$\Rightarrow$ G enthält mind. $\frac{n}{2}$ Knoten vom Grad ≤ 11.

Denn: Ann: nein.

$\Rightarrow$ mind. $\frac{n}{2}$ Knoten sind vom Grad > 11 bzw. dann ≥ 12.

$\Rightarrow \sum_{v \in V} \text{degree}(v) \geq \frac{n}{2} \cdot 12 = 6n$

$\Rightarrow \lightning$ zu (*)

// (*) sagt $\sum_{v \in V} \text{degree}(v) < 6n$!

Betrachte Algorithmus:

Alg. startet mit Markierung der drei begrenzten Knoten.

In diesem Moment gibt es mind. $\frac{n}{2} - 3$ unmarkierte Knoten mit Grad ≤ 11.

Dann wählt der Alg. einen dieser Knoten und markiert ihn zusammen mit seinen höchstens 11 Nachbarn.

Das wird solange wiederholt bis es keine unmarkierten Knoten mit Grad ≤ 11 mehr gibt.

Deshalb werden während jeder Iteration höchstens 12 Knoten markiert.

Es gibt also mindestens $\left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$

Während jeder Iteration wird ein Knoten zu I hinzugefügt.

Ergebnis: $\forall$ Triang. G gilt: G enthält unabh. Knotenmenge I, so dass

1) $|I| \geq \left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$
2) $\forall v \in I$: degree(v) ≤ 11
3) I enthält keinen der drei Randknoten.
4) I kann in Zeit $O(n)$ berechnet werden.

$\longmapsto$ 3.4.5.9 Lemma: Sei I eine unabh. Knotenmenge von G gemäß 3.4.5.8.
Sei G' der Graph, der durch Entfernen von I aus G entsteht

$\Rightarrow$ 1). Jede Fläche von G' ist ein 1-faches Polygon. mit maximal 11 Knoten

2). Die äußeren Flächen (Dreiecke) von G und G' sind gleich.

$\longmapsto$ 3.4.5.10 Alg. zu Konstruktion der Datenstruktur D zur Darstellung der Folge $S_1 .. S_k$:

Datenstruktur D: Knoten + Pointer.

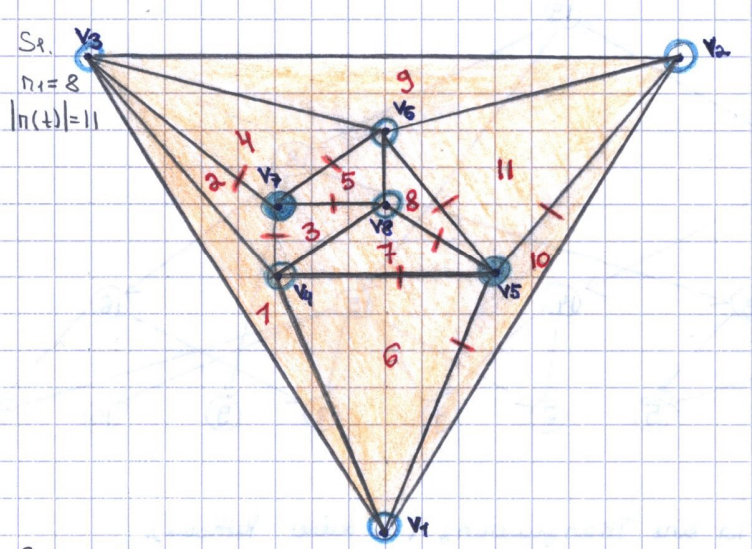
Knoten v speichert ein Dreieck t $\quad v = n(t)$.

