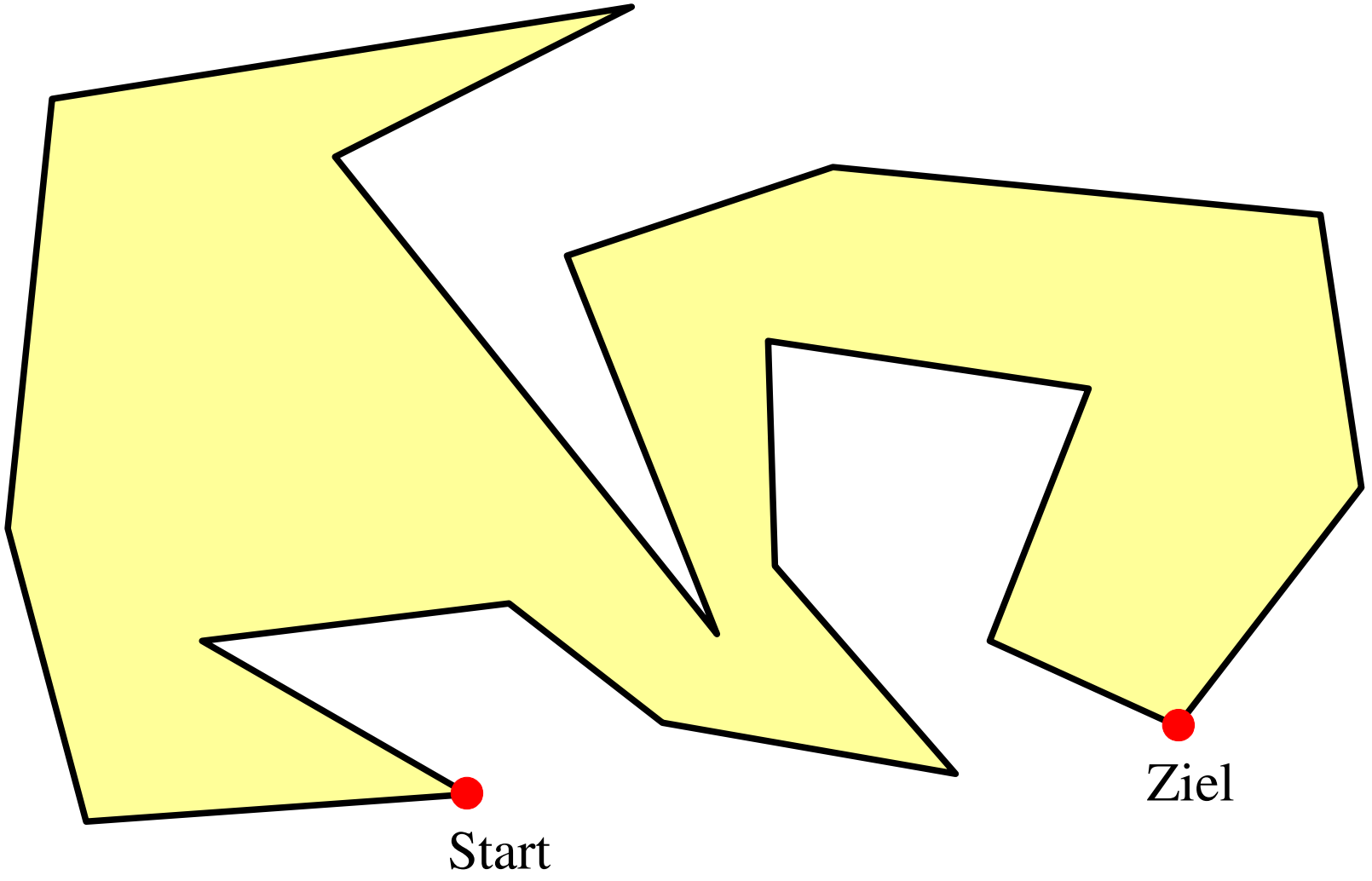
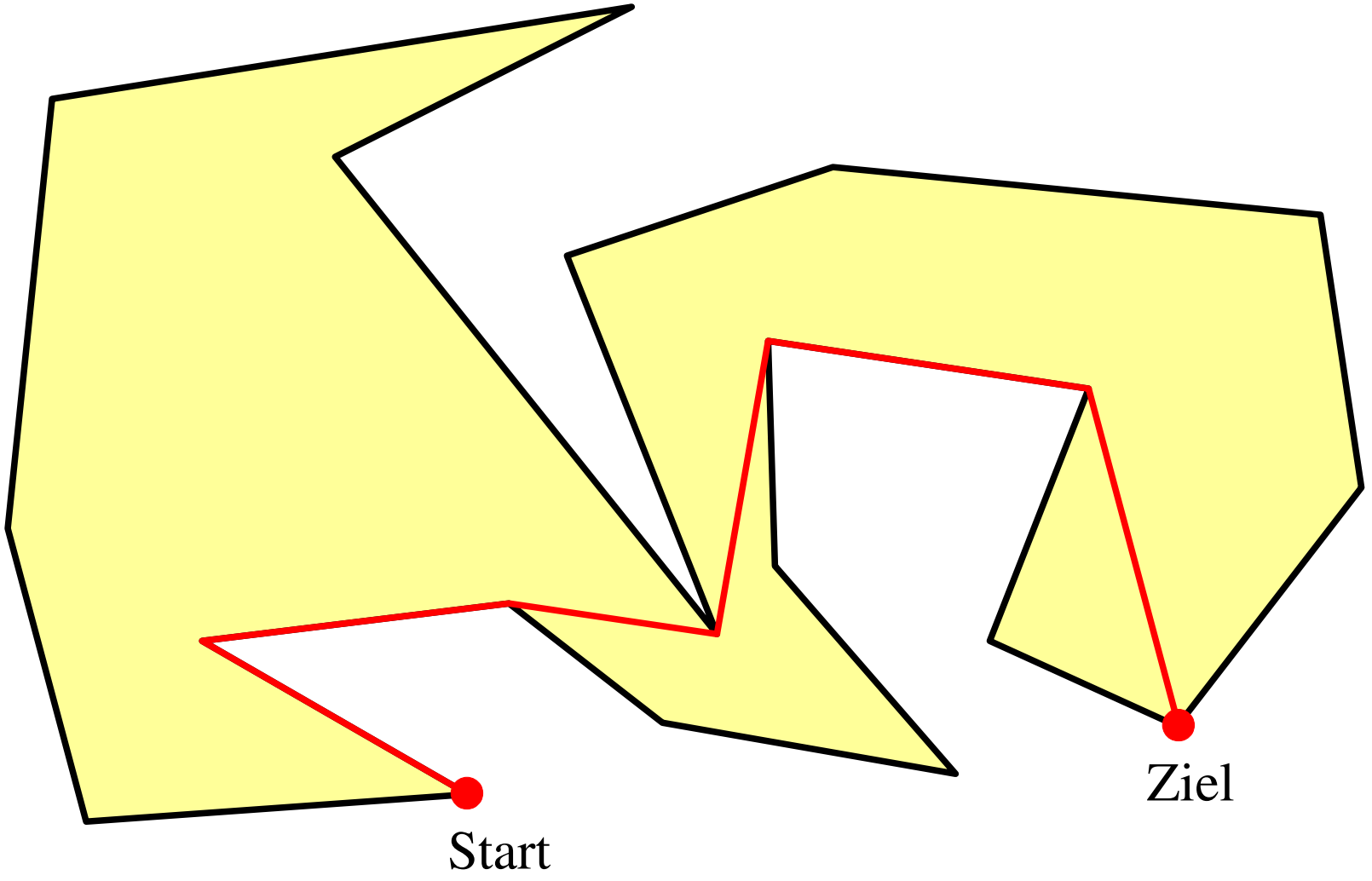
Kürzeste Wege in Polygonen

1. Beschreibung der Aufgabenstellung

Gegeben ist ein einfaches Polygon sowie eine Startecke und eine Zielecke.



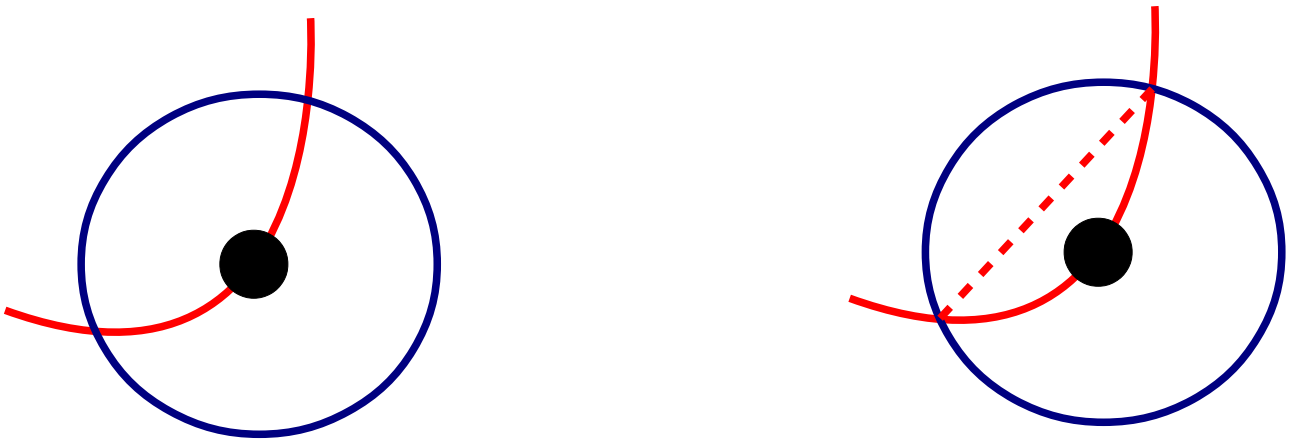
Gesucht ist ein kürzester Weg von der Startecke zur Zielecke, der innerhalb des Polygons verläuft.



Ein kürzester Weg zwischen zwei Punkten in einem einfachen Polygon ist immer eine **Folge von Strecken**.

