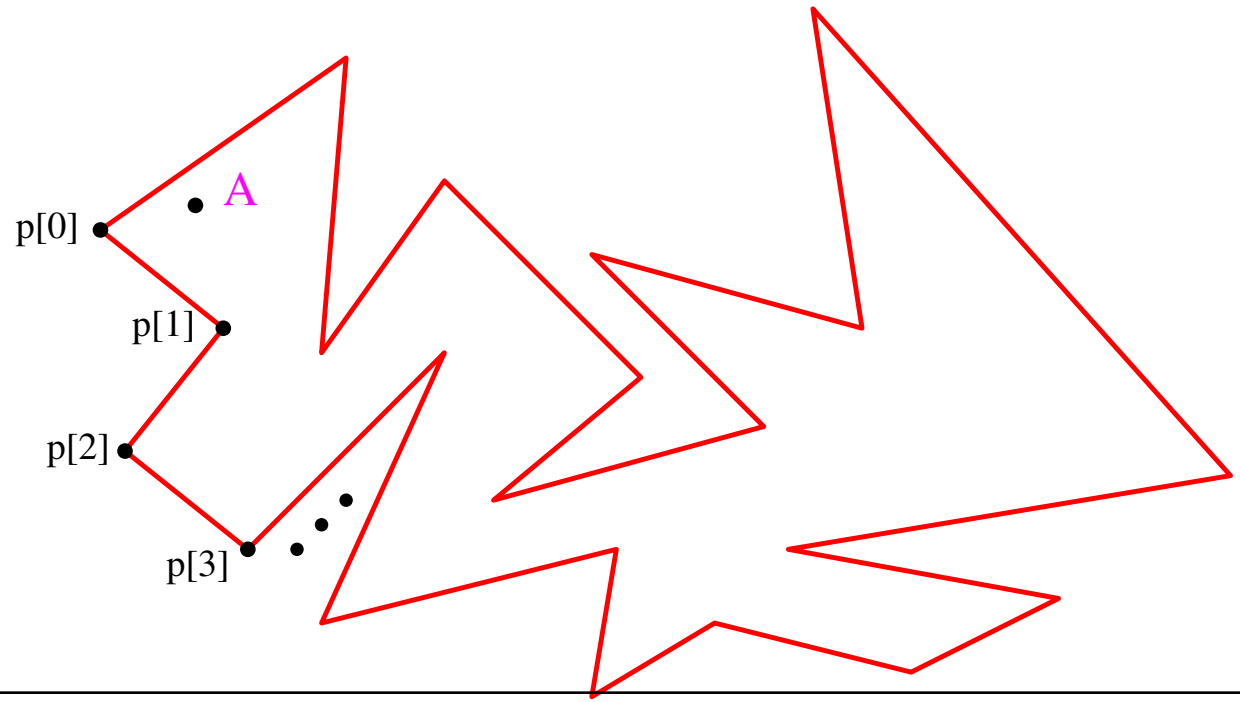
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order



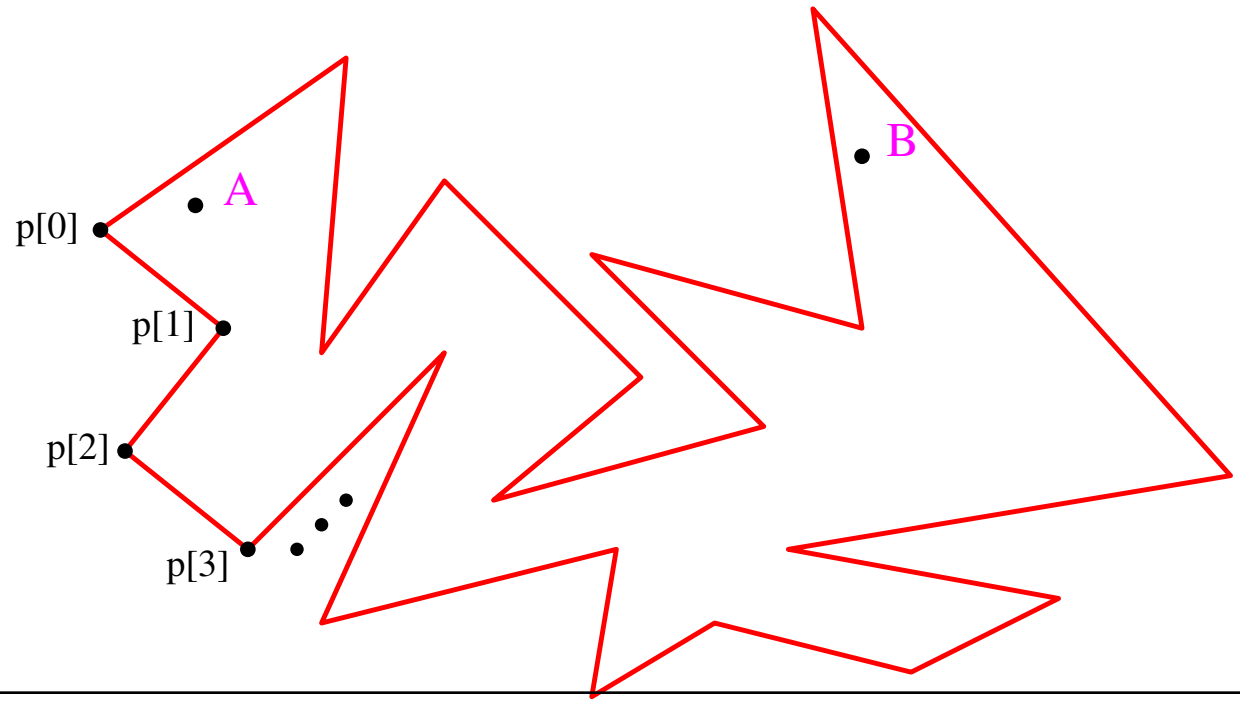
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order
- Start A



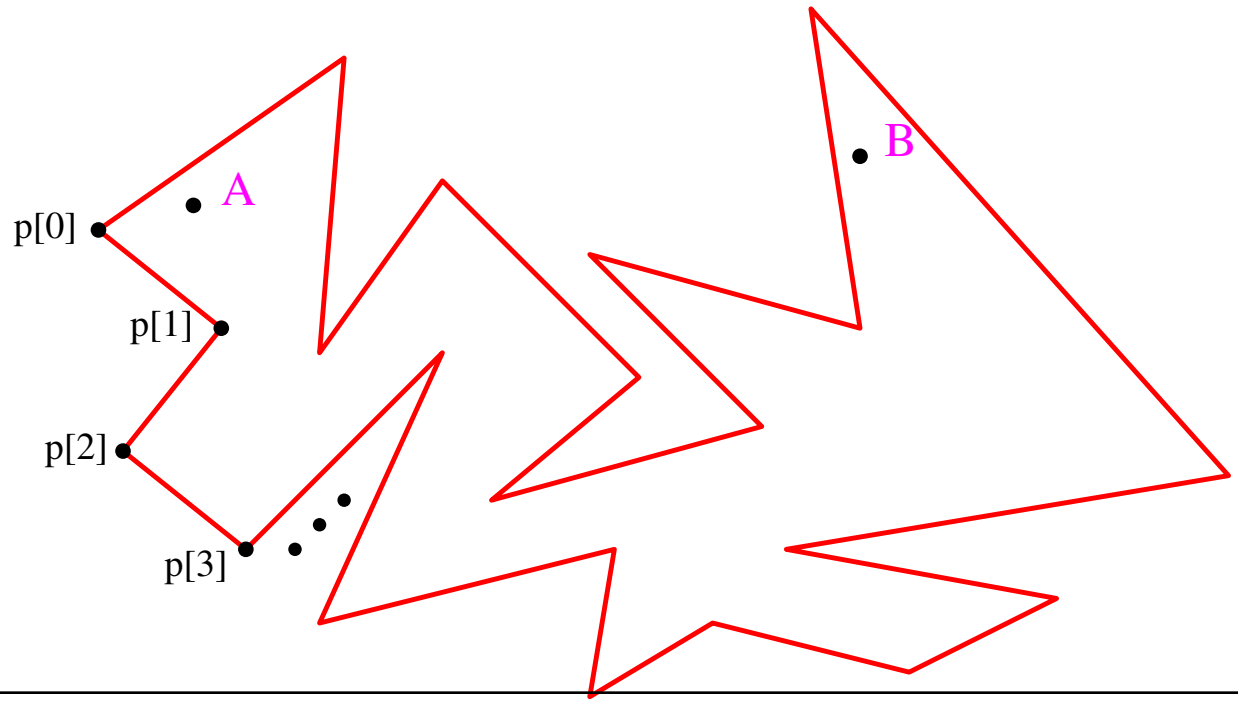
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order
- Start A und Ziel B



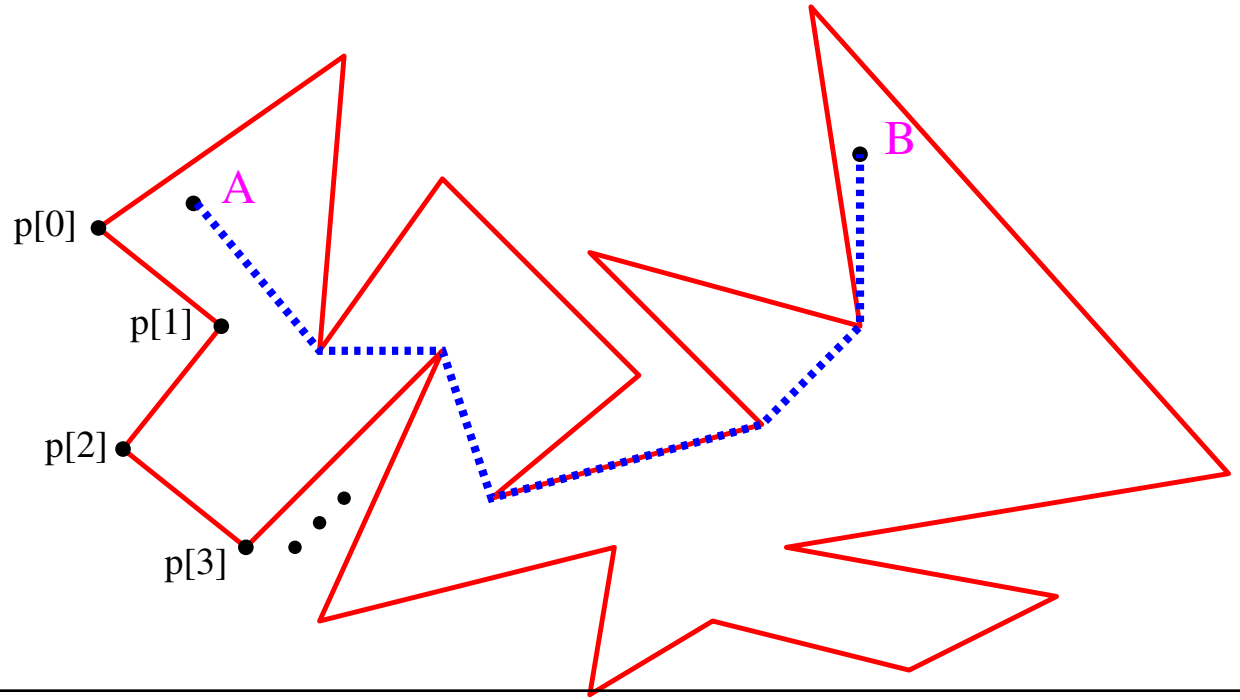
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order
- Start A und Ziel B
- Berechne kürzesten Weg von A nach B innerhalb P