Triangulierung S: Pkte, Kanten, Flächen

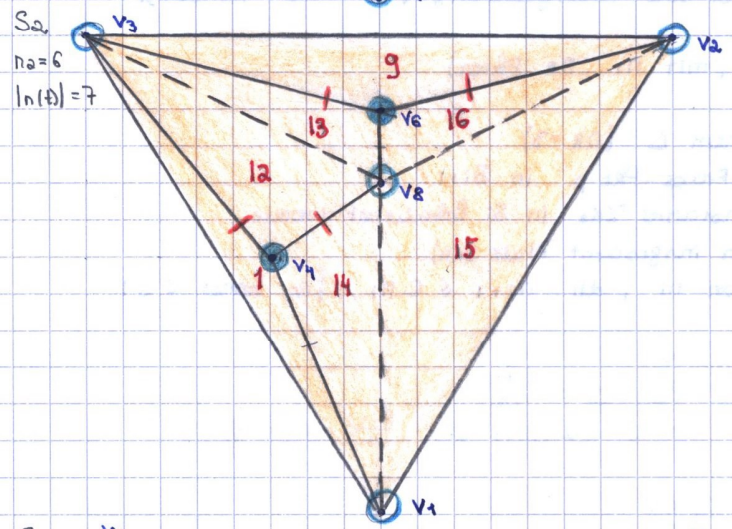
$n_i = |S_i|$ = # der Pkte.

1. $i \leftarrow 1$, $S_1 \leftarrow G$.
2. $\forall$ Dreiecke t in G do
3. erzeuge einen Knoten n(t)
4. od.
5. while $|S_i| > 3$ do
6. berechne unabh. Knotenmenge I in S_i mit mind. $\left\lceil \frac{1}{12} \left(\frac{n}{2} - 3 \right) \right\rceil$ Knoten, die nicht am Rand liegen und maximal Grad 11 haben.
7. Entferne die Knoten von I und ihre angegrenzten Kanten aus S_i
8. Triangulire den entstandenen Graphen und nenne das Resultat S_{i+1}.
9. $\forall$ Dreiecke t von S_{i+1}, die nicht in S_i enthalten sind (d.h. neue Dreiecke) do
10. erzeuge Knoten n(t)
11. $\forall$ Knoten n(t') mit $t' \in S_i$ und t' schneidet t do
12. erzeuge einen Pointer $n(t) \longrightarrow n(t')$
13. od
14. od
15. $i \leftarrow i+1$
16. od

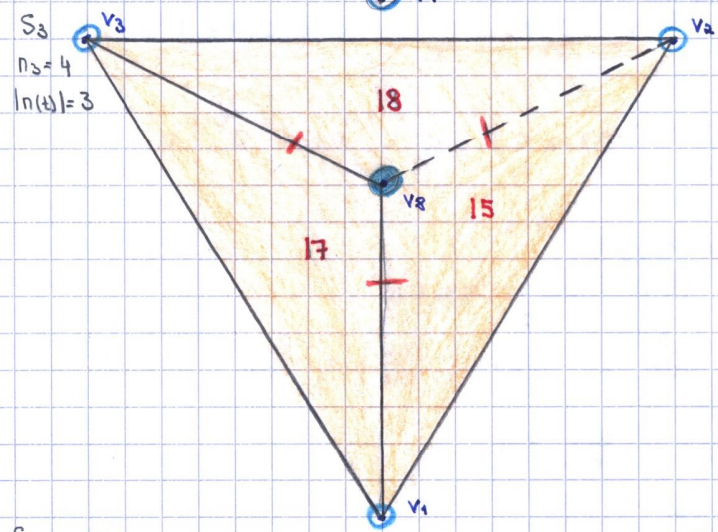
⟶ 3.4.5.11. Beispiel:



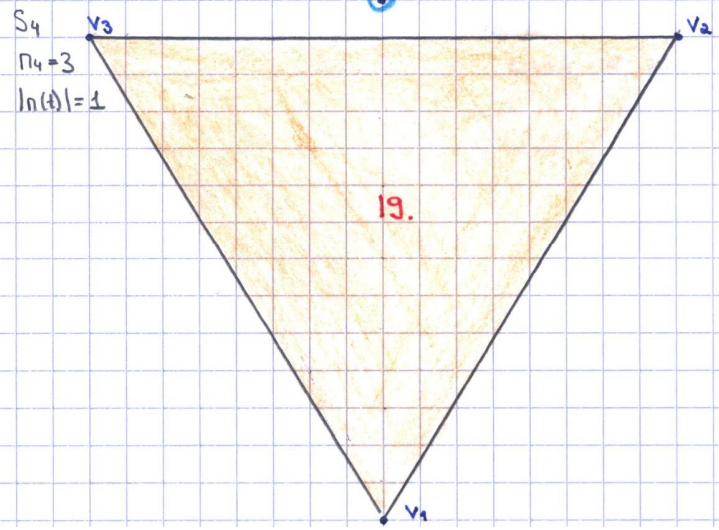
i = 1
I = { v5, v7 }
n(1) ... n(11) n(16)
neu Dreiecke: n(12), n(13), n(14), n(15)
n(12) ⟶ n(2), n(4), n(3), n(5)
n(13) ⟶ n(4), n(5)
n(14) ⟶ n(6), n(7)
n(15) ⟶ n(10), n(6), n(7), n(8), n(11)
n(16) ⟶ n(11), n(8)



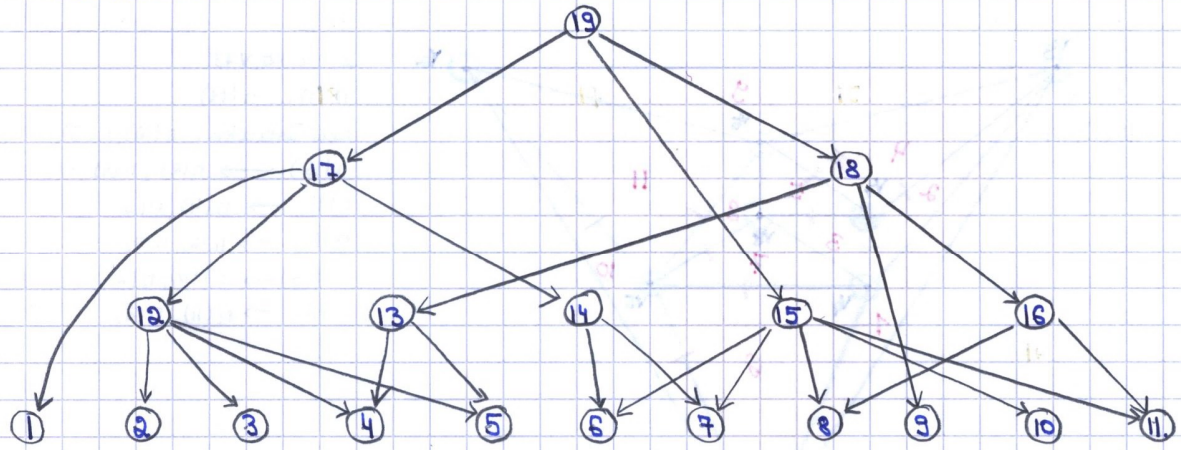
i = 2
I = { v6, v4 }
neue Dreiecke: n(16), n(17), n(18)
n(17) ⟶ n(1), n(14), n(12)
n(18) ⟶ n(13), n(9), n(16)



i = 3
I = { v8 }
neues Dreieck: n(19)
n(19) → n(15), n(17), n(18)



Datenstruktur D:



3.4.5.12. Zusammenfassung:

Planare Unterteilung G ist eine Triangulierung (bei dieser Methode).
(auch Rand ist Δ).

Ziel: Folge $S_1 .. S_R$ (mit $n_1 .. n_R$ Pkten)

✓ 1) $S_1 = G$

✓ 2) S_R besteht aus einem Δ ($n_R = 3$)

3) Lokalisierung von Frage-Pkt p in S_{i+1}
⇒ p kann in konstanter Zeit in S_i lokalisiert werden.

4) $S_1 .. S_R$ brauchen insgesamt Platz $O(n)$

5) n_{i+1} ist Bruchteil von n_i, d.h. $n_{i+1} \leq c \cdot n_i$ für konst $c \leq 1$
⇒ $R = O(\log n)$.