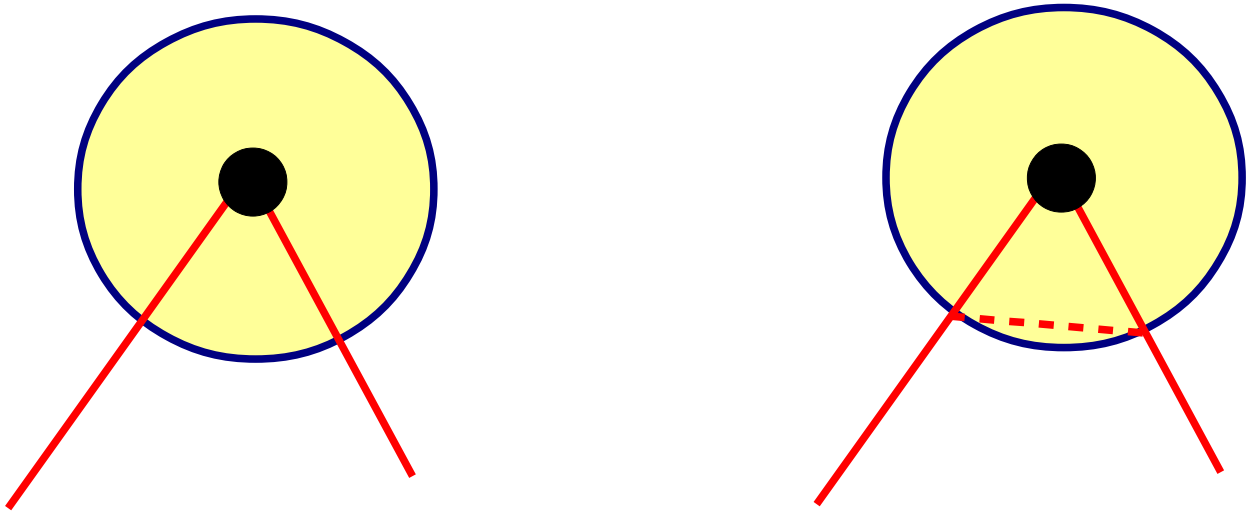
Angenommen dem wäre nicht so.

Dann erhält man sofort einen Widerspruch, denn man findet leicht einen noch kürzeren Weg:



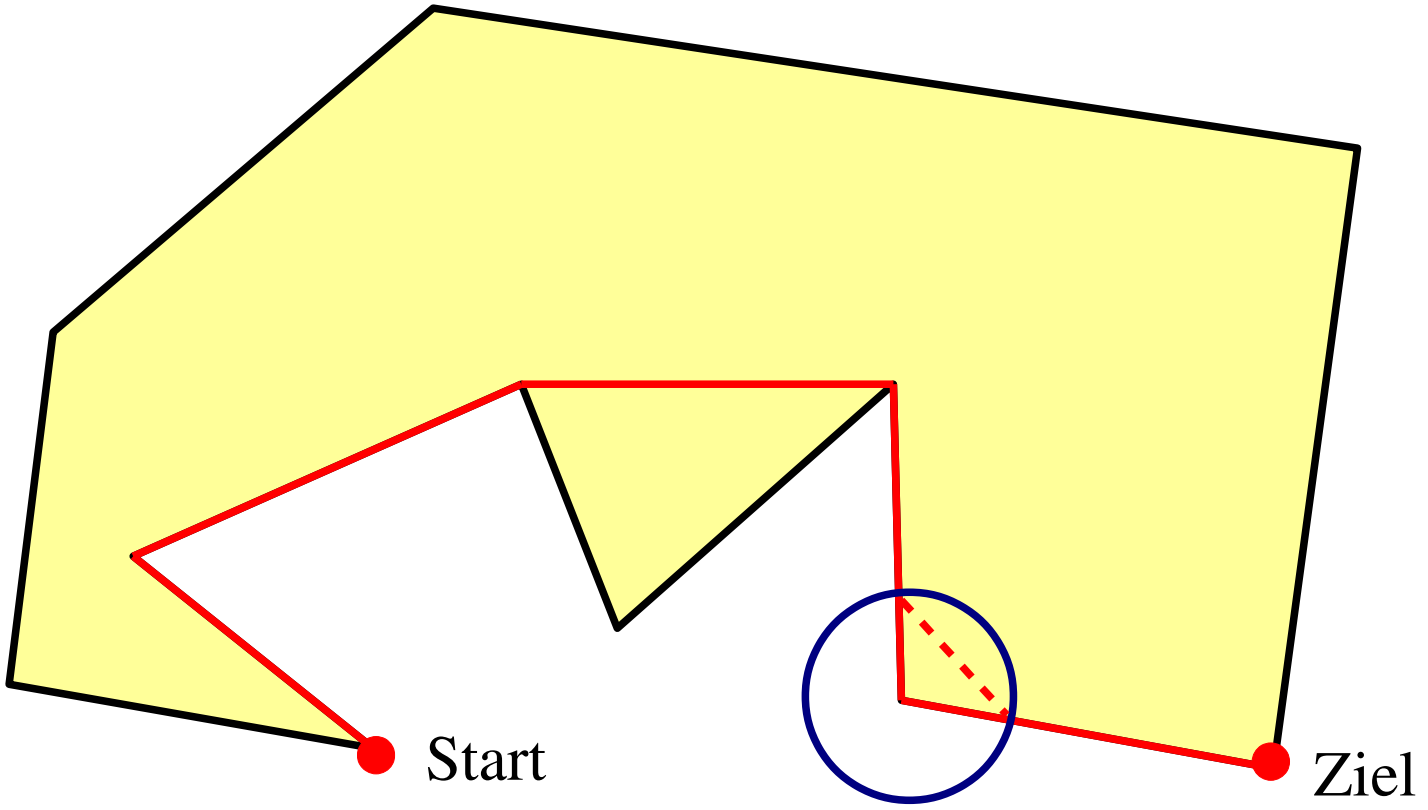
Die Ecken eines kürzesten Weges sind immer **Ecken des Polygons**.

Auch hier ergibt sich sofort ein Widerspruch, wenn man annimmt, dies wäre nicht so.



Alle Ecken eines kürzesten Weges, ausgenommen die Startecke und die Zielecke, müssen **reflexe Ecken** des Polygons sein.

Damit müssen auch alle Ecken eines kürzesten Weges reflex sein.



Kürzeste Wege sind **eindeutig bestimmt**.

Nehmen wir an es gäbe **zwei verschiedene** kürzeste Wege.

Dann **verzweigen sich** diese auf dem Weg vom Startpunkt zum Zielpunkt in einem Punkt *a* und kommen in einem Punkt *b* das erste Mal nach *a* wieder zusammen.

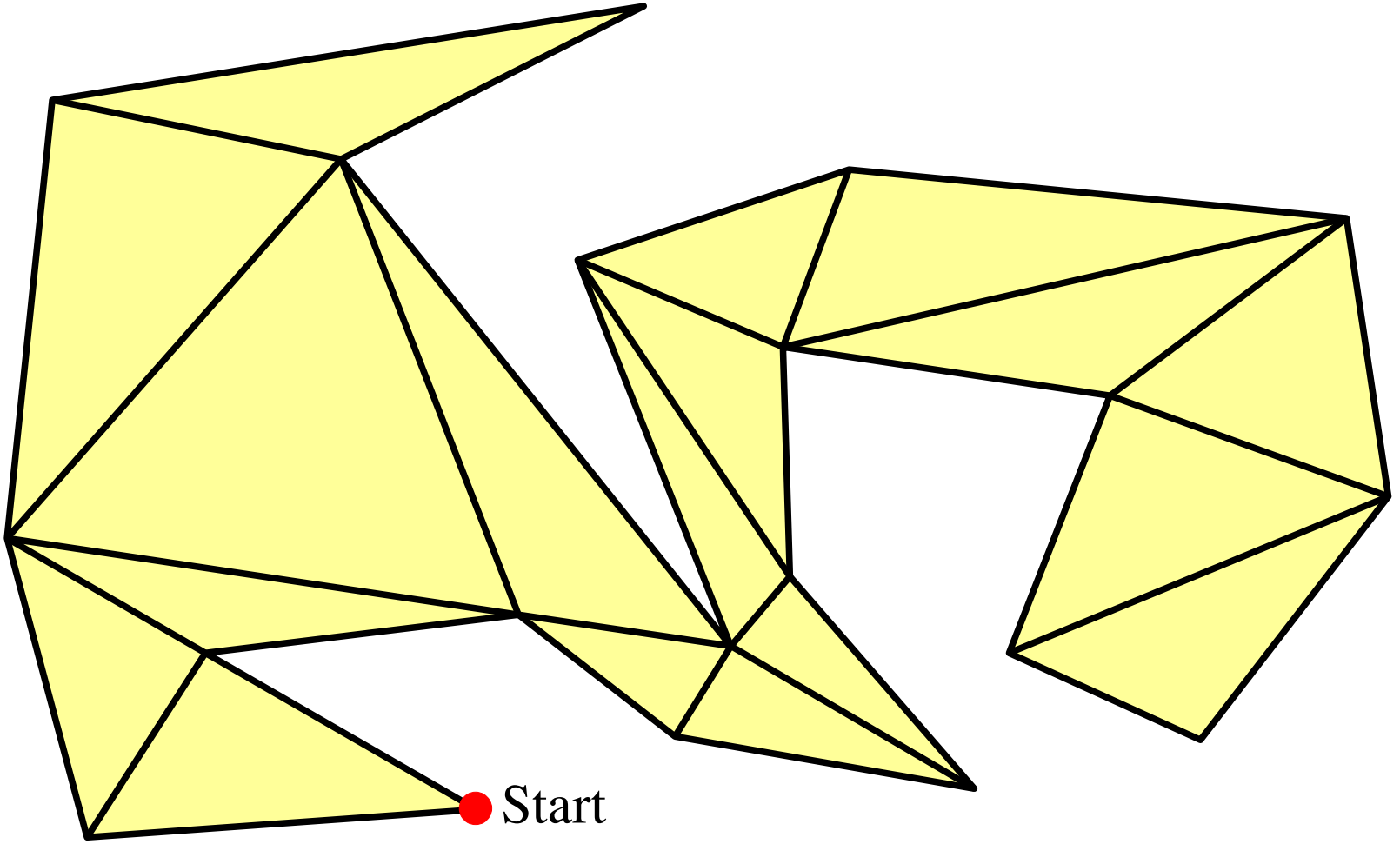
Wir betrachten das **Polygon *H***, welches von den beiden kürzesten Wegen zwischen *a* und *b* berandet wird.