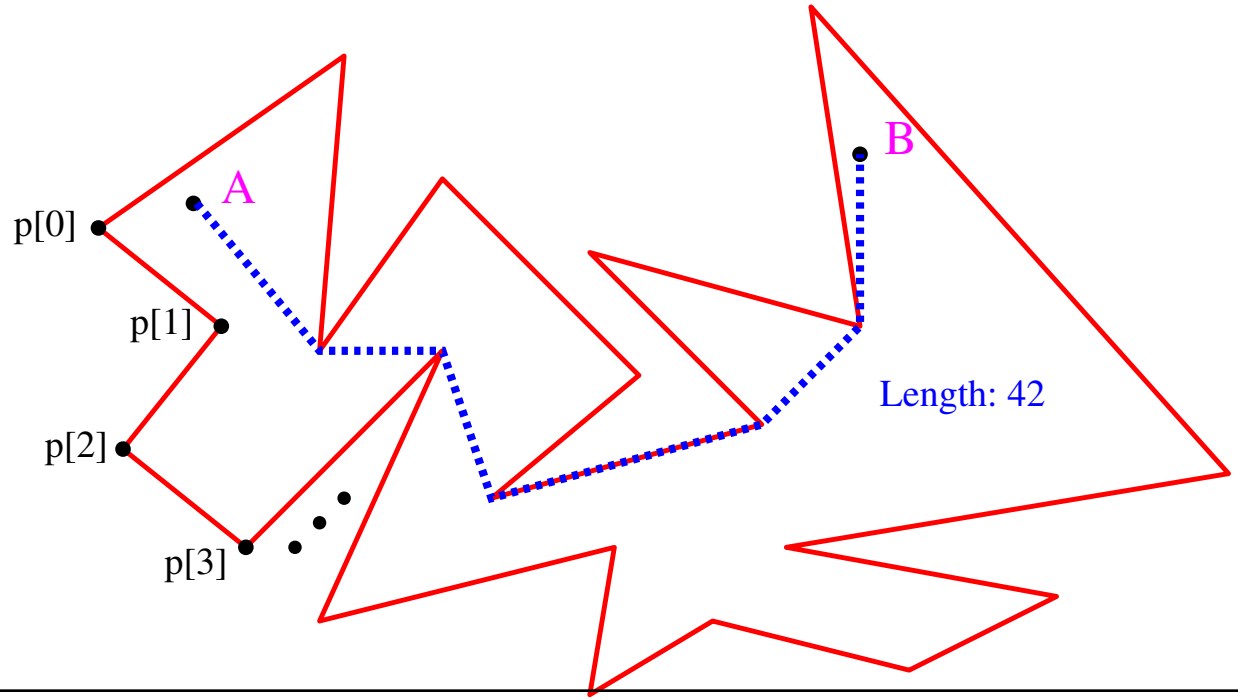
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order
- Start A und Ziel B
- Berechne kürzesten Weg von A nach B innerhalb P
- Polygonale Kette mit Knoten aus P



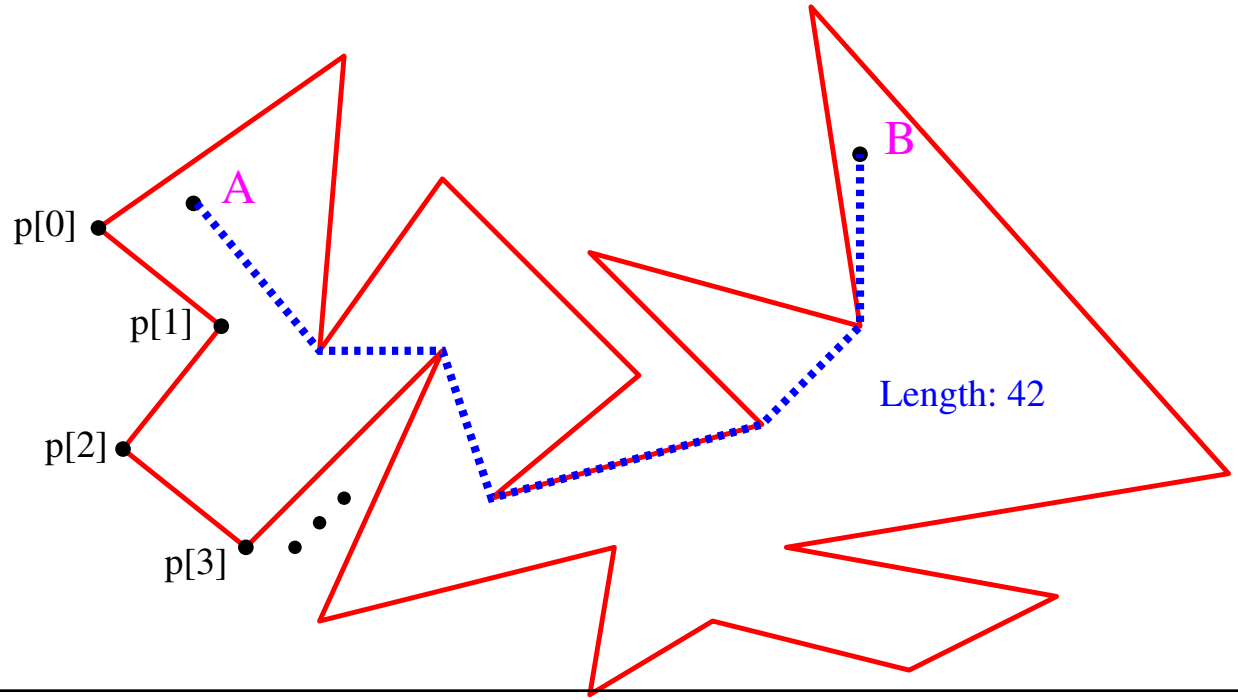
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order
- Start A und Ziel B
- Berechne kürzesten Weg von A nach B innerhalb P
- Polygonale Kette mit Knoten aus P/Länge des Pfades



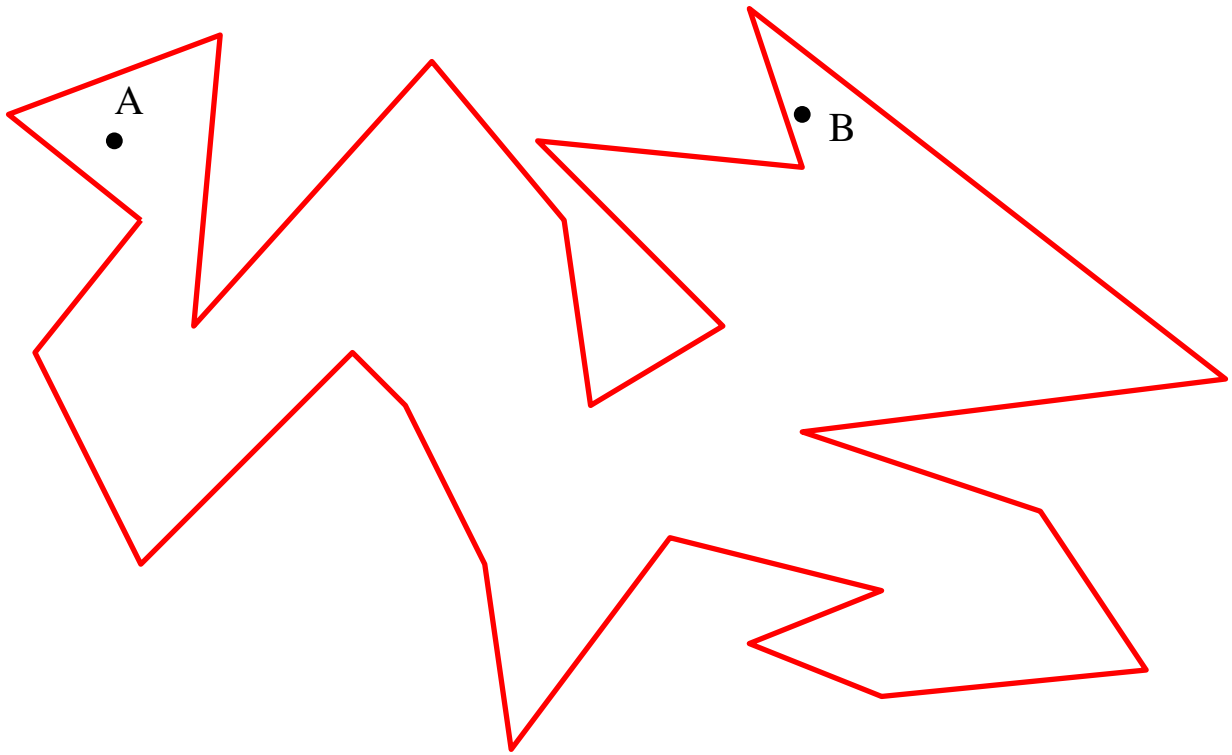
Shortest Path in einfachen Polygonen

- Einfaches Polygon P, DS: Seq. Knoten/Kanten, CCW-order
- Start A und Ziel B
- Berechne kürzesten Weg von A nach B innerhalb P
- Polygonale Kette mit Knoten aus P/Länge des Pfades
- Algorithmus: $\Omega(n)$, $|P| = n$



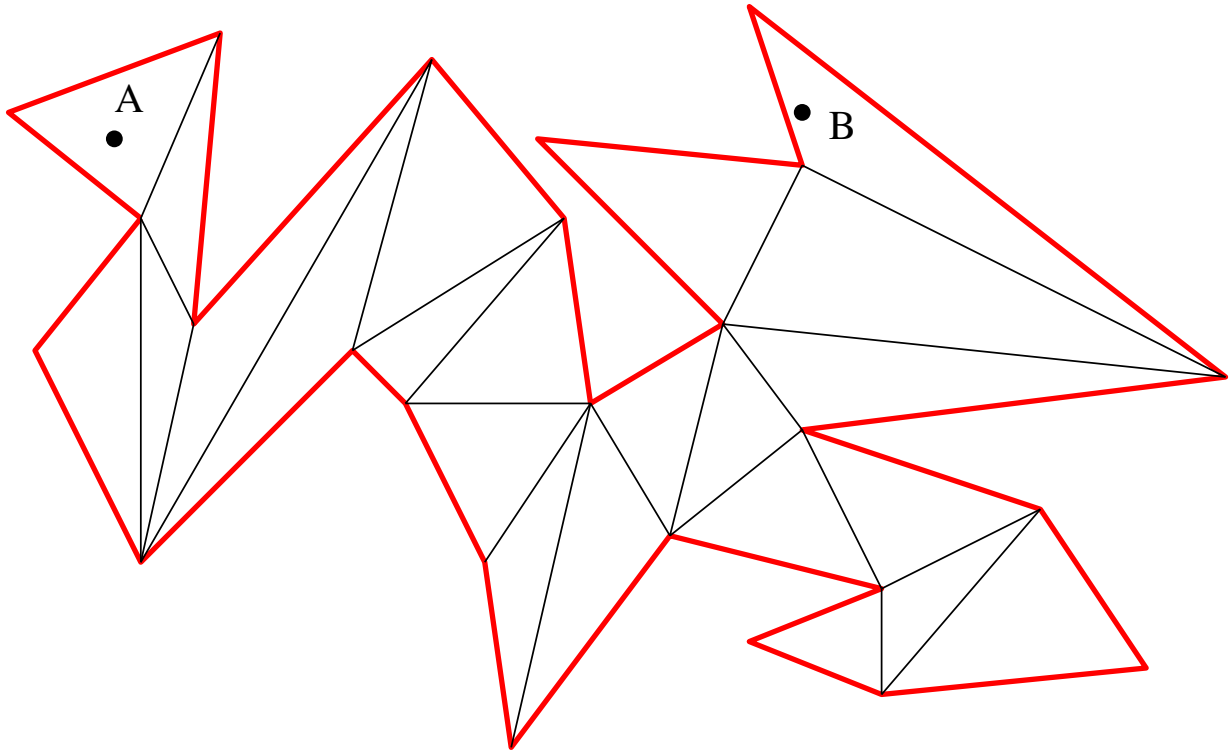
Einfache, effiziente Lösung (Lee/Preparata)

Einfache, effiziente Lösung (Lee/Preparata)



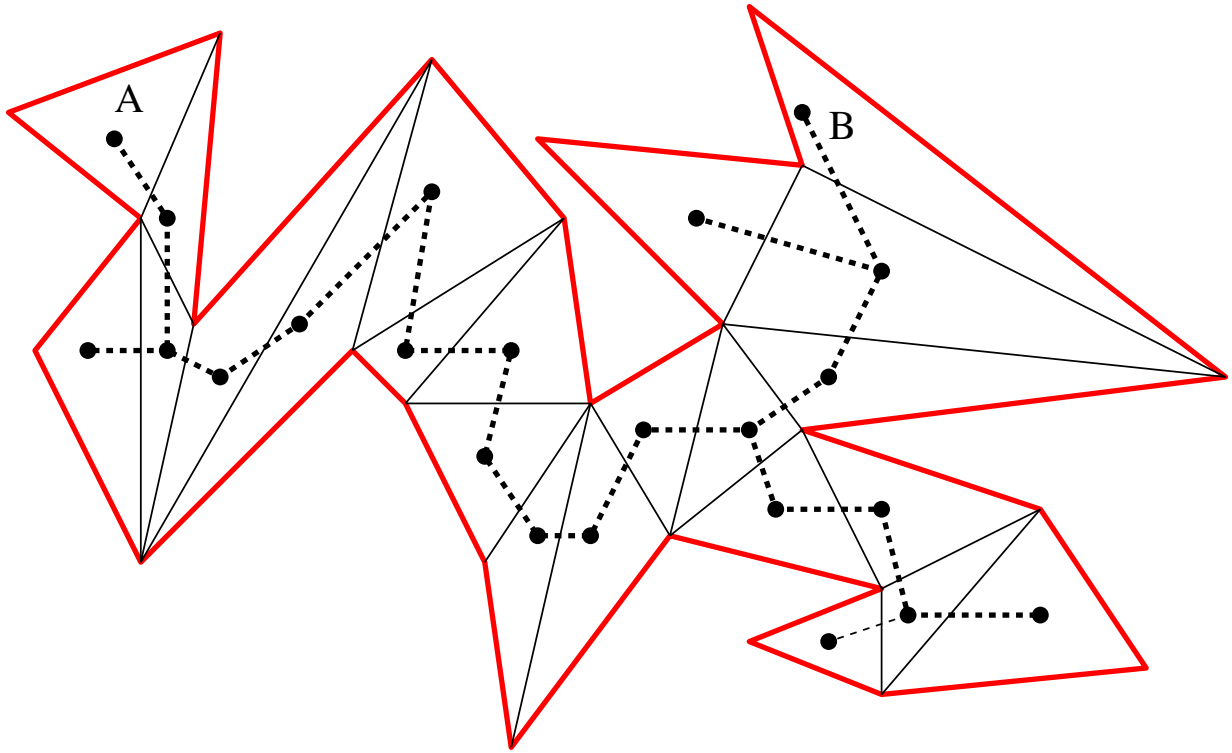
Einfache, effiziente Lösung (Lee/Preparata)