Es gilt: $n_{i+1} < n_i \ \forall i$

⇒ $n_{i+1} < c_i \cdot n_i$ für $c_i > \frac{n_{i+1}}{n_i}$

Ferner: $\frac{n_{i+1}}{n_i} < 1$ ⇒ c_i wählbar als $\frac{n_{i+1}}{n_i} < c_i < 1$

Gilt $\forall i$ und alle $c_i < 1$
⇒ wähle $c := c_R$ $\quad [\, n_3 < \underbrace{c_3}_{<1} \cdot n_2 < \underbrace{c_3 \cdot c_2}_{<1} \cdot n_1 < c_3 \cdot n_1 \,]$

⇒ $n_{i+1} < c \cdot n_i$

⇒ $3 = n_R < c \cdot n_{R-1} < c^2 \cdot n_{R-2} < \ldots < c^{R-1} \cdot n_1 = c^{R-1} \cdot n$

⇒ $n > \frac{3}{c^{R-1}}$

⇒ $\log_2 n > \log_2 3 + \log_2 \left(\frac{1}{c}\right)^{R-1}$

⇒ $\log_2 n > \frac{\ln 3}{\ln 2} + (R-1) \cdot (\underbrace{\log_2 1}_{=0} - \log_2 c)$

⇒ $R - 1 < \frac{\log n - \frac{\ln 3}{\ln 2}}{-\frac{\ln c}{\ln 2}} = \log n \cdot \left(-\frac{\ln c}{\ln 2}\right) + \frac{\ln 3}{\ln c}$

⇒ $R < \underbrace{-\frac{\ln c}{\ln 2}}_{\substack{>0 \text{ weil } c<1 \\ \text{const}}} \cdot \log n + \underbrace{\frac{\ln 3}{\ln c} + 1}_{\text{const}}$

⇒ $R \leq M \cdot \log n$ mit $M = \text{konst}$

⇒ $R = O(\log n)$.

3.4.5.13. Algorithmus für Point-Location:

Eingabe: Pkt q
Datenstruktur $\mathcal{D}$.

$\mathcal{D}$.search(q) liefert Dreieck von G, das q enthält.

if q außerhalb Dreieck i.d. Wurzel then
  Aussage: "äußeres Gebiet von G".
else
  $v \leftarrow$ Wurzel
  while v kein Blatt (dh. $\mathcal{D}$.outdeg(v)>0) do
    forall Knoten u mit $\exists$ Pointer $v \to u$ do
      if q innerhalb Dreieck von u then
        $v \leftarrow u$;
      fi
    od
  od
  Ausgabe: Dreieck von v
fi

Schleifeninvariante: Während der Ausführung der while-Schleife gilt stets, dass $q \in$ Dreieck von v.

Bemerkung: Platz:

$$\#\text{Knoten} = n_1 + n_2 + \dots + n_R \leq$$
$$\leq n_1 + c \cdot n_1 + \dots + c^{R-1} \cdot n_1 =$$
$$= n_1 \sum_{\nu=0}^{R-1} c^\nu = n_1 \cdot \frac{c^R - 1}{c-1} = n \cdot \underbrace{\frac{c^R - 1}{c-1}}_{\text{const}} = \mathcal{O}(n)$$

⟶ 3.4.5.13. Algorithmus für Point Location.

Eingabe: Pkt q, Datenstruktur D.
$D.search(q)$ liefert Dreieck von G, dass q enthält
if q außerhalb Dreieck in der Wurzel then
    Aussage: äußeres Gebiet von G.
else
    $v \leftarrow$ Wurzel
    while v kein Blatt (dh. $D.outdeg(v) > 0$) do
        forall Knoten u mit $\exists$ Pointer $v \rightarrow u$ do
            if q innerhalb Dreieck von u then
                $v \leftarrow u$;
            fi
        od
    od
    Ausgabe: Dreieck von v
fi.

Schleifen-Invariante: Während der Ausführung der while-Schleife gilt stets, dass $q \in$ Dreieck von v.

Knoten $\hat{=}$ Objekt einer Klasse "Dreieck".
→ mit orient. Teste ...

Kapitel IV: Bewegungsplanung i.d. Ebene.

4.1. Einführung:

4.1.1. Allgemeines Problem:

Geg: Eine Szene S von Hindernissen (Polygone, Segmente, ...), ein Roboter R (Polygon, Kreisscheibe, ...) und zwei Pkte A, B.