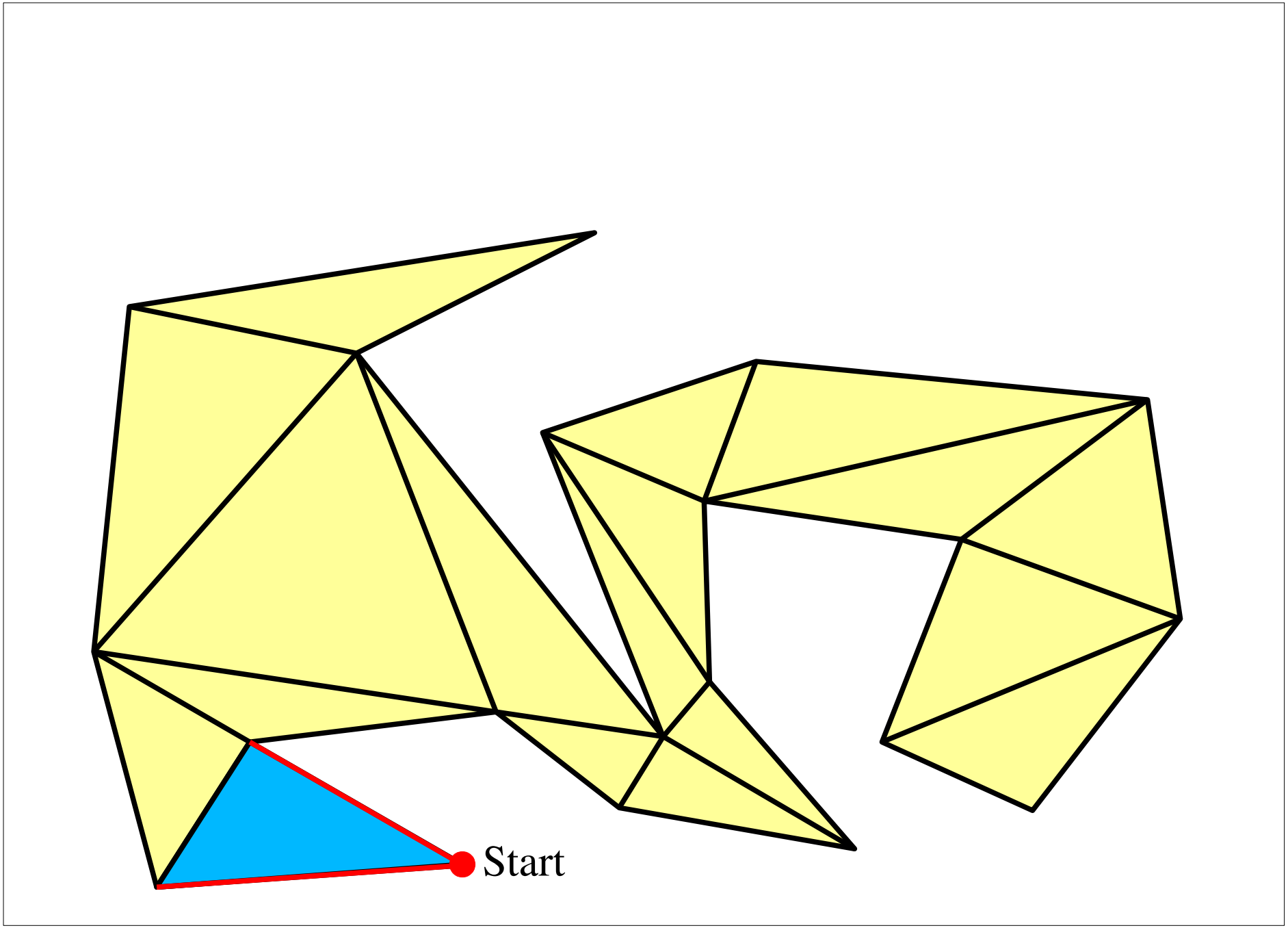
Von den Innenwinkeln von *H* sind **höchstens zwei** kleiner als π.

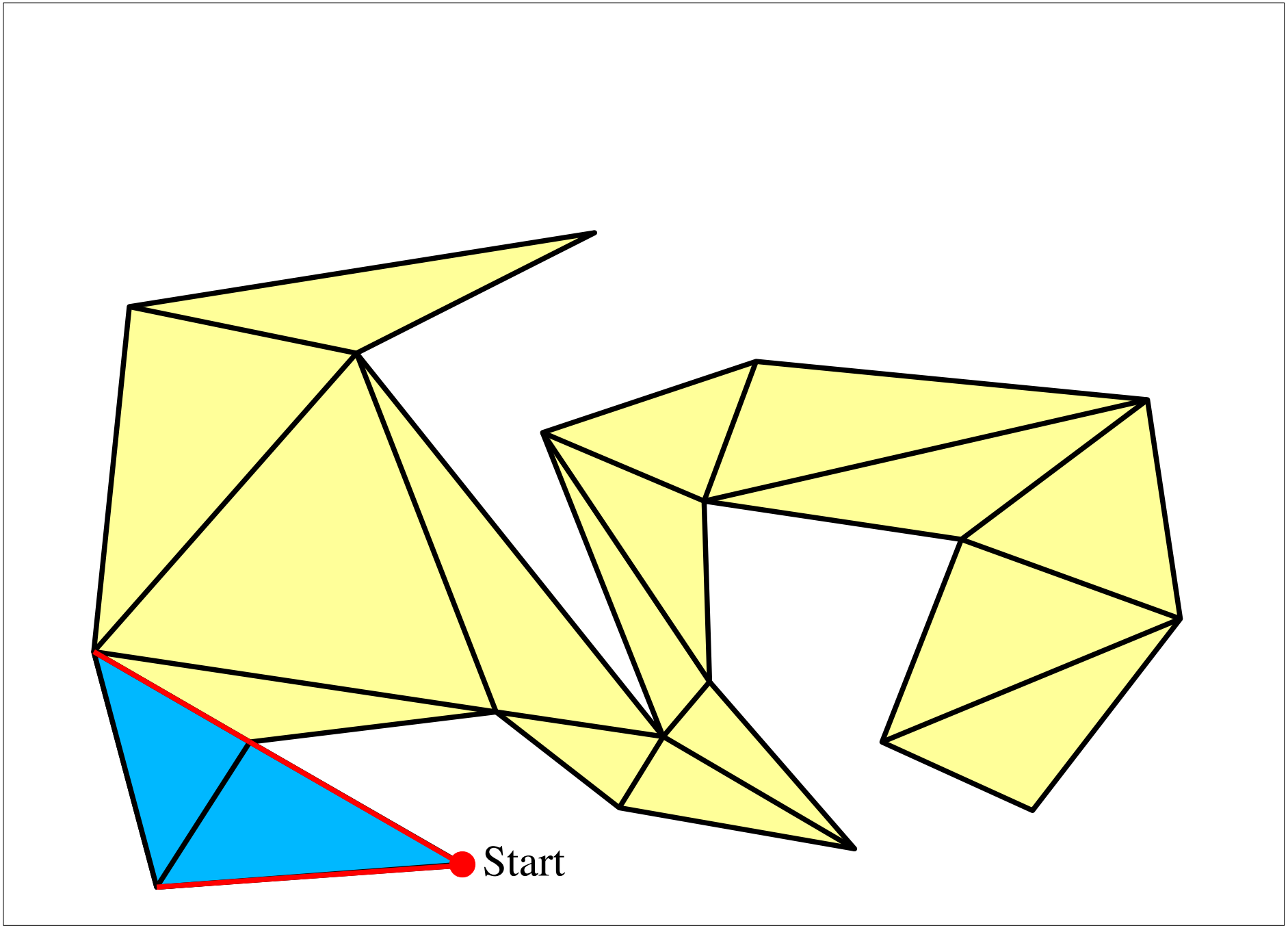
Die Innenwinkelsumme von *H* ist also größer als $(z-2)\pi$, wobei *z* die Anzahl der Ecken von *H* ist. Das kann aber nicht sein.