- Triangulation von P: Graph T, DS: DCEL, Adjacency list



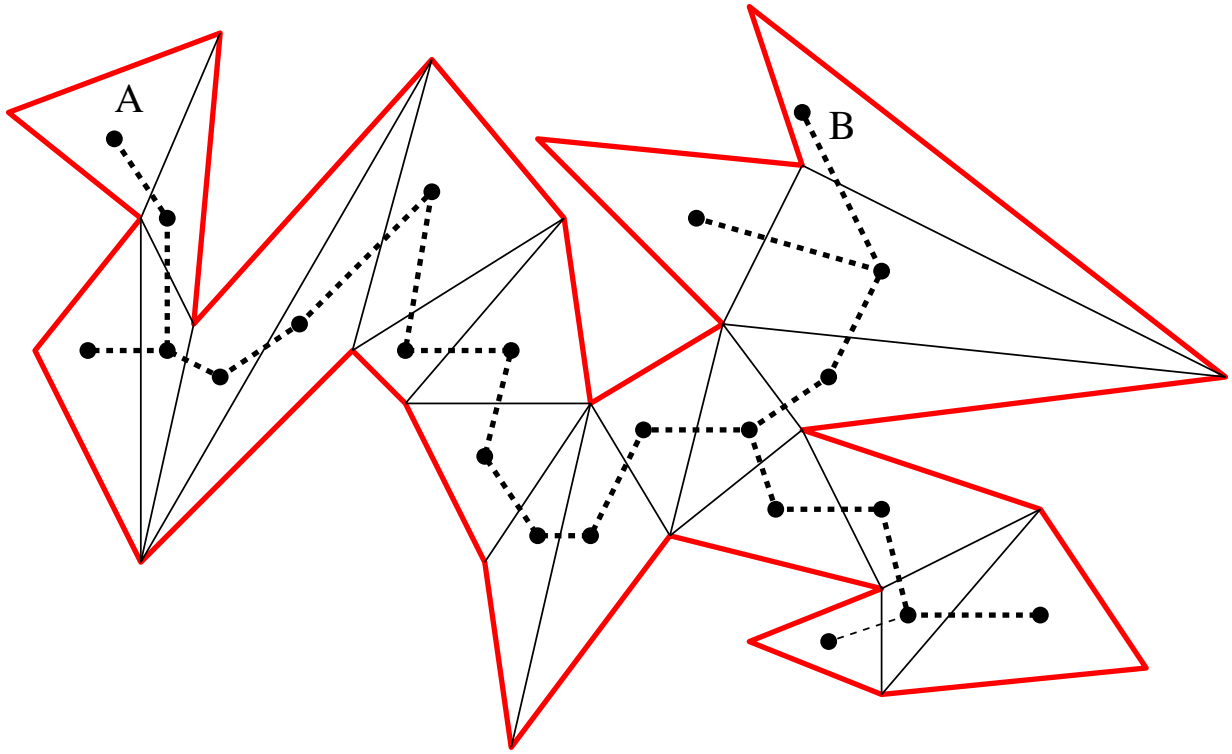
Einfache, effiziente Lösung (Lee/Preparata)

- Triangulation von P: Graph T, DS: DCEL, Adjacency list
- Dualer Graph: $D(T)$ (Tree),



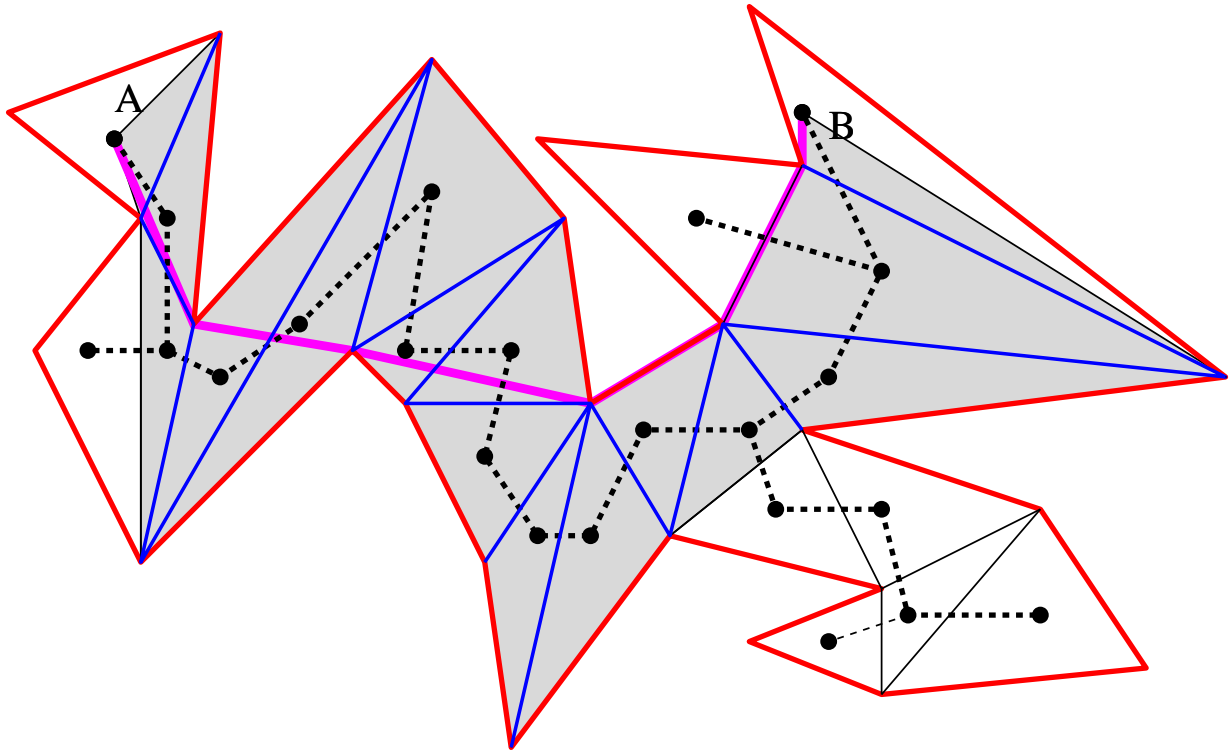
Einfache, effiziente Lösung (Lee/Preparata)

- Triangulation von P: Graph T, DS: DCEL, Adjacency list
- Dualer Graph: $D(T)$ (Tree), Depth First Search (DFS) on $D(T)$



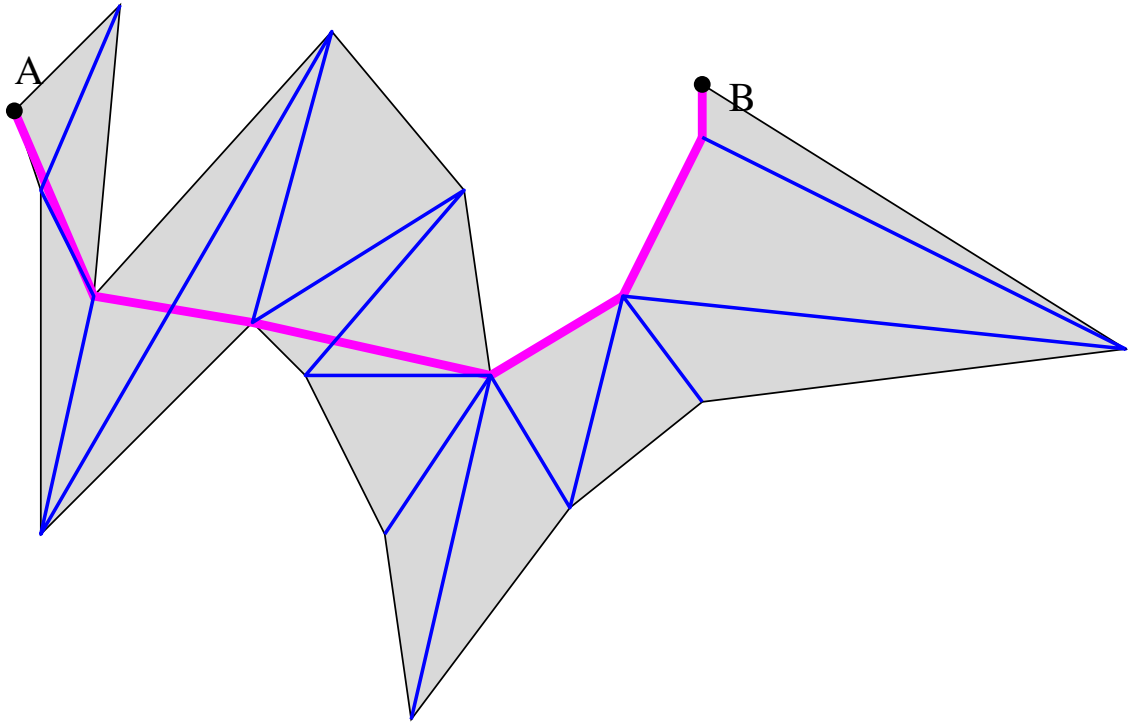
Einfache, effiziente Lösung (Lee/Preparata)

- Triangulation von P: Graph T, DS: DCEL, Adjacency list
- Dualer Graph: $D(T)$ (Tree), Depth First Search (DFS) on $D(T)$
- Subpolygon P', Kette von Dreiecken/Diagonalen



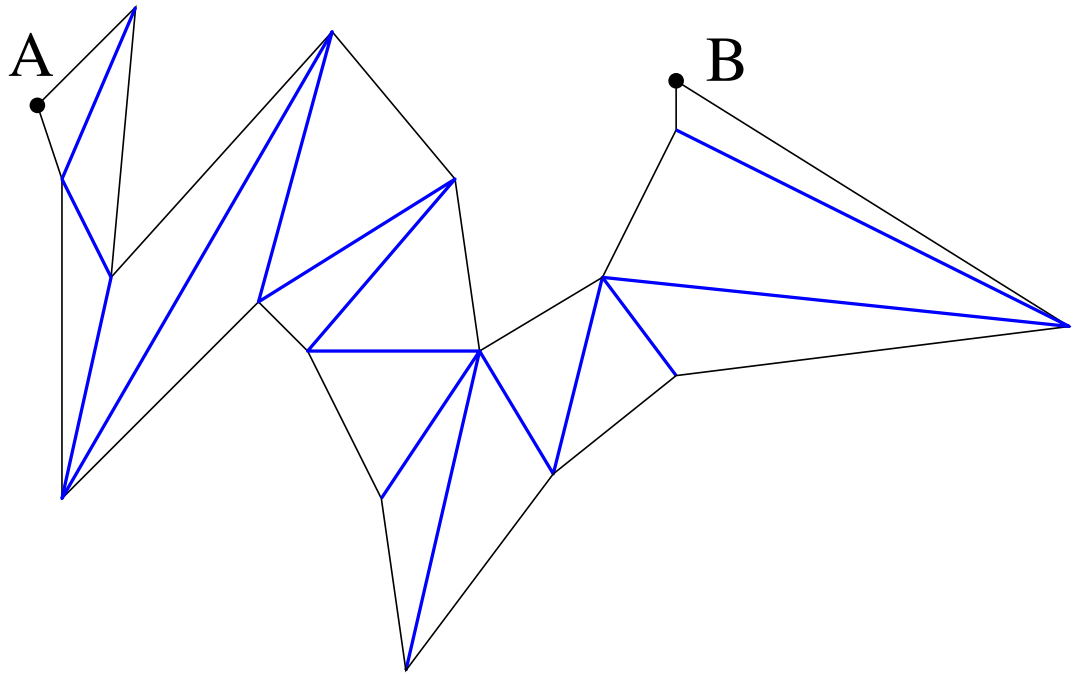
Einfache, effiziente Lösung (Lee/Preparata)

- Triangulation von P: Graph T, DS: DCEL, Adjacency list
- Dualer Graph: $D(T)$ (Tree), Depth First Search (DFS) on $D(T)$
- Subpolygon P', Kette von Dreiecken/Diagonalen