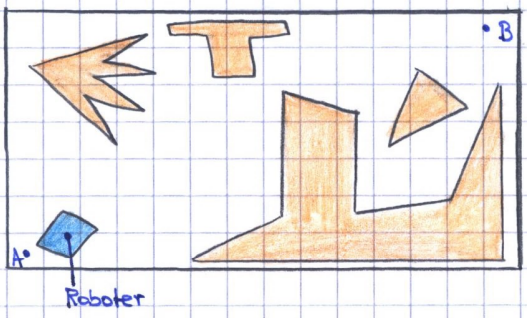
Aufgabe: Beantworte die Frage, ob R von A nach B bewegt werden kann, ohne mit Hindernissen aus S zu kollidieren bzw. gib einen kollisionsfreien Weg an.

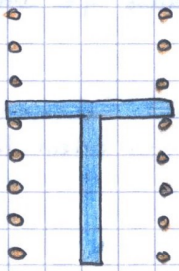
→ Piano Moves Problem.

4.1.2. Bemerkungen:

1) Wir behandeln zwei Spezialfälle:
   - S ist Menge von Kurvensegmenten u. R ist eine Kreisscheibe.
   - S ist Menge von konvexen Polygone, R ist konvexes Polygon.
2) Bewegung: nur Translationen.

Es gibt Vielzahl weiterer Varianten:
- Rotationen erlaubt
- Kürzester Weg.
- Sichester Weg.





4.2. Problem 1: R ist Kreis und S Menge von Segmenten.

4.2.1. Idee: Versuche den sichersten Weg zu finden, dh. R hält immer max. möglichen Abstand zu den Segmenten.

4.2.2 Frage: Was sind die Orte mit maximalen Abstand zu allen Hindernissen?

4.2.3 Antwort: Alle Pkte auf (Knoten & Kanten des Voronoi-Diagramms von S. VD(S)

4.2.4. Voronoi Diagramm von Segmenten i.d. Praxis:

→ annähern durch setzen von Hilfspkten (genügend dicht) und Berechnung des VD's dieser Pkte.

Anschließend Entfernung aller Voronoi-Kanten, die von Segmenten geschnitten werden.

4.2.5 Definition: $\forall p \in \mathbb{R}^2$ ist

a) Freiheit (p) := minimaler Abstand von p zu allen Hindernissen.

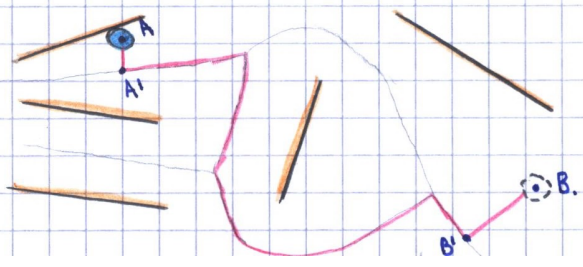
b) p heißt frei $\Leftrightarrow$ Freiheit (p) $\geq r$, wobei r = Radius von R.
(dh. Kreisscheibe R kann mit Mittelpkt auf p plaziert werden, ohne ein Segment zu schneiden).

c) $FP := \{p \in \mathbb{R}^2 : p \text{ ist frei}\}$.

4.2.6. Idee für Algorithmus:

- Bewege R von Anfangsposition A (Ann. freie Position) auf nächste Voronoi Kante (→ Position A').
- Bewege R auf Voronoi-Kanten, die mindestens Freiheit r haben, so nah wie möglich an B heran. (→ Position B')
- Bewege R von B' nach B (Ann. B ist frei).