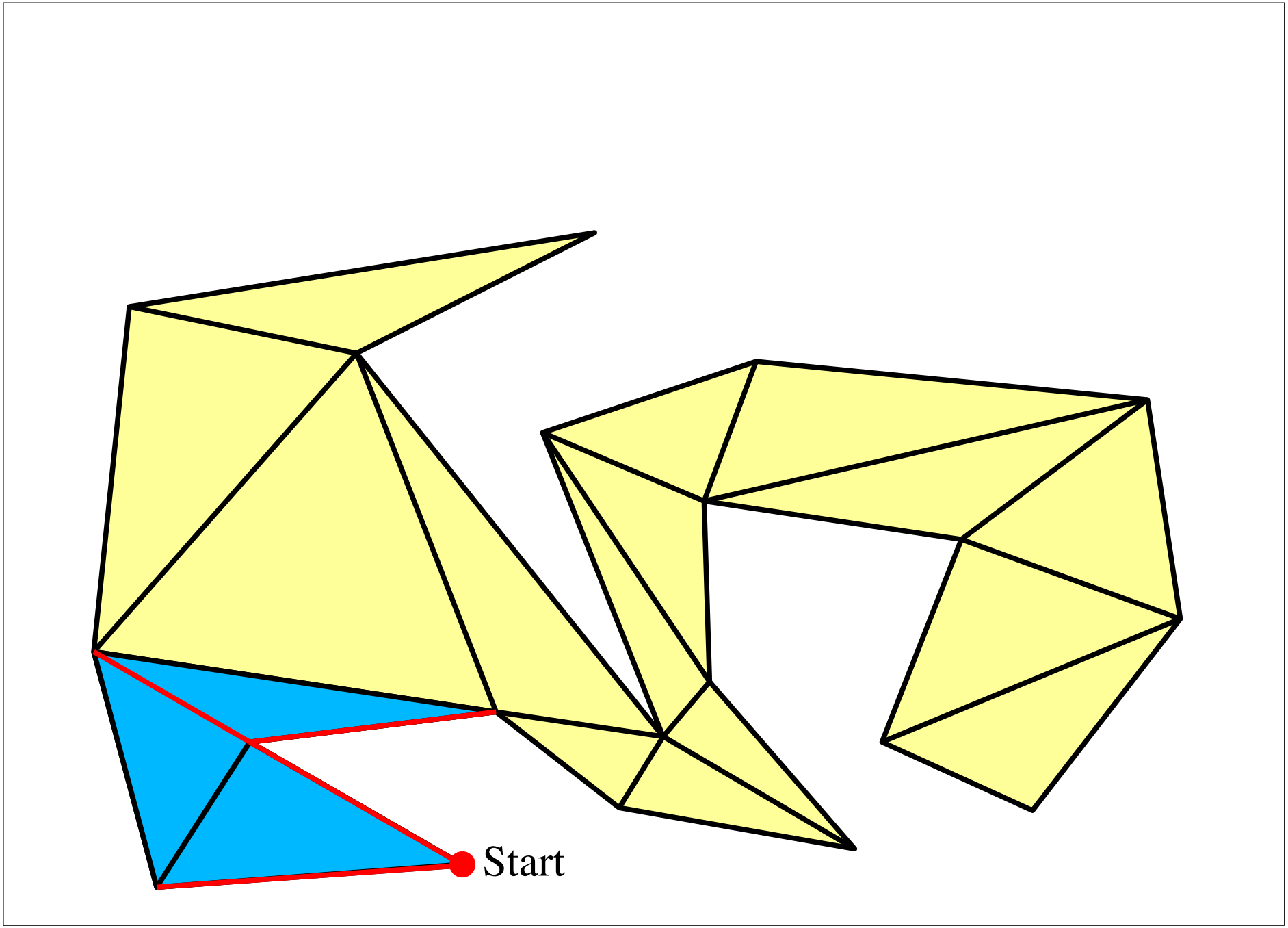
2. Grundidee für einen Algorithmus

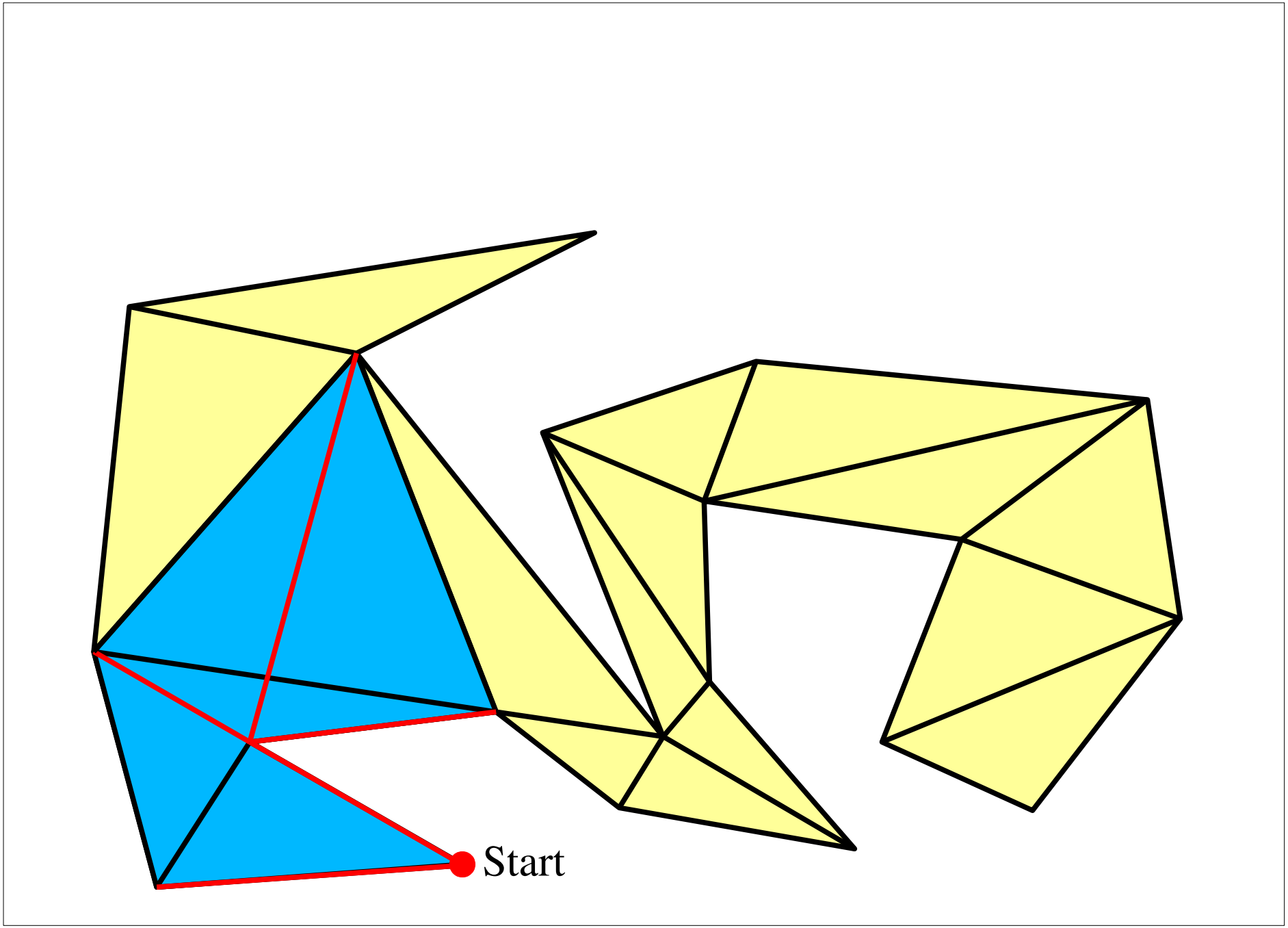
Zuerst triangulieren wir das Polygon. Dann gehen wir von Dreieck zu Dreieck.