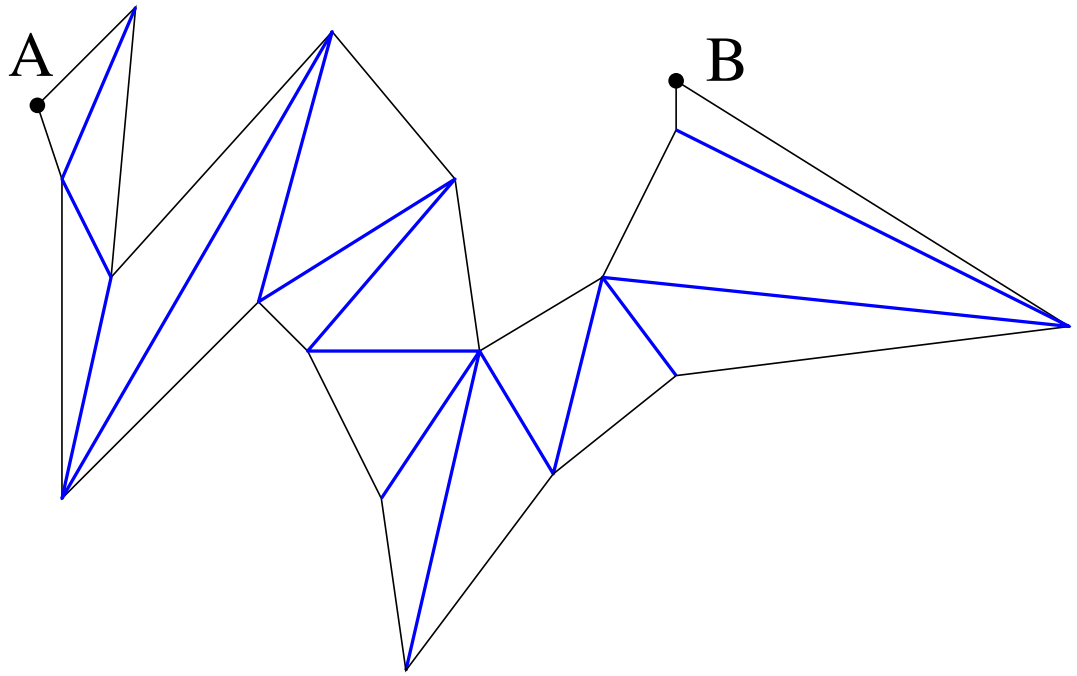
Shortest Path in P'

Shortest Path in P'



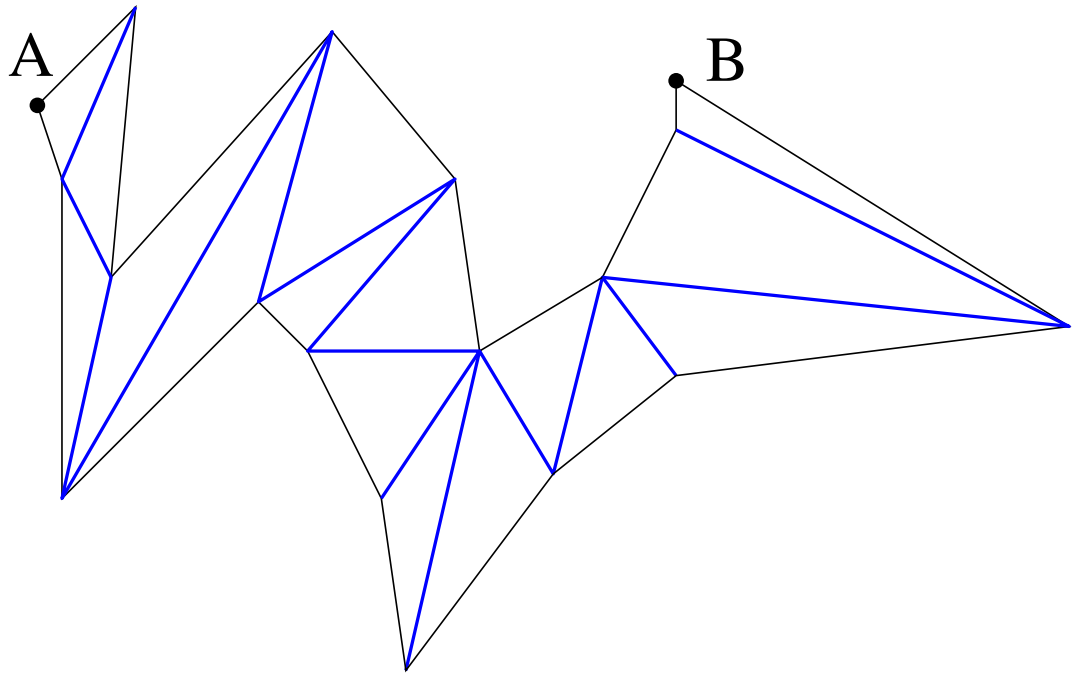
Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)



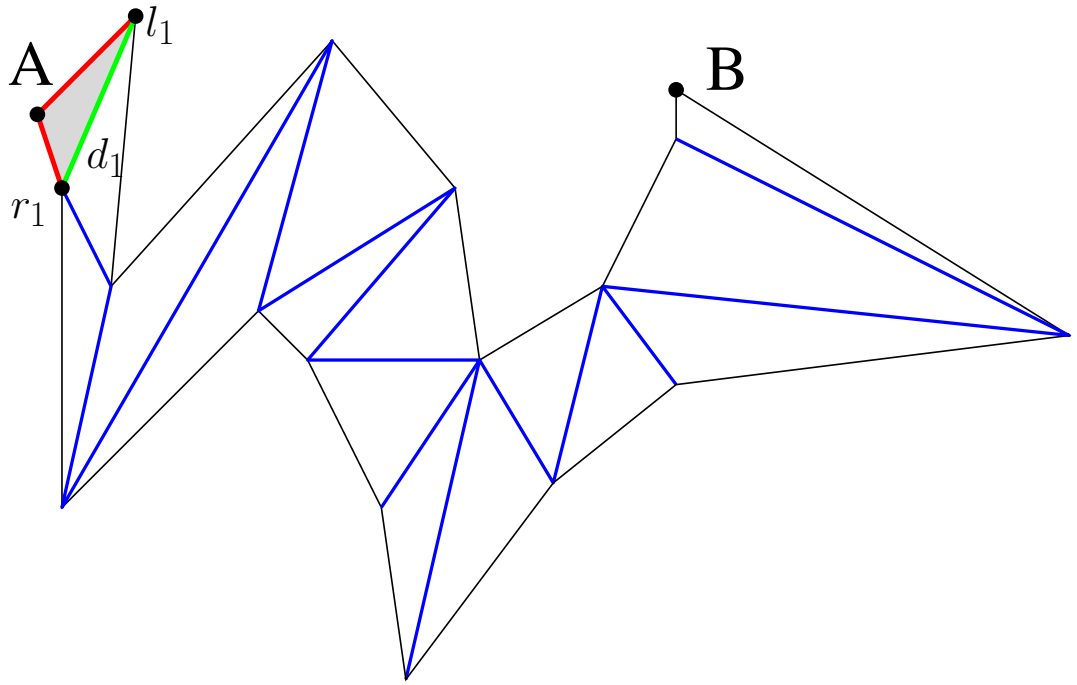
Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)
- Induktiv: Shortest paths zu den Endpunkten der Diagonalen



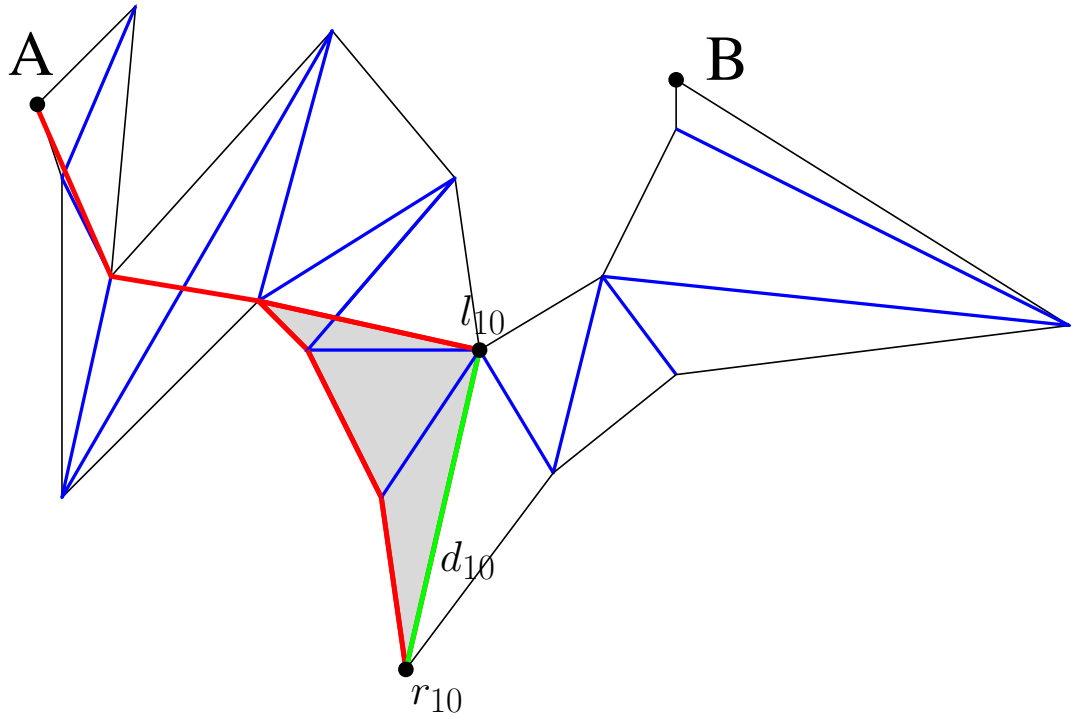
Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)
- Induktiv: Shortest paths zu den Endpunkten der Diagonalen
- Ind. Anf,



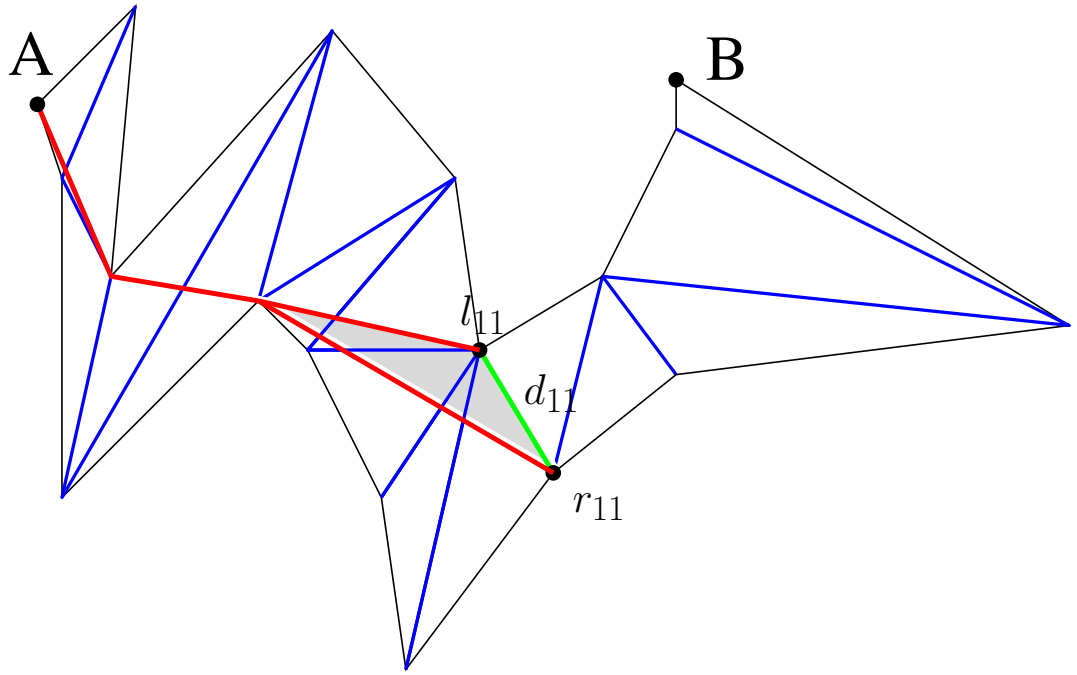
Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)
- Induktiv: Shortest paths zu den Endpunkten der Diagonalen
- Ind. Anf, Ind. Schritt,