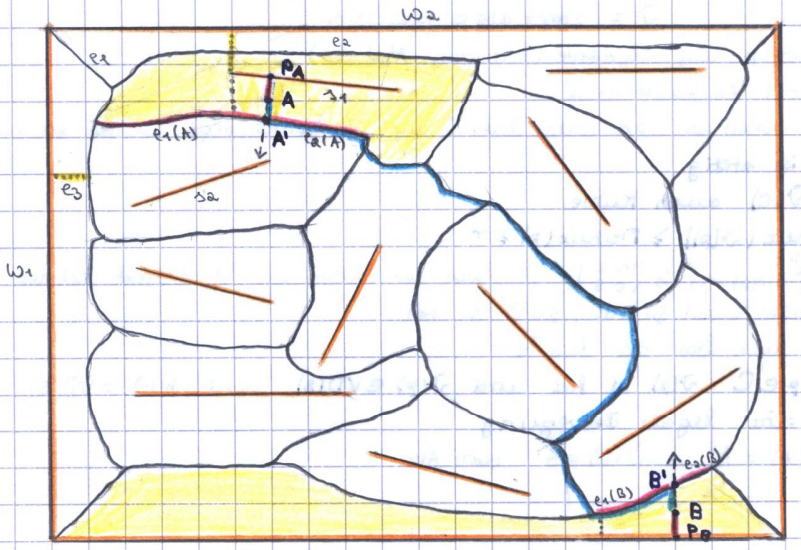
4.2.7 Beispiel:



4.2.8 Algorithmus:

1. Konstruiere VD(S). Speichere für jede Kante die beiden Orte, die e definieren. (entweder Segmentendpkte oder Segmentwand).
   Und Freiheit(e) := minimaler Abstand eines Pktes auf e zu einem Segment.
   (Freiheit(e) ≥ r ⇒ R kann über e bewegt werden).
   ↑ Preprocessing.
2. Bestimme für A und B jeweils den nächsten Pkt P_A bzw. P_B auf einem Segment durch lineare Suche auf S (→ Point Location)
   (P_A, P_B sind Schnittpkte des Lots von A bzw. B auf nächstes Segment).
   Dh: Betrachte Lots von A bzw. B auf alle Segmente in S, suche nach einem Lot, der eine Voronoi-Kante schneidet → lin. Suche.
3. A' ← erster Schnittpkt des Strahls $\overrightarrow{P_A A}$ mit VD(S)
   B' ← erster Schnittpkt des Strahls $\overrightarrow{P_B B}$ mit VD(S)
   Falls A' bzw. B' keine Voronoi-Knoten, dann mache sie zu künstlichen Knoten durch Spalten der entsprechenden Kante e in e_1, e_2.
   → Berechne Freiheit(e_1), Freiheit(e_2)!
4. Suche einen Pfad P in VD(S) von A' nach B' mit Freiheit(e) ≥ r ∀ e ∈ P.
   z.B. durch DFS oder BFS.
   Falls kein solcher Pfad existiert, melde "keine Lösung".
   → STOP.
5. Der Weg A → P → B ist eine mögliche Bewegung.

4.2.9 Bsp:



1. für e_1 speichere: (ω_1, ω_2) — Freiheit(e_1) = 0 cm
   für e_2 speichere: (s_1, ω_2) — Freiheit(e_2) = 0,5 cm
   für e_3 speichere: (Endpkt von s_2, ω_1) — Freiheit(e_3) = 0,5 cm. usw. ...
2. Berechne Lots von A und B auf alle Segmente (ind. Ränder)
   Von den berechneten suche die mit dem kleinsten Abstand aus (lin. Suche)
   Dadurch wissen wir in welcher Region sich Ort A und B befinden (point Location)
   Im Bsp. sind die jeweiligen Regionen gelb.
3. Teile e(A) in $e_1(A)$ und $e_2(A)$ und e(B) in $e_1(B)$ und $e_2(B)$
   Freiheit($e_1(A)$) = 0,6 cm, Freiheit($e_2(A)$) = 0,7 cm
   Freiheit($e_1(B)$) = 0,3 cm, Freiheit($e_2(B)$) = 1 cm.
4. ∀ e ∈ blau gilt Freiheit(e) ≥ r

4.2.10. Lemma:

1) Falls A bzw. B frei ist $\Rightarrow$ $\exists$ Bewegung von A nach A' bzw. von B nach B'.
2) $\exists$ Bewegung von A nach B $\Leftrightarrow$ $\exists$ Bewegung von A' nach B', die nur Voronoi-Kanten besitzt.