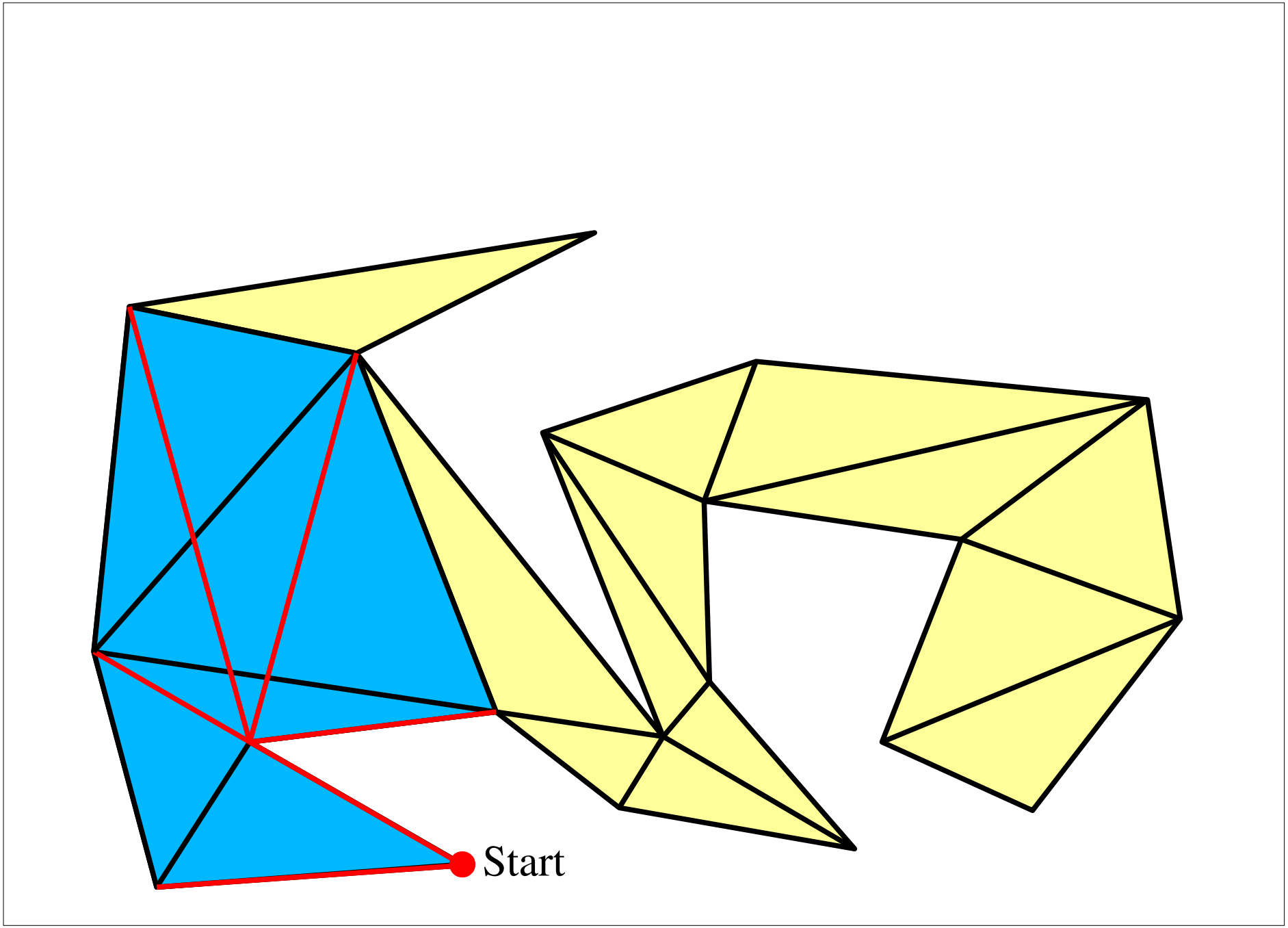


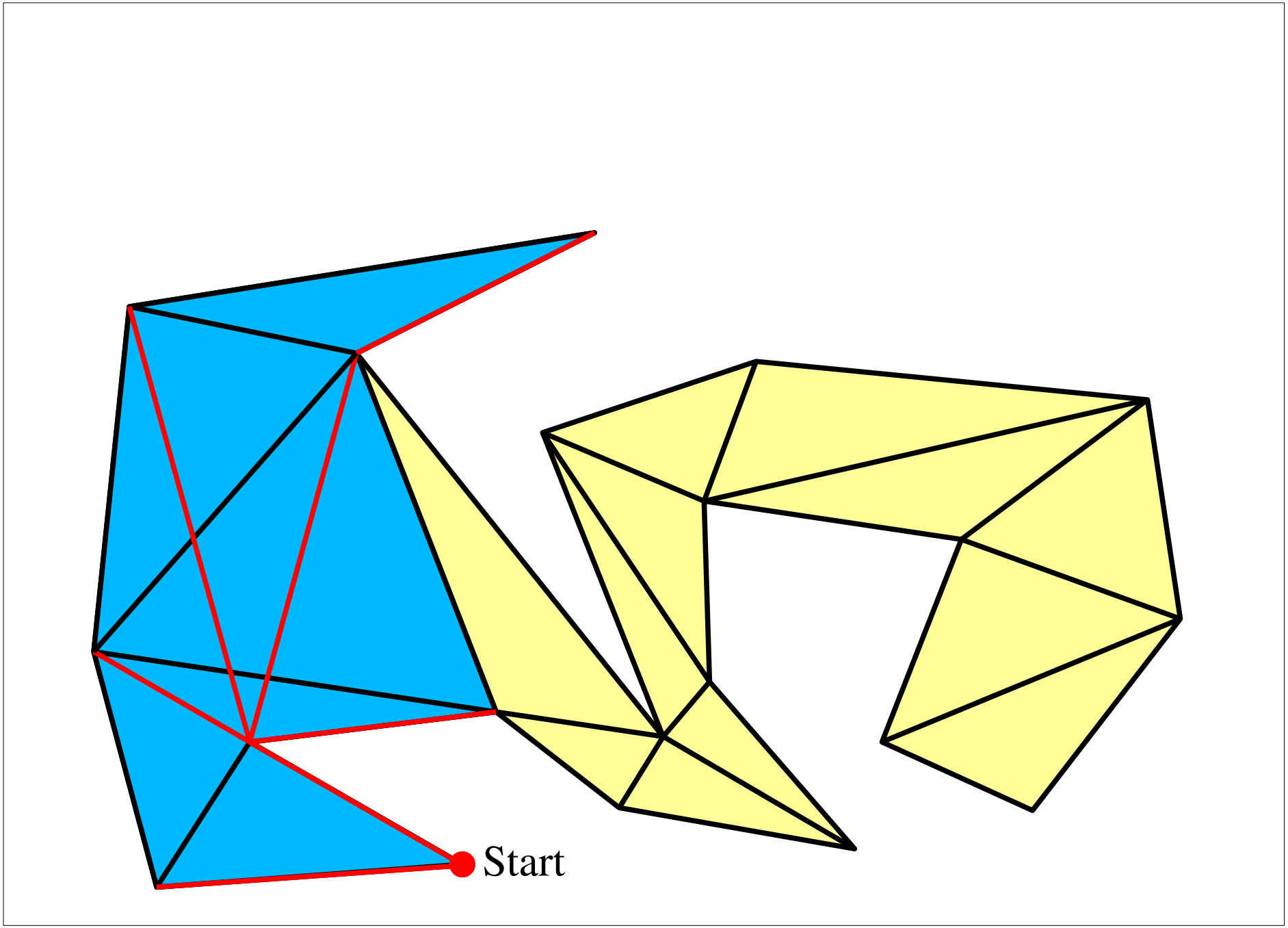

Start


Start


Start


Start


Start

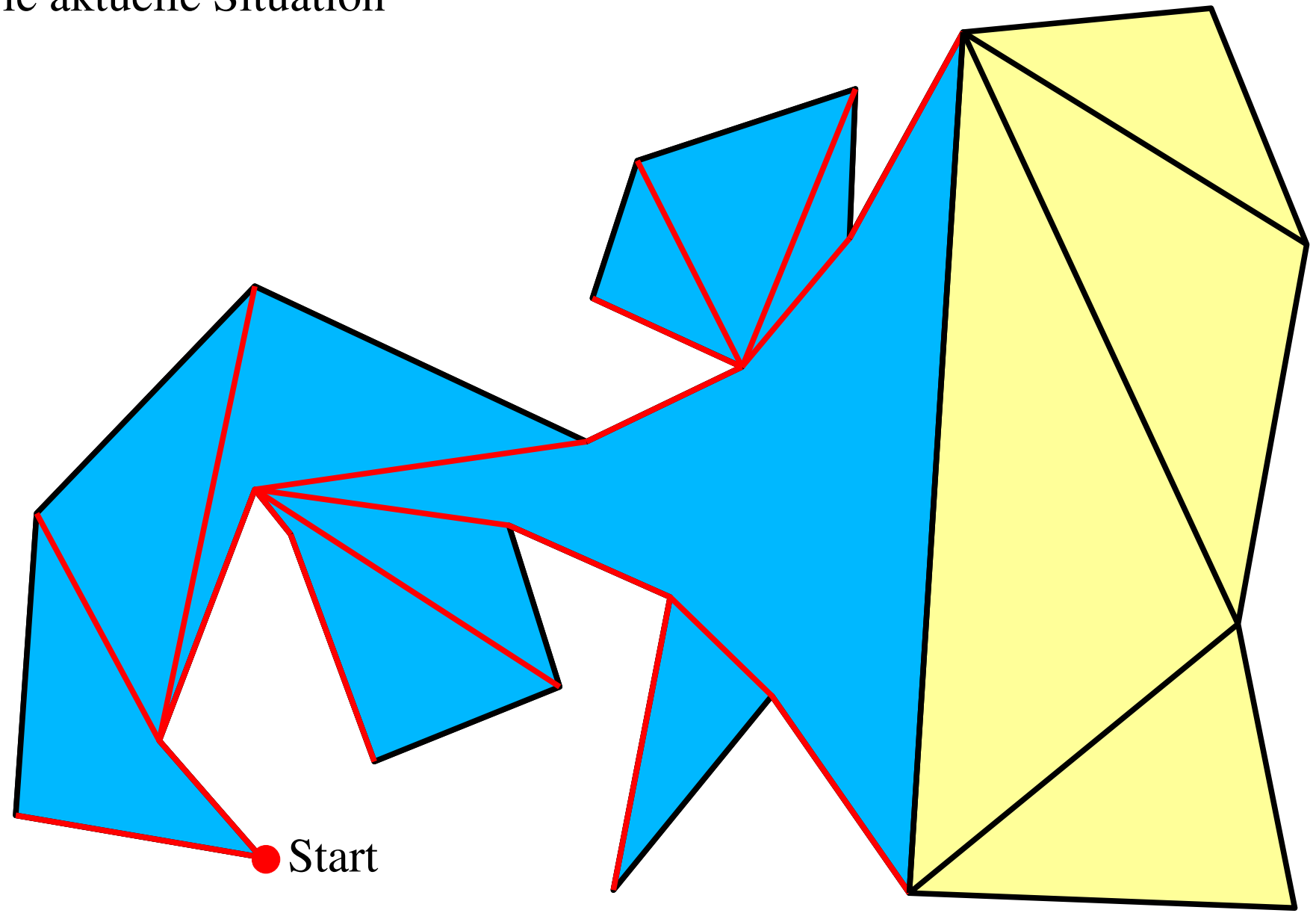

Start

Wir berechnen eigentlich sogar die kürzesten Wege vom Startpunkt **zu allen Ecken** des Polygons.

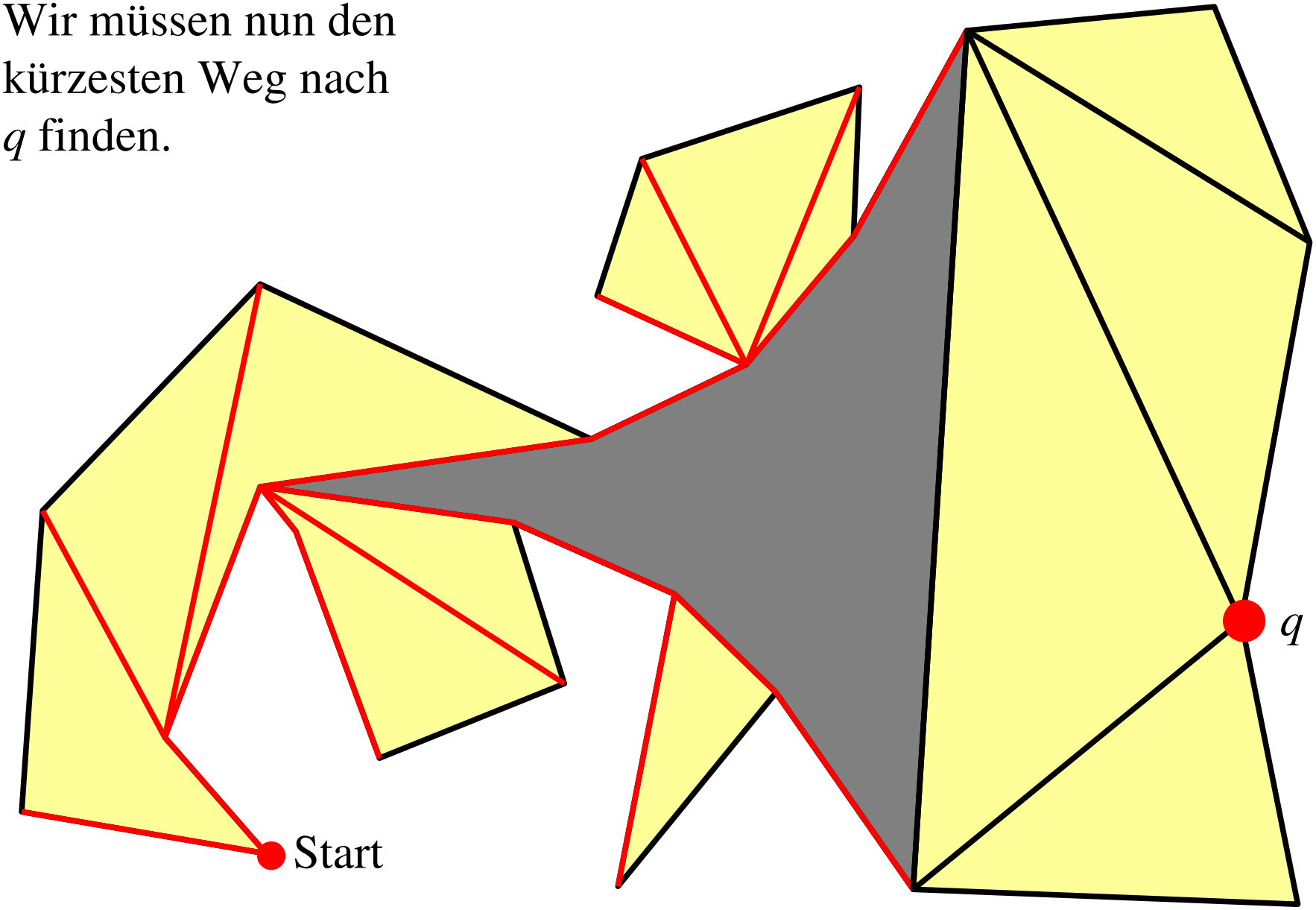
Die entstehende Struktur nennen wir den **Kürzeste-Wege-Baum** oder shortest path tree von P, kurz $spt(P)$.