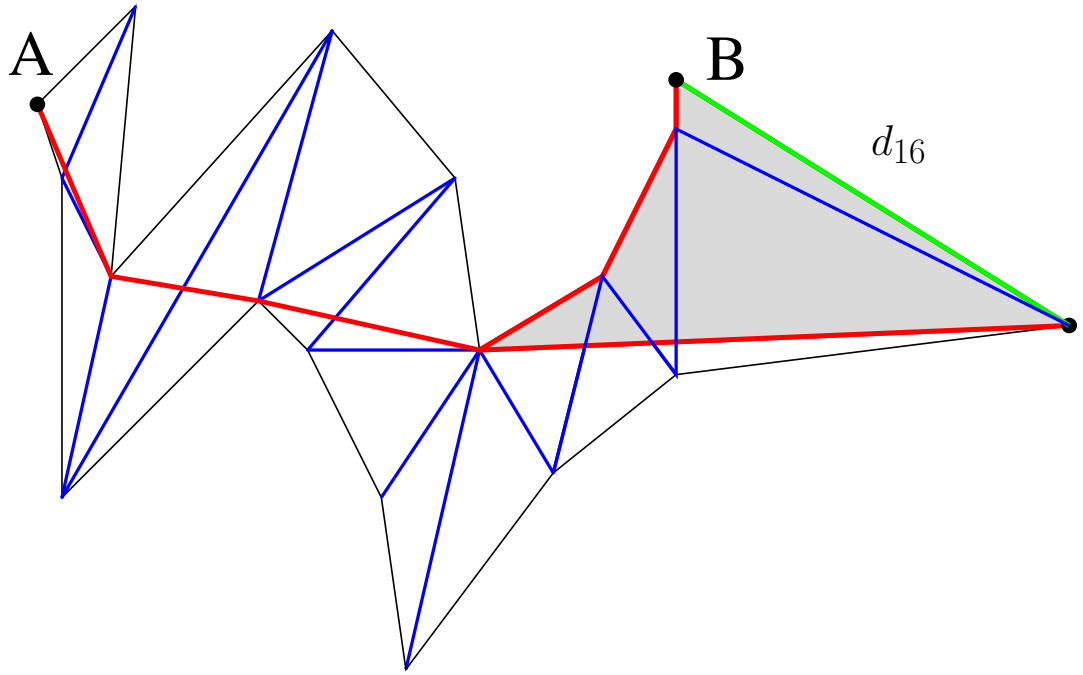
Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)
- Induktiv: Shortest paths zu den Endpunkten der Diagonalen
- Ind. Anf, Ind. Schritt,



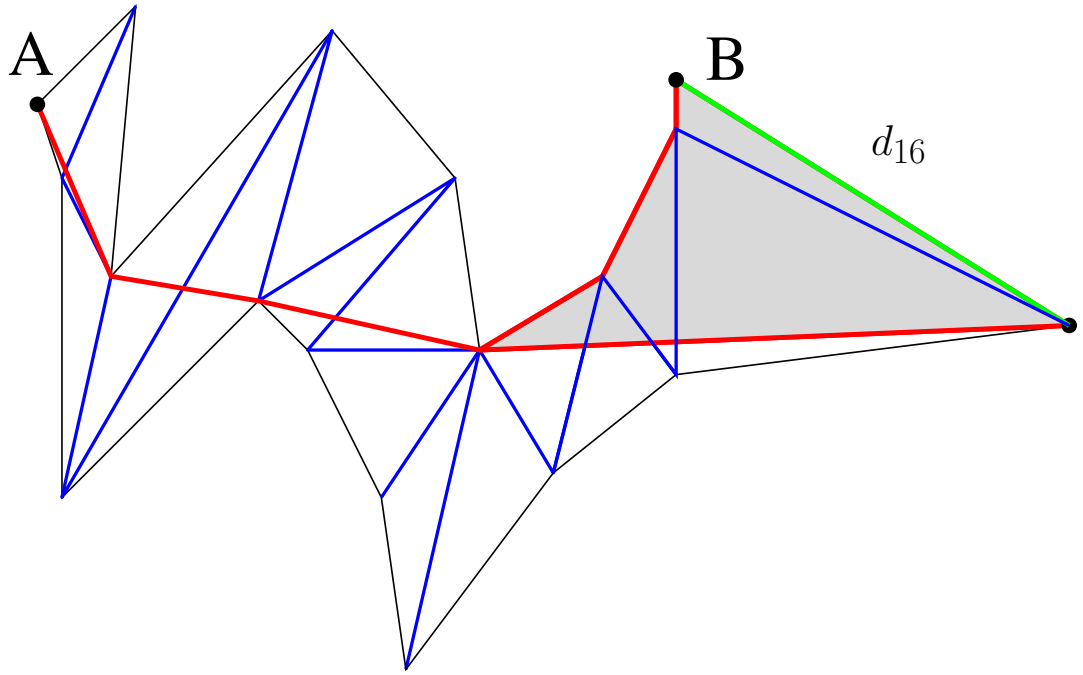
Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)
- Induktiv: Shortest paths zu den Endpunkten der Diagonalen
- Ind. Anf, Ind. Schritt, Letzte Diagonale$\Rightarrow$ Ergebnis,



Shortest Path in P'

- Kette von of Dreiecken/Diagonalen (Data structure)
- Induktiv: Shortest paths zu den Endpunkten der Diagonalen
- Ind. Anf, Ind. Schritt, Letzte Diagonale$\Rightarrow$ Ergebnis, Laufzeit?



Zusammenfassung: Alg. 1.4

Zusammenfassung: Alg. 1.4

- Triangulation von P:

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation:

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen:

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P':

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P': $O(n)$ Zeit und Platz

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P': $O(n)$ Zeit und Platz
- Berechne Shortest Path zu End-Diagonale, induktiv:

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P': $O(n)$ Zeit und Platz
- Berechne Shortest Path zu End-Diagonale, induktiv: $O(n)$??

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P': $O(n)$ Zeit und Platz
- Berechne Shortest Path zu End-Diagonale, induktiv: $O(n)$??
- Shortest Path von A nach B innerhalb P in $O(n)$ Zeit und Platz!

Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P': $O(n)$ Zeit und Platz
- Berechne Shortest Path zu End-Diagonale, induktiv: $O(n)$??
- Shortest Path von A nach B innerhalb P in $O(n)$ Zeit und Platz! optimal!!

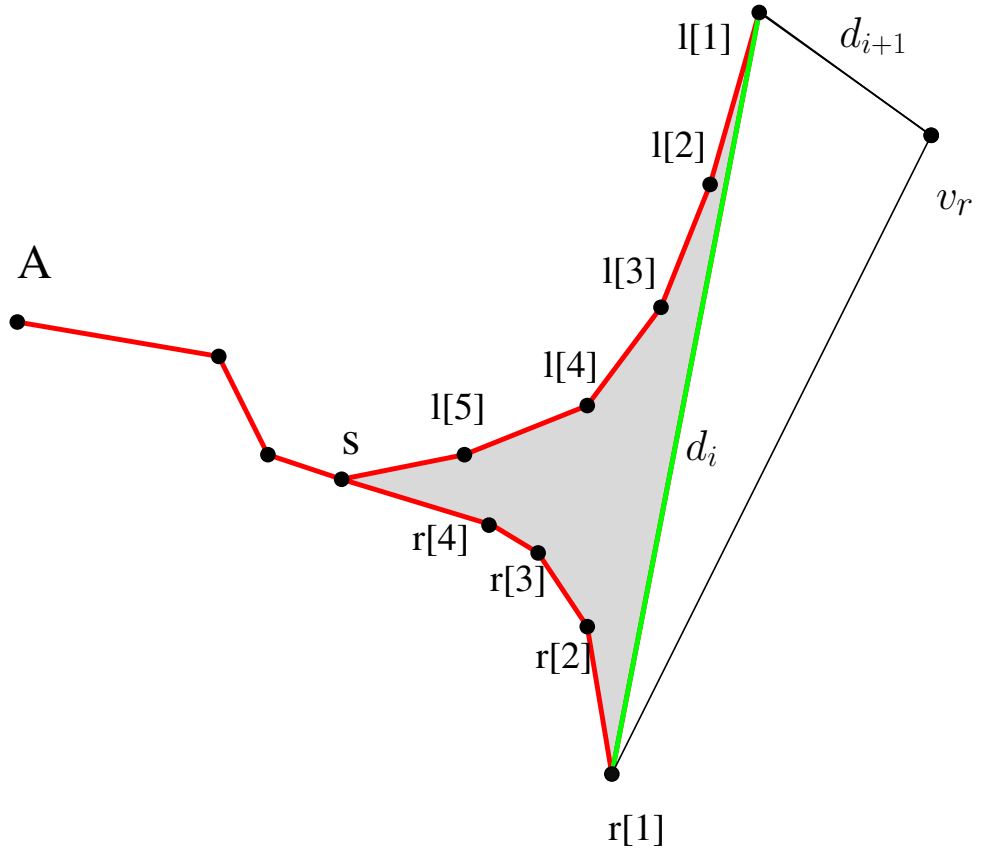
Zusammenfassung: Alg. 1.4

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS auf Dualen Graphen: $O(n)$ Zeit
- Berechne P': $O(n)$ Zeit und Platz
- Berechne Shortest Path zu End-Diagonale, induktiv: $O(n)$??
- Shortest Path von A nach B innerhalb P in $O(n)$ Zeit und Platz! optimal!!
- Theorem 1.11

Verbleibt: Indukiver Schritt!

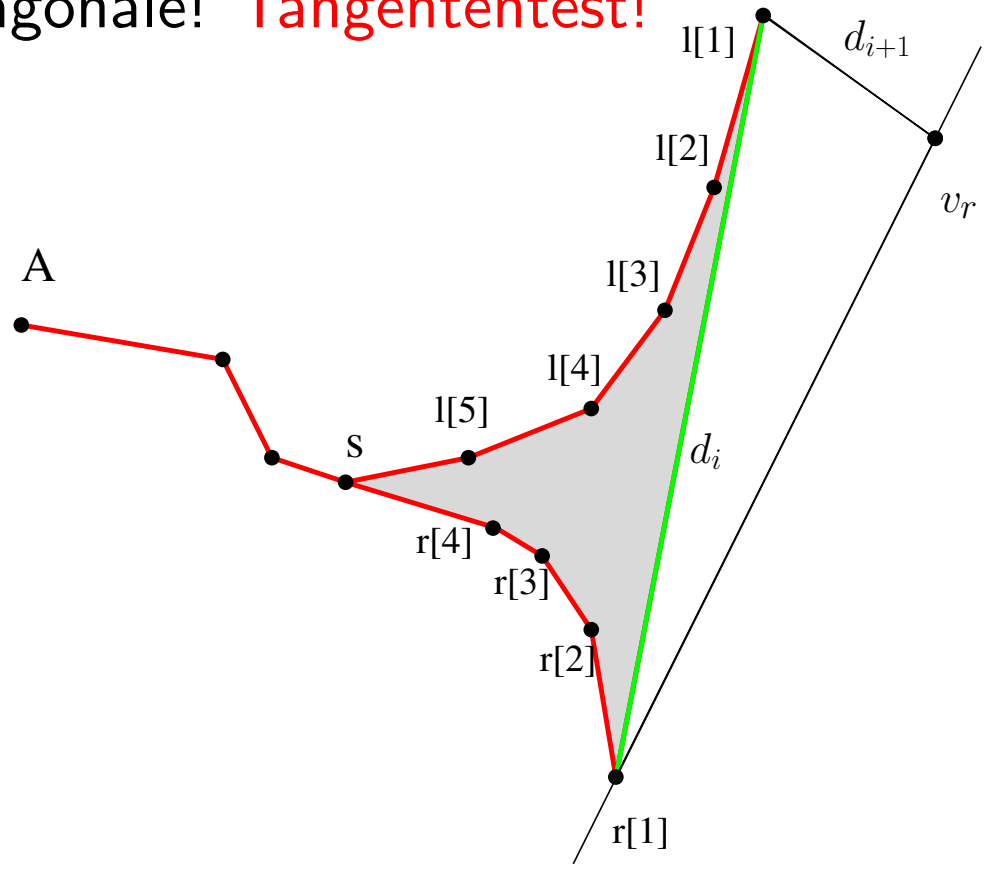
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!