Beweis:

1). Sei $VR(s)$ die Voronoi-Region, die A enthält
$\Rightarrow p_A \in S_v \in S$
Seien A, B frei $\Rightarrow$ Freiheit(A) $\geq r \wedge$ Freiheit(B) $\geq r$
$\text{Freiheit}(A) := \min_{s \in S} |A - s|$, $\text{Freiheit}(B) := \min_{s \in S} |B - s|$

Es gibt: $r \leq \text{Freiheit}(A) \leq \text{Freiheit}(p) \quad \forall p \in \overline{AA'}$
$\Rightarrow$ Es gibt eine legale Bewegung von R von A nach A'
weil $\text{Freiheit}(p) \geq r \quad \forall p \in \overline{AA'}$.
$B \to B'$ analog.

2) „$\Leftarrow$": $\exists$ Bewegung von A' nach B', die nur Voronoi-Kanten besitzt.
Beh folgt nach Teil 1), falls A und B frei sind. (?)

„$\Rightarrow$": z.z. $\exists$ Bew von A' nach B', die nur Vor. K. besitzt.
Eine beliebige legale Bewegung von A nach B wird durch eine Kurve $C \subset \mathbb{R}^2$ definiert. (oBdA eine einfache Kurve).
$\forall p \in C$ ist p frei, dh. $\text{Freiheit}(p) \geq r$
Betr. Abb. $\vec{v}: \mathbb{R}P \to VD(S)$ bea: $C \subset \mathbb{R}P$ $(D(\vec{v}) = C)$
die durch Schritte 2 bis 3 des Algorithmus definiert ist.
$\vec{v}(p) = p'$. $\vec{v}$ = Streckung o projektion
Betr. Lots von p auf Segmente (mit min. Abstand)
Schnittpkt mit Vor.kante ist p'.
Bea: Jetzt wissen wir noch nicht, dass C auf Vor.kanten liegt, wollen genau das zeigen

$\Rightarrow$ a). $\vec{v}$ ist stetig
$\Rightarrow \vec{v}(C)$ auch Kurve.
b) $\text{Freiheit}(\vec{v}(p)) \geq \text{Freiheit}(p) \geq r$
$\min''_{s \in S} |\vec{v}(p) - s| \geq \min_{s \in S} |p - s|$ weil wir immer den kl. Abstand nehmen.
p haben zuerst gesucht, $p \in C \Rightarrow$ Beh.
$\geq r$ auch klar da Bew. $\exists$.
$\Rightarrow \forall p \in C$: $\vec{v}(p)$ ist frei und $\vec{v}(p) \in VD(S)$ weil $W(\vec{v}) = VD(S)$
$\Rightarrow \vec{v}$ ist eine legale Bewegung
Bea: $\vec{v}(A) = A' \wedge \vec{v}(B) = B'$ nach Alg.

4.2.11. Laufzeit:

Schritt 1: $O(n \log n)$ durch Plane Sweep.
Schritt 2: $O(n)$ lin. Suche in S
Schritt 3: $O(n)$ lin Such auf $VD(S)$, $\overrightarrow{p_A A}$ berechnen $\to O(1)$, A' berechnen $\to O(1)$, e_1, e_2 aufteilen und $\text{Freiheit}(e_1)$, $\text{Freiheit}(e_2)$ Berechnen $\to O(1)$
Überprüfen ob A' bzw B' Vor. Knoten $\to O(n)$, denn lin. Suche bei den gespeicherten Voronoiknoten.
Schritt 4: $O(n)$ Pfadsuche in $VD(S)$ (ZB DFS)
Schritt 5: $O(n)$ Ausgabe des Pfades.

4.2.12. Satz: Bewegungsplanungsproblem für eine Kreisscheibe in einer Szene von n Liniensegmenten in der Ebene kann in Zeit $O(n \log n)$ und Platz $O(n)$ gelöst werden.
(Platz $O(n)$: weil in einzelnen 5 Schritten immer Platz $O(n)$.)

Beweis: s.o.