Im Folgenden wird der Schritt von einem Dreieck der Triangulation zum nächsten genauer beschrieben.

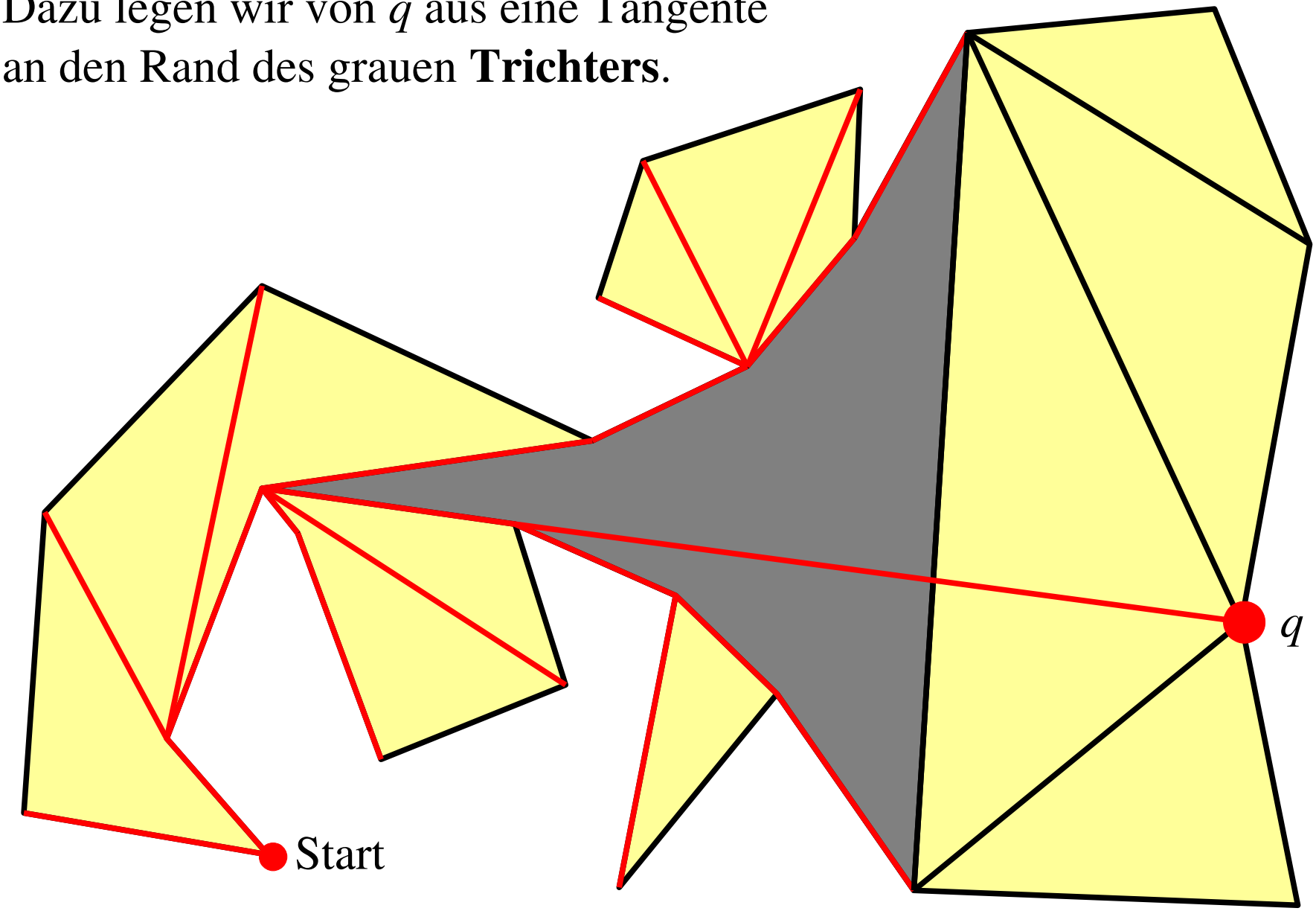
Die aktuelle Situation



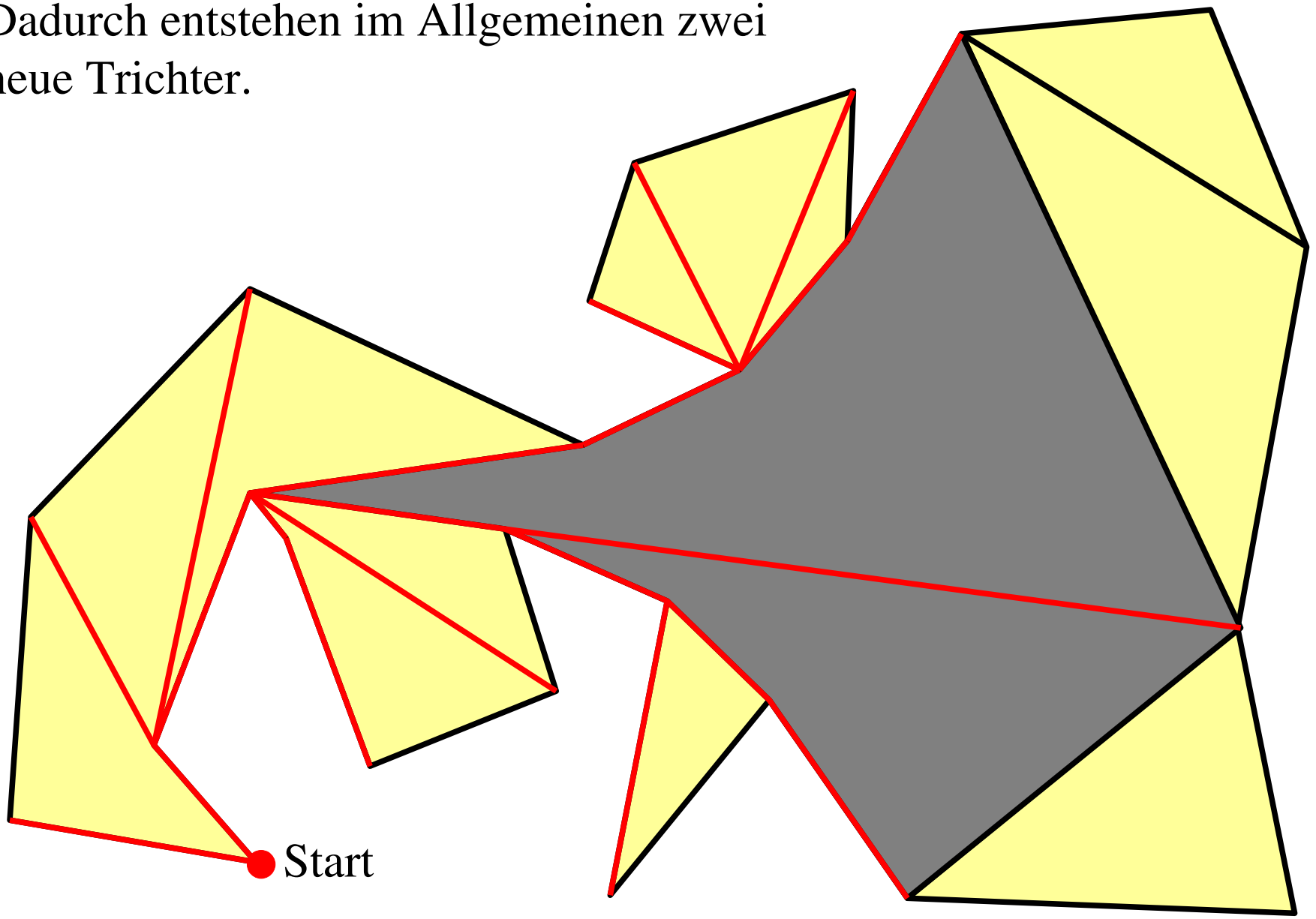
Wir müssen nun den kürzesten Weg nach q finden.



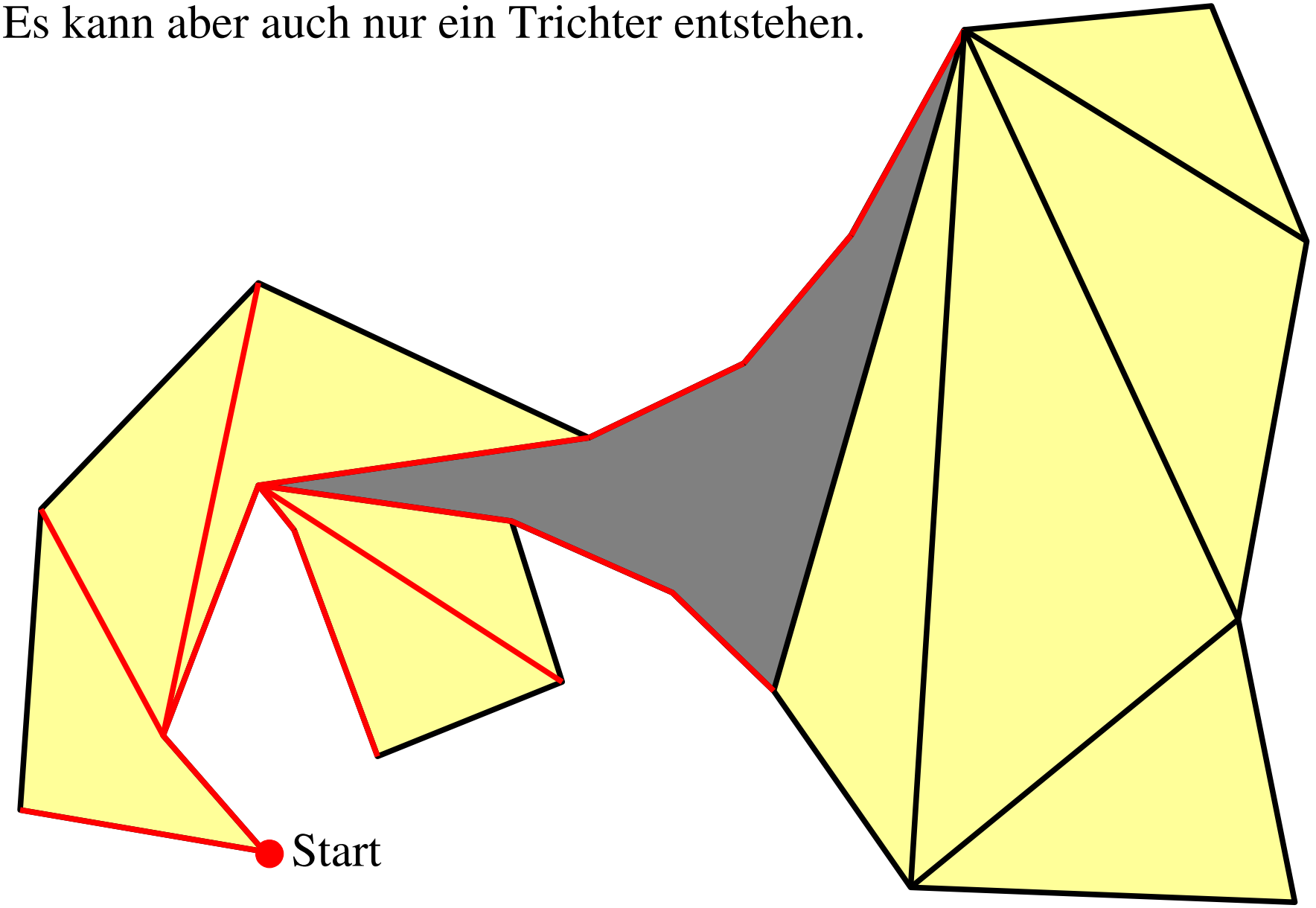
Dazu legen wir von q aus eine Tangente an den Rand des grauen **Trichters**.