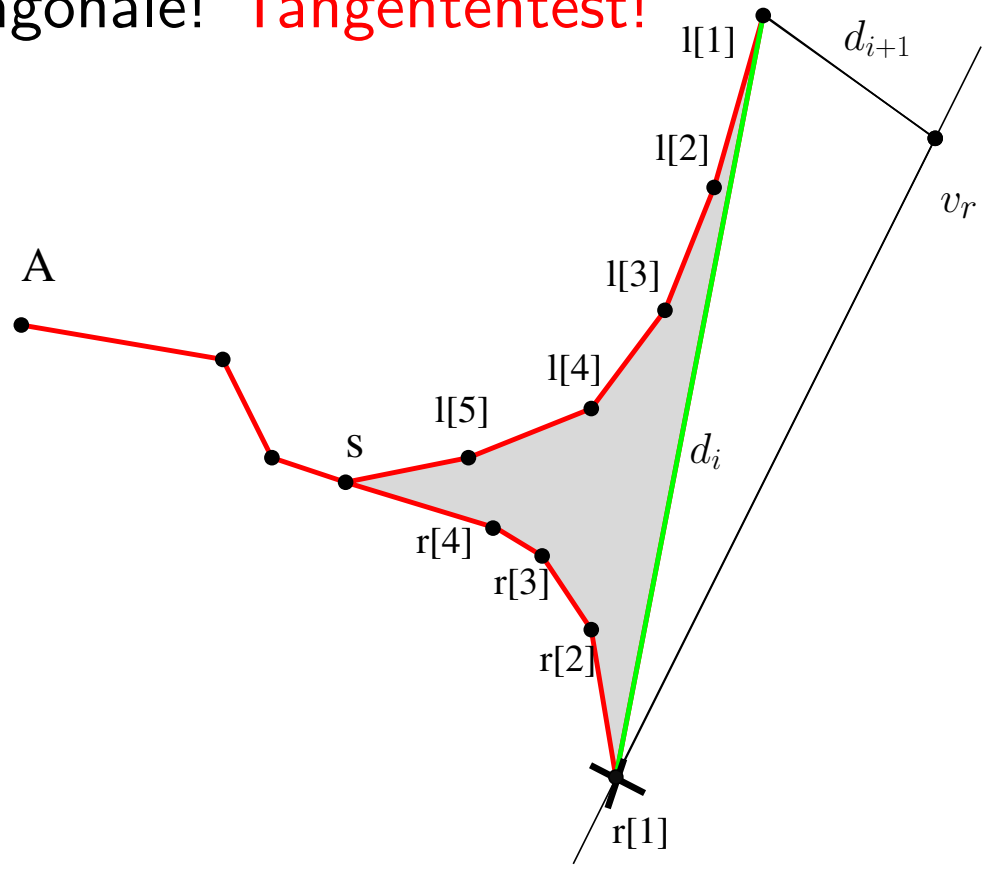
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



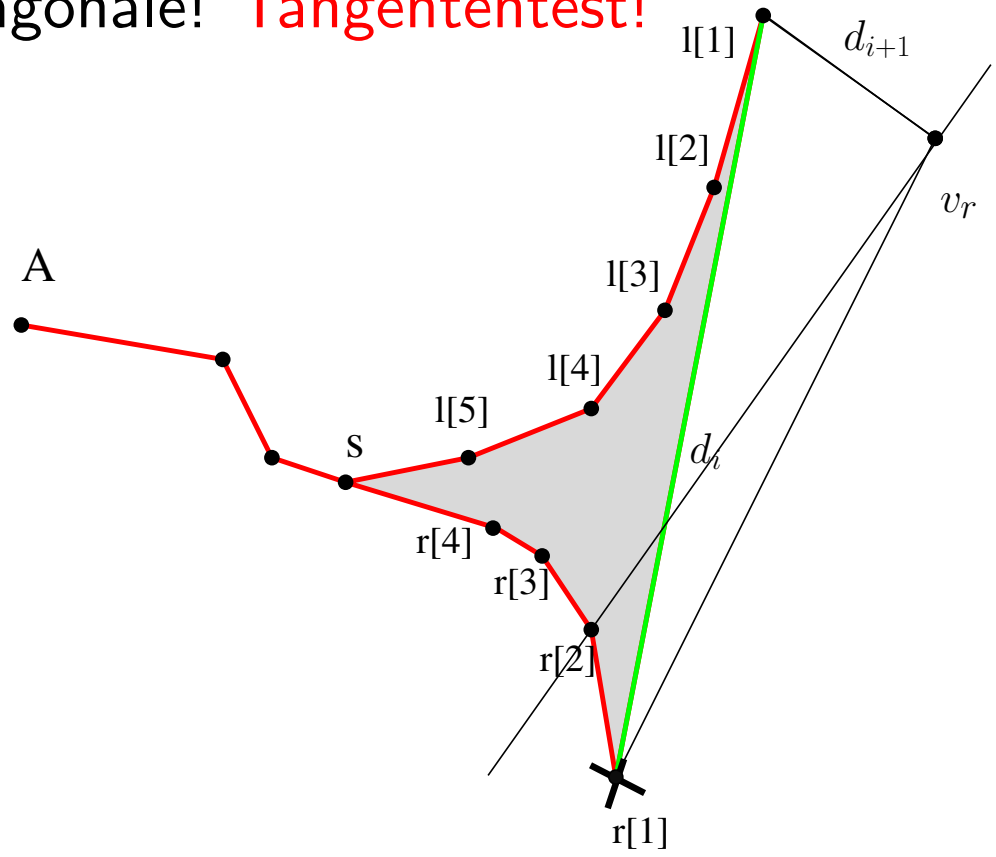
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



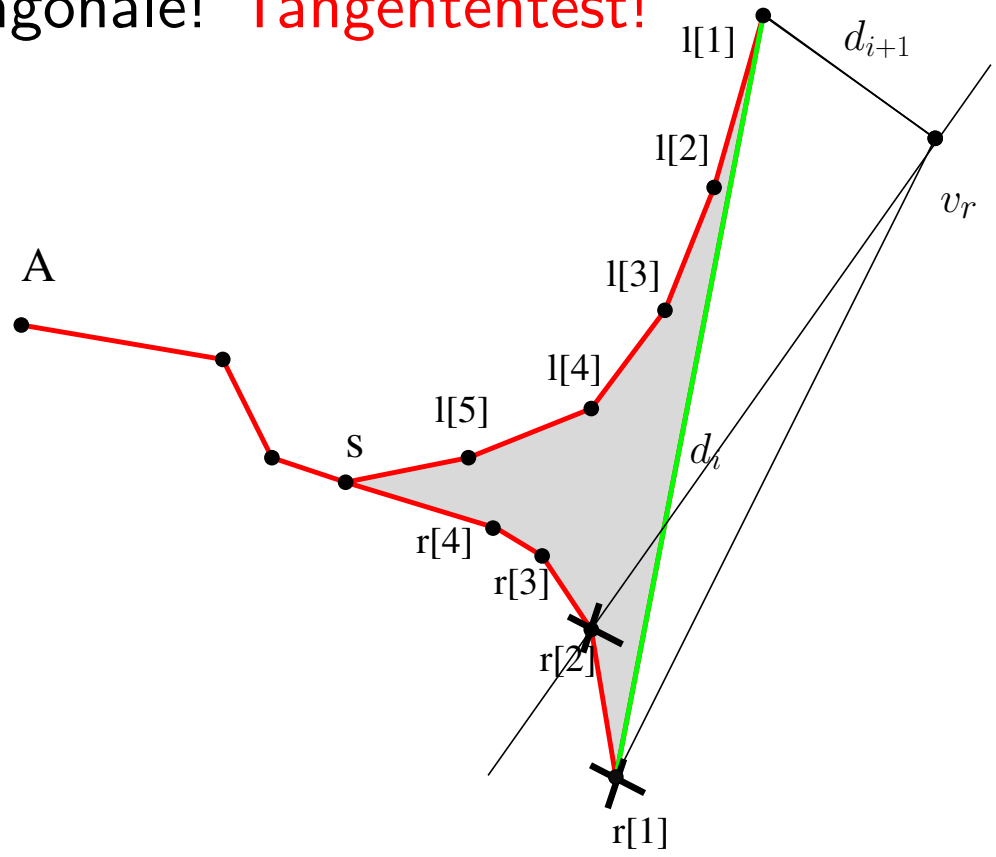
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



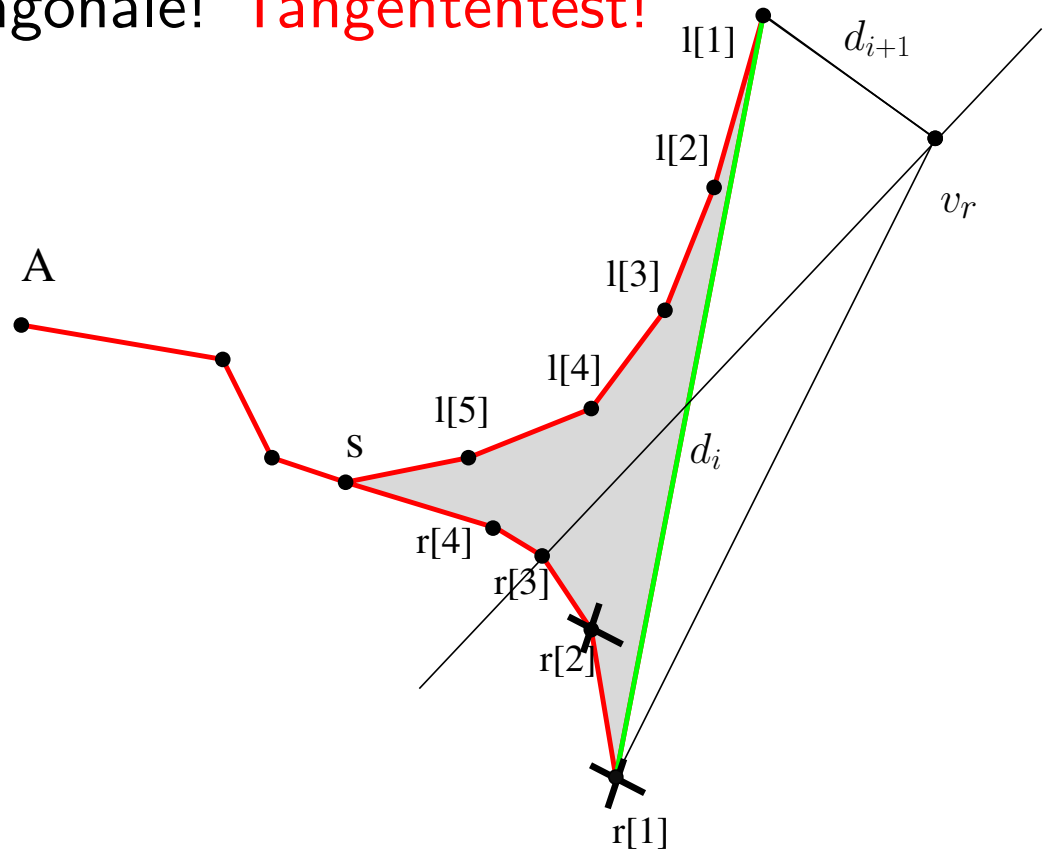
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



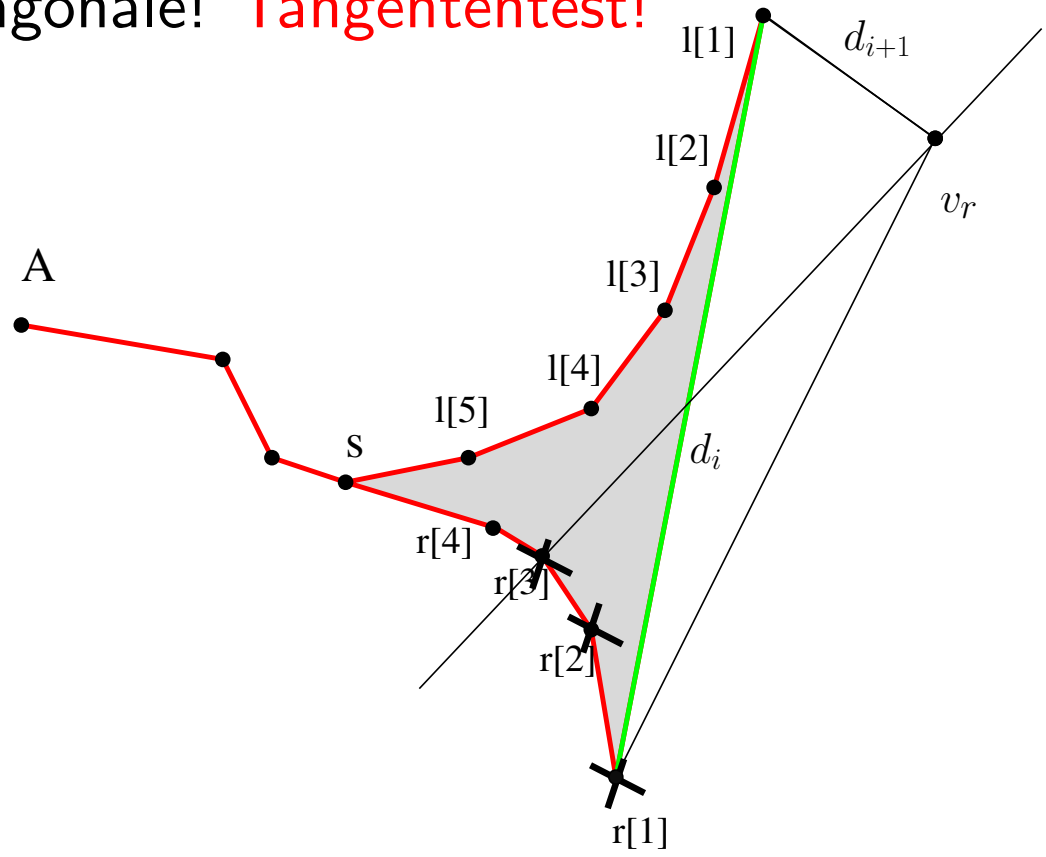
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