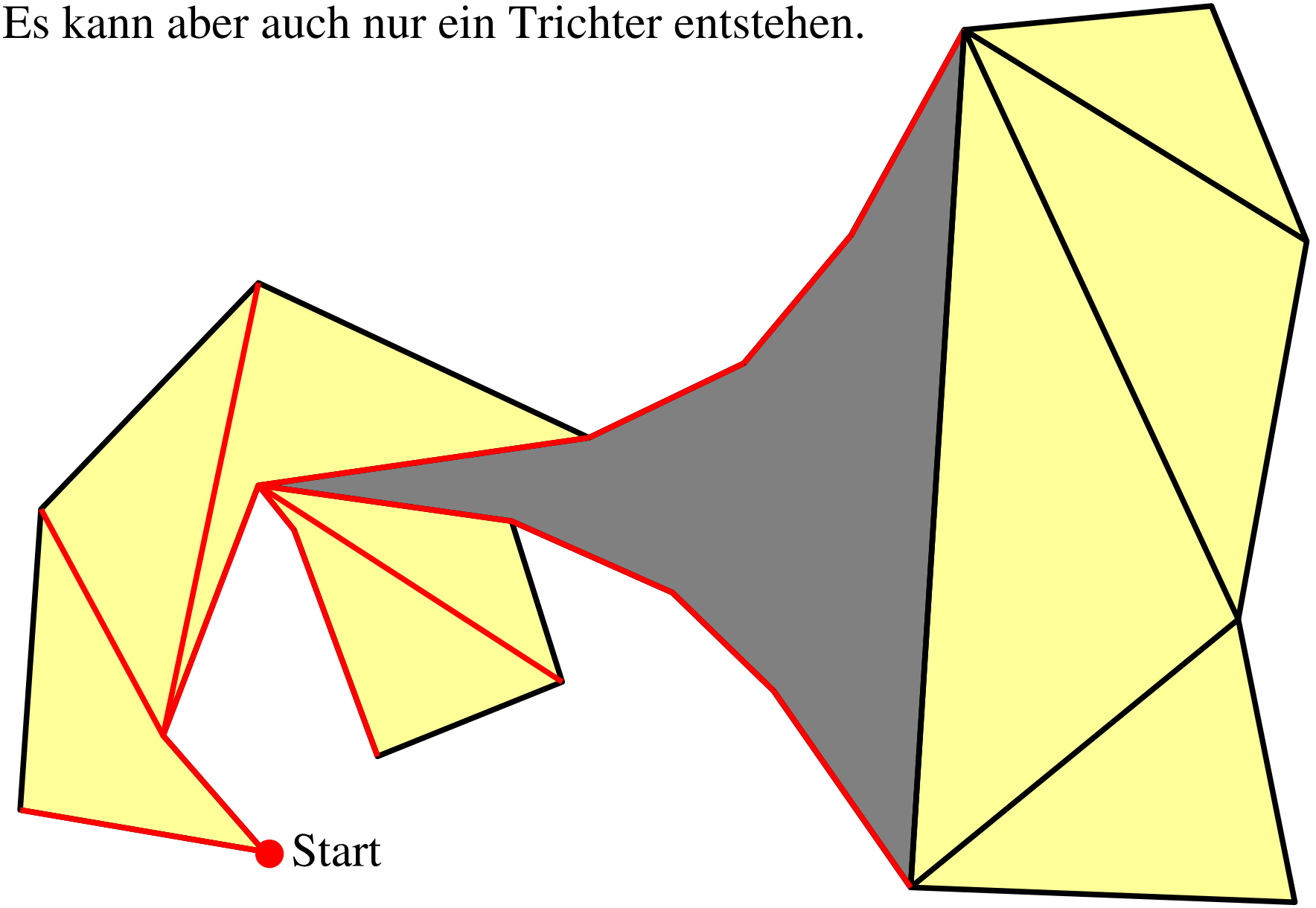
Dadurch entstehen im Allgemeinen zwei neue Trichter.



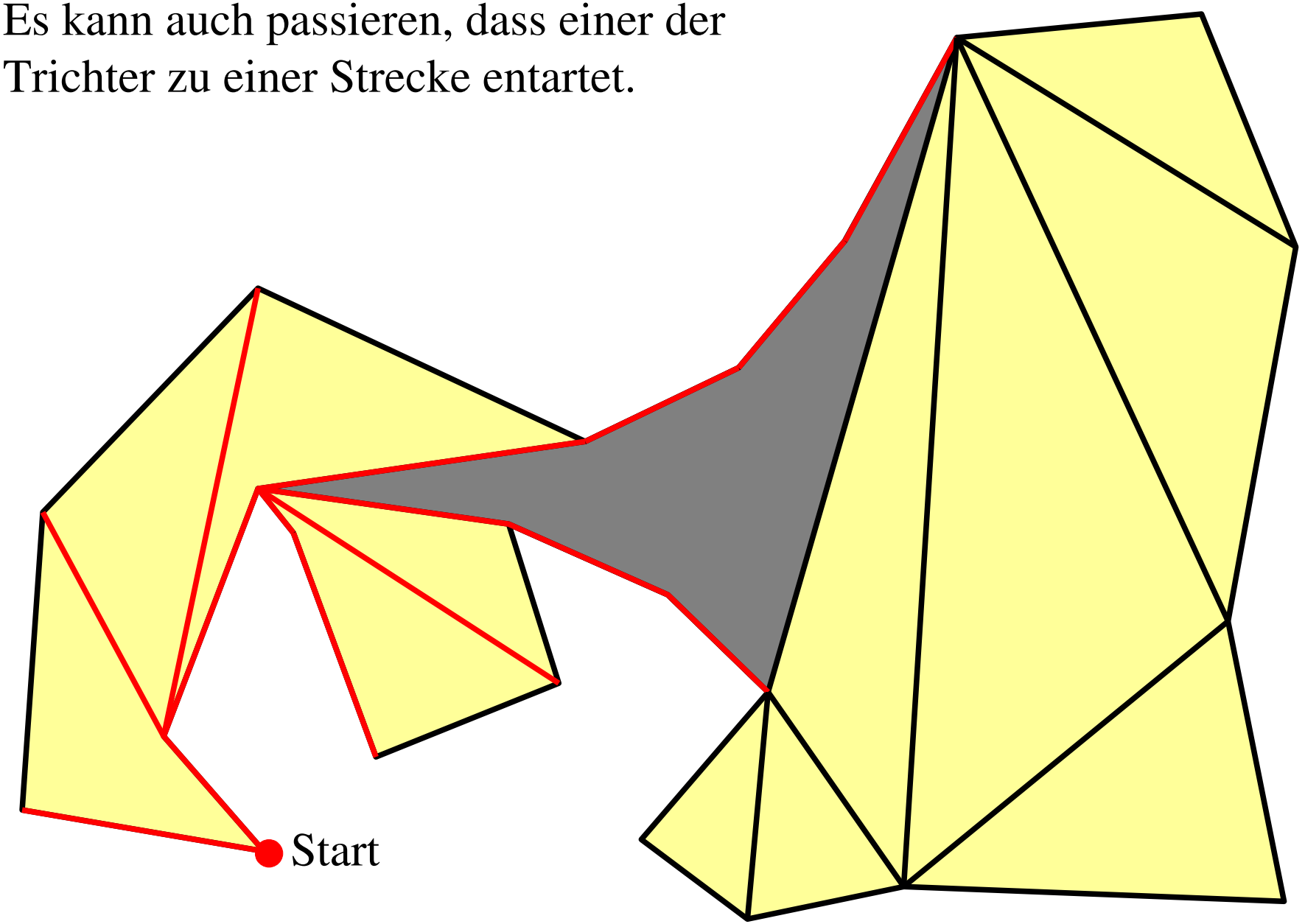
Es kann aber auch nur ein Trichter entstehen.