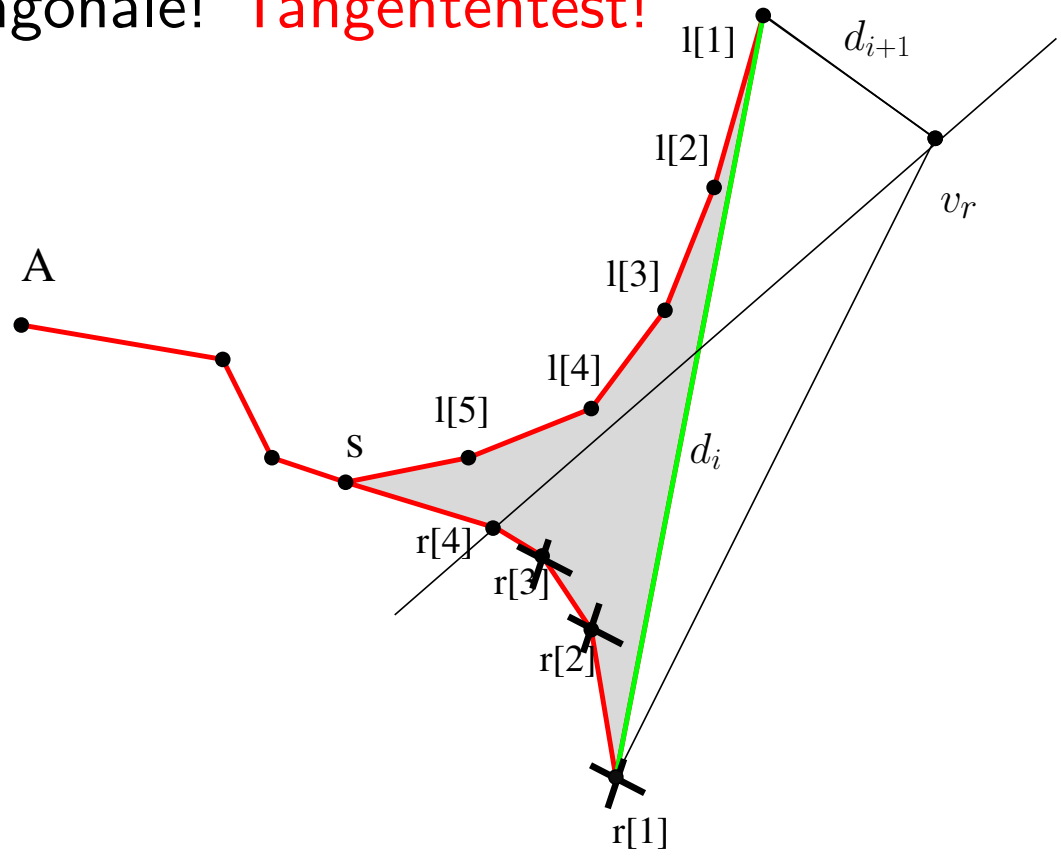
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



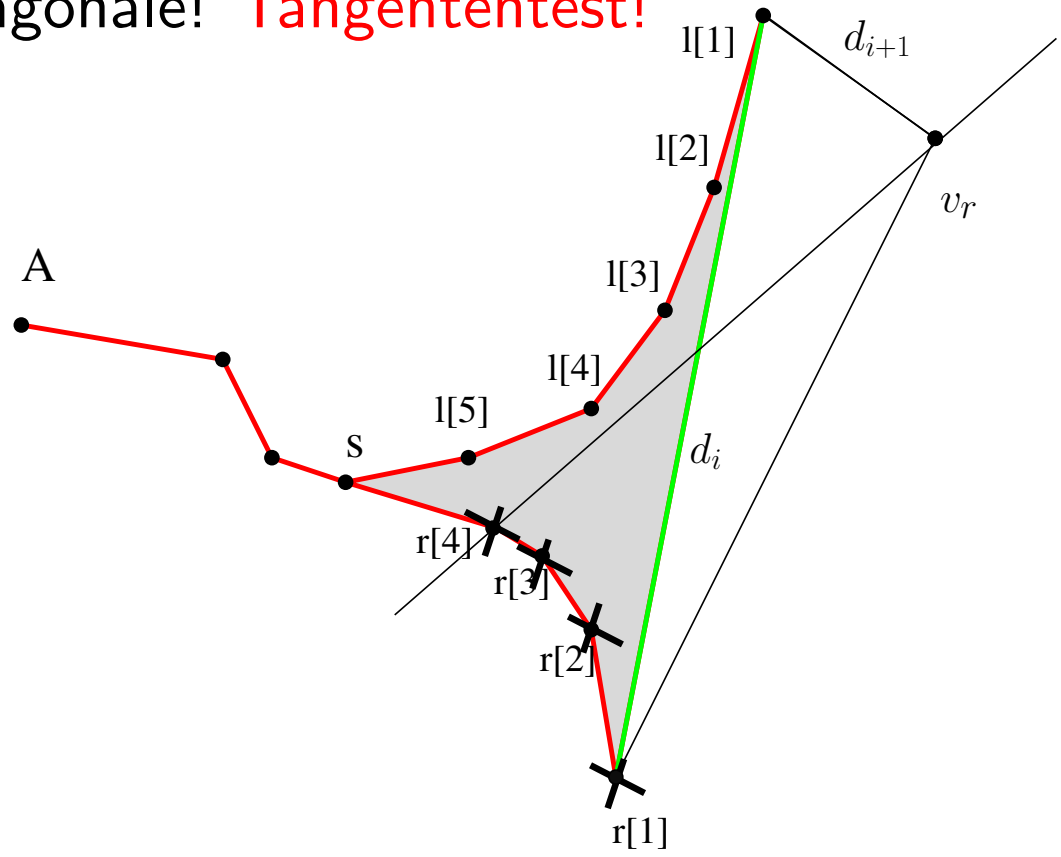
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



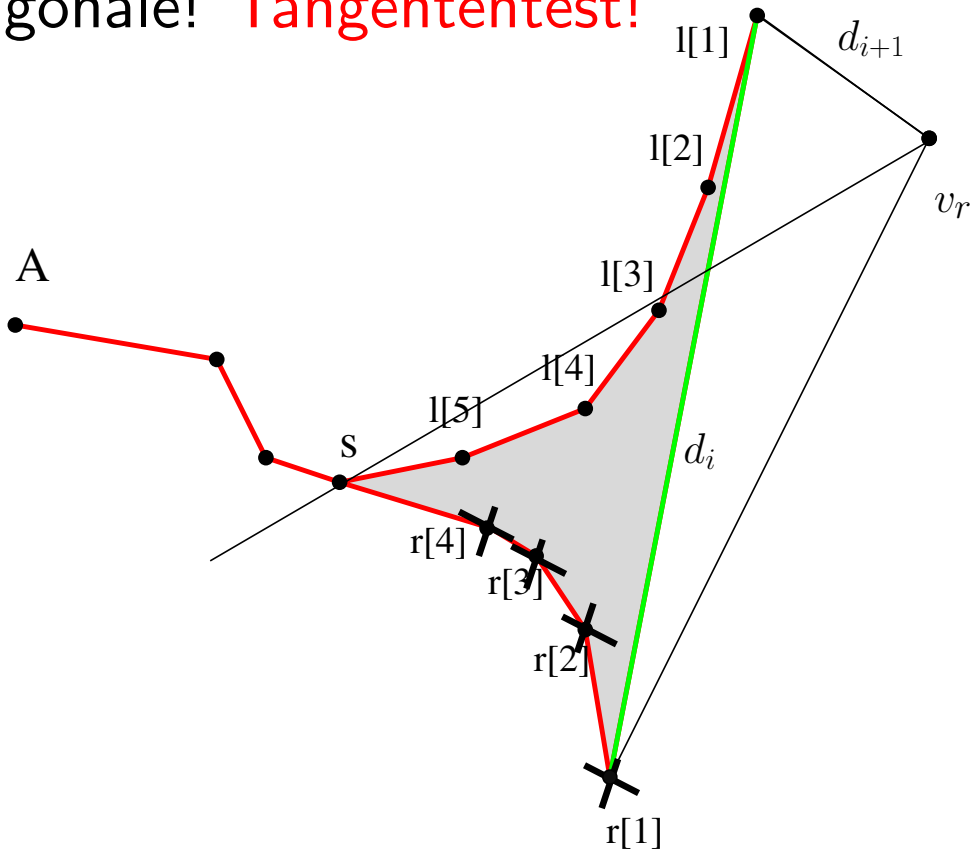
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



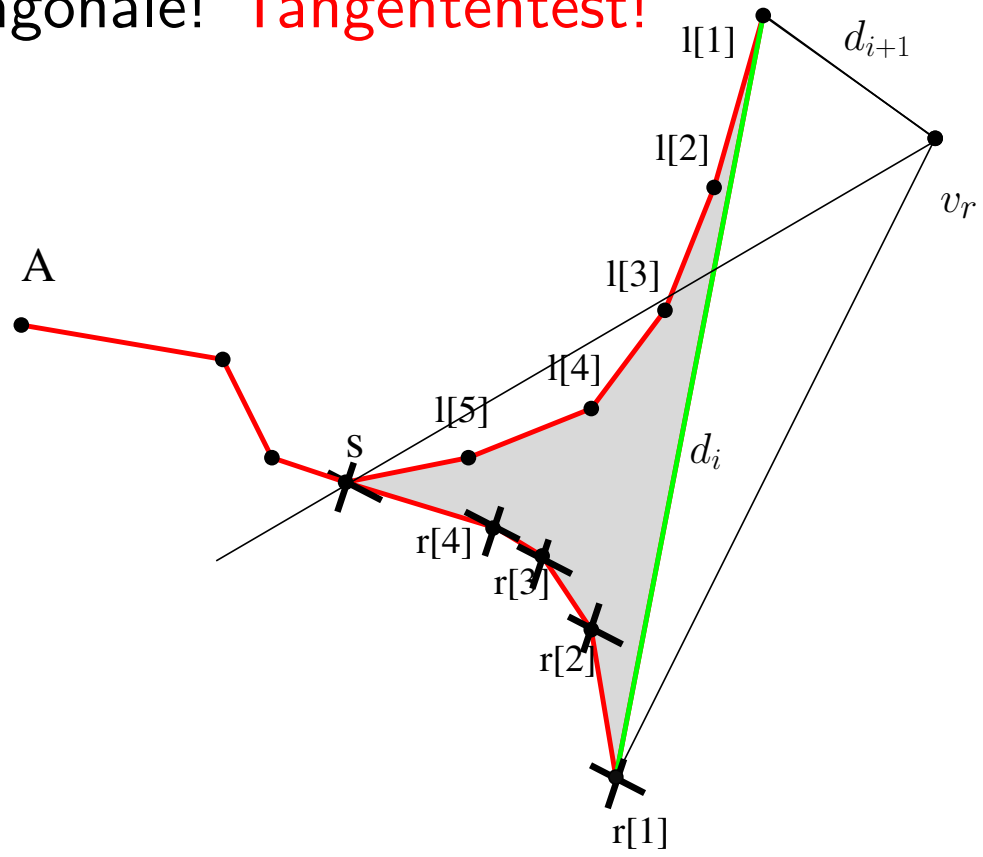
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



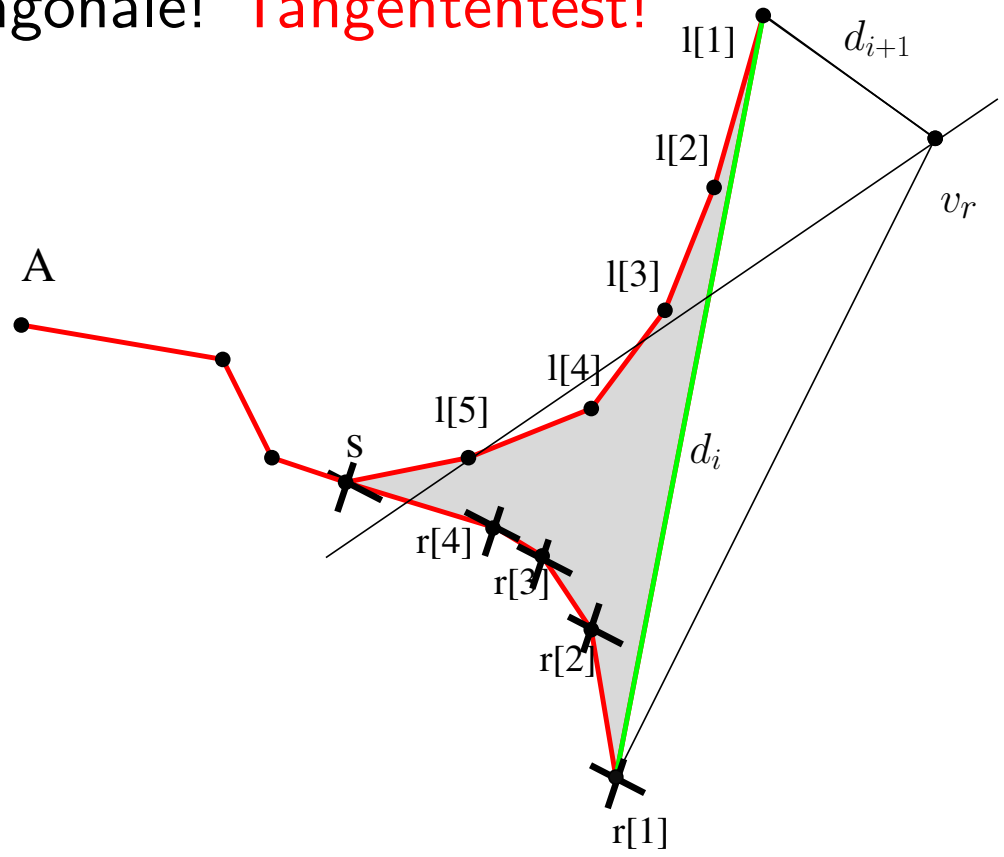
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