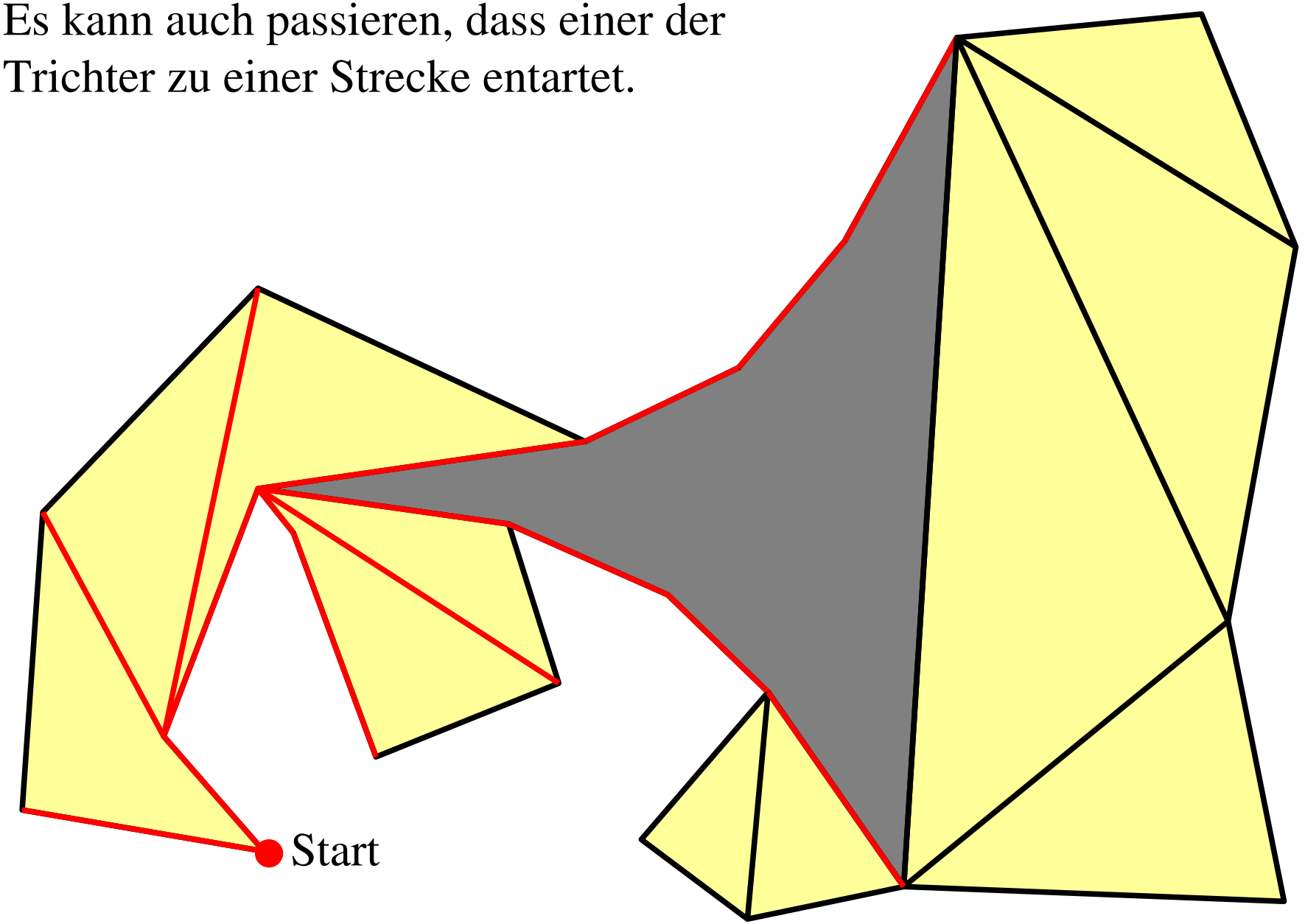
Es kann aber auch nur ein Trichter entstehen.



Es kann auch passieren, dass einer der Trichter zu einer Strecke entartet.

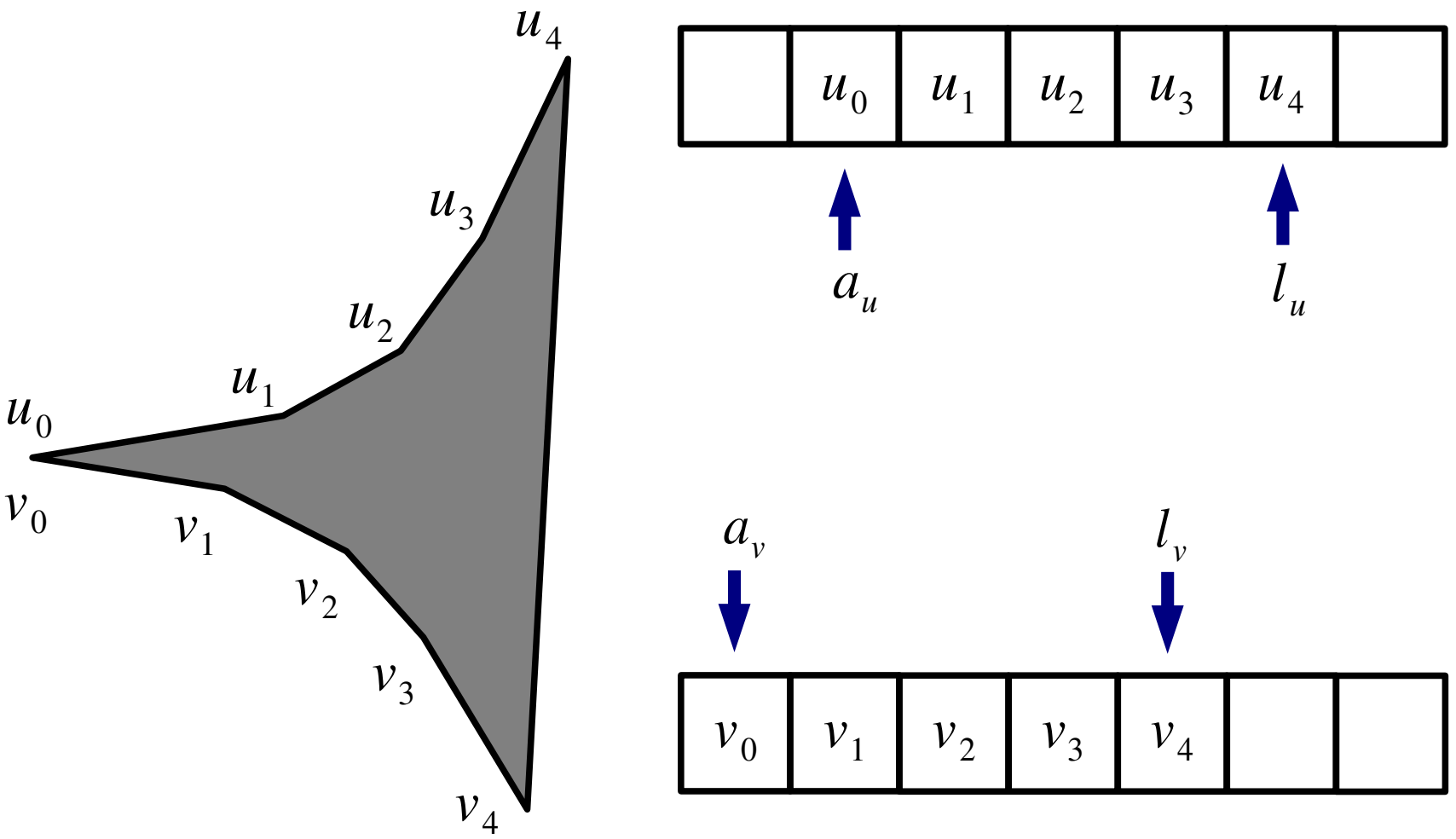


Es kann auch passieren, dass einer der Trichter zu einer Strecke entartet.

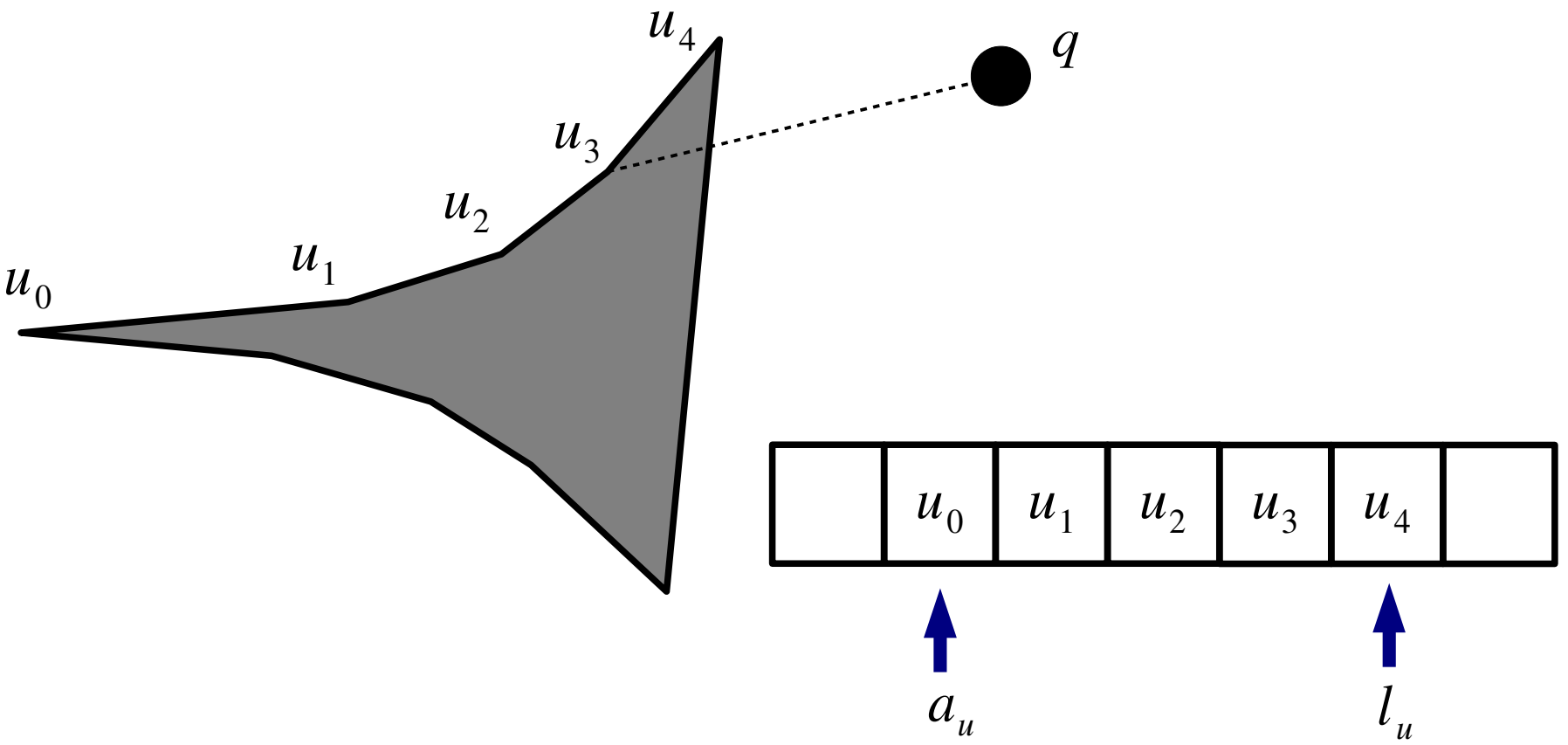


3. Verwaltung der Trichter