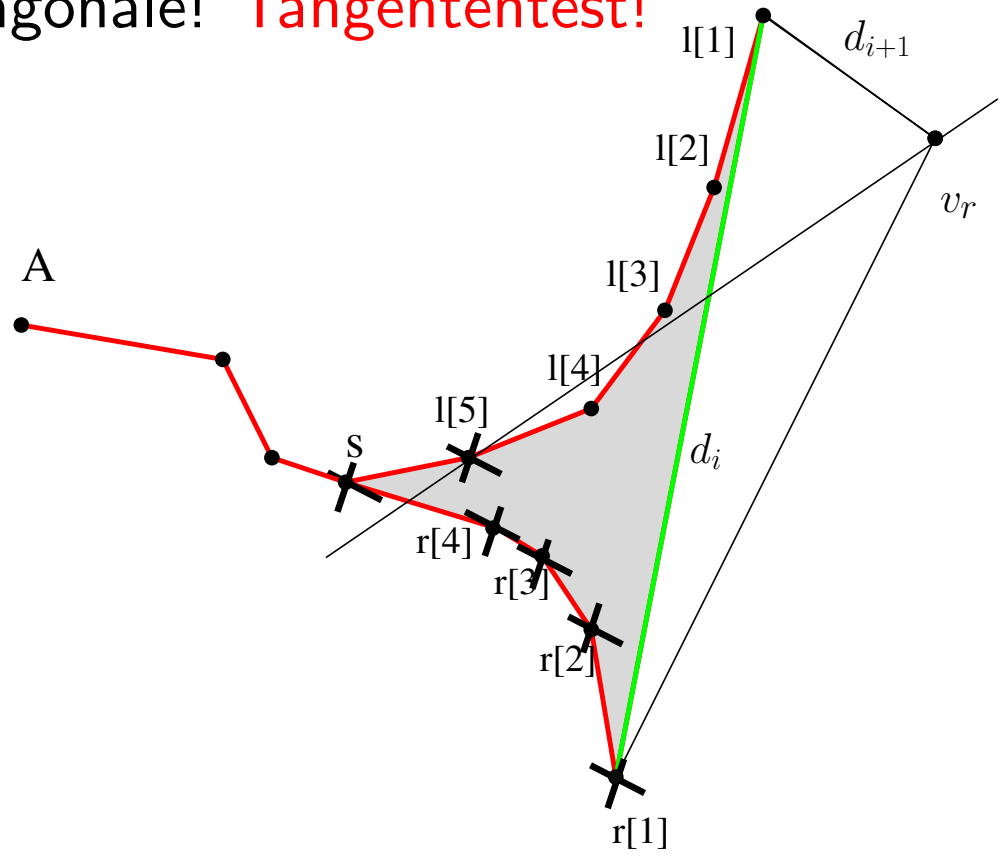
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



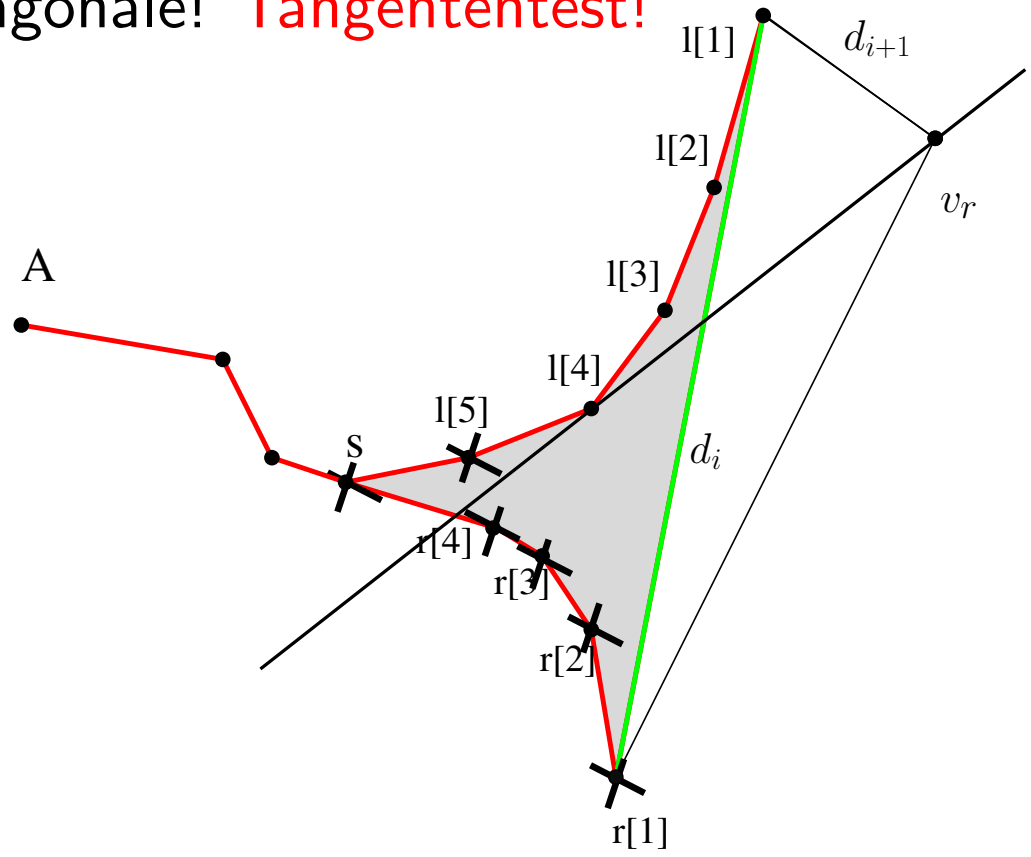
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



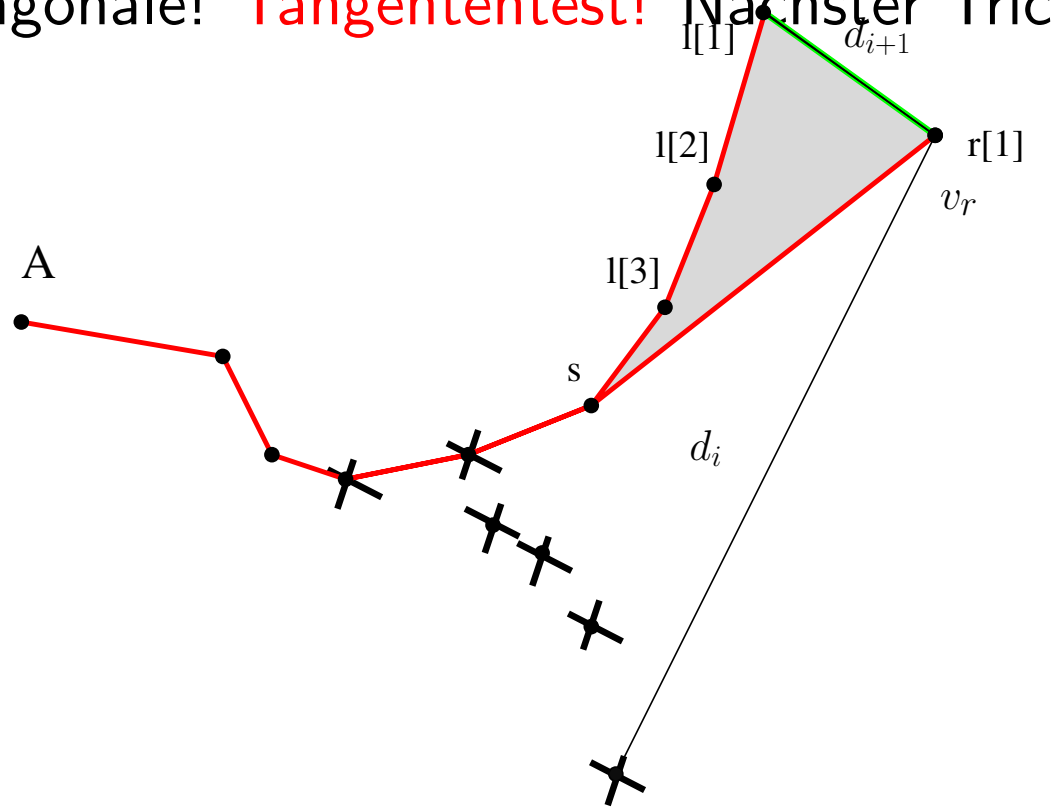
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest!



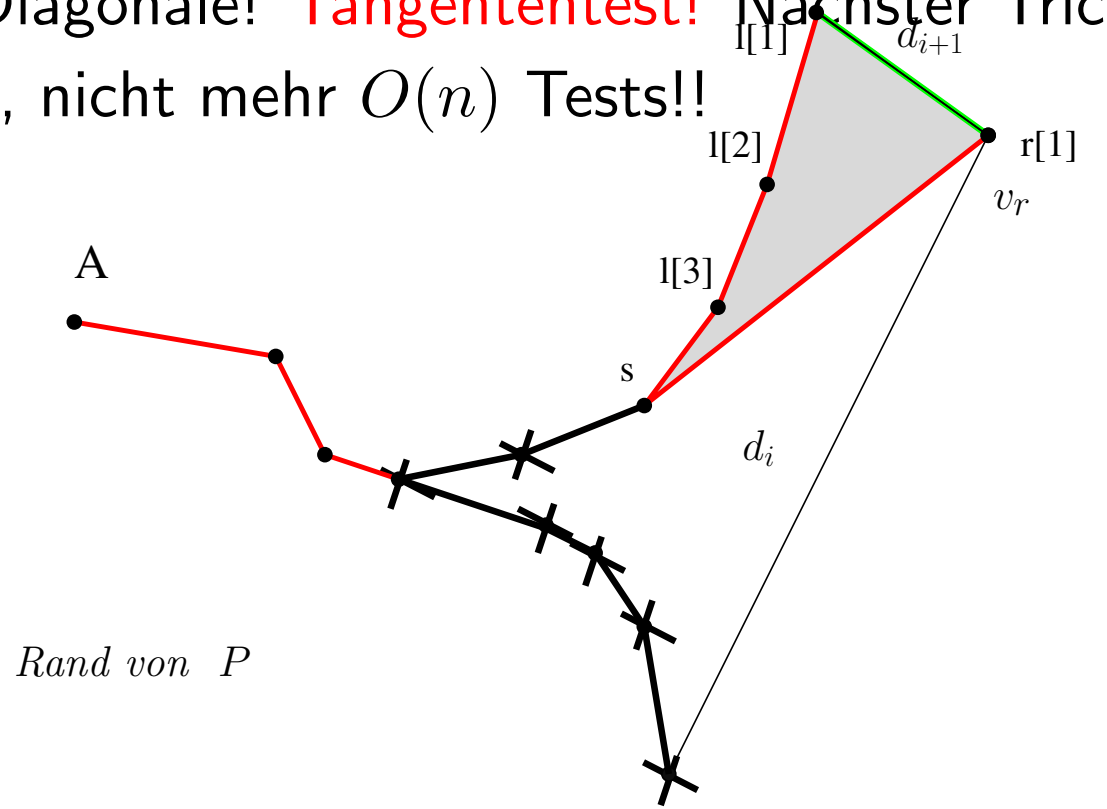
Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest! Nächster Trichter!



Verbleibt: Indukiver Schritt!

- Trichter Situation! DS: Zwei Ketten/Shortest Path!
- Nächste Diagonale! Tangententest! Nächster Trichter!
- Insgesamt, nicht mehr $O(n)$ Tests!!



Zusammenfassung: Lee/Preparata: Shortest Path innerhalb P

- Triangulation von P: $O(n)$ Zeit und Platz
- Dualer Graph der Triangulation: $O(n)$ Zeit und Platz
- DFS im Dualen Graphen: $O(n)$ Zeit
- Berechnung P': $O(n)$ Zeit und Platz
- Berechnung Shortest Path für End-Diagonale, induktiv: $O(n)$
- Shortest Path von A nach B in P in $O(n)$ Zeit und Platz
- Optimal für einzelne Anfrage!!
- Theorem 1.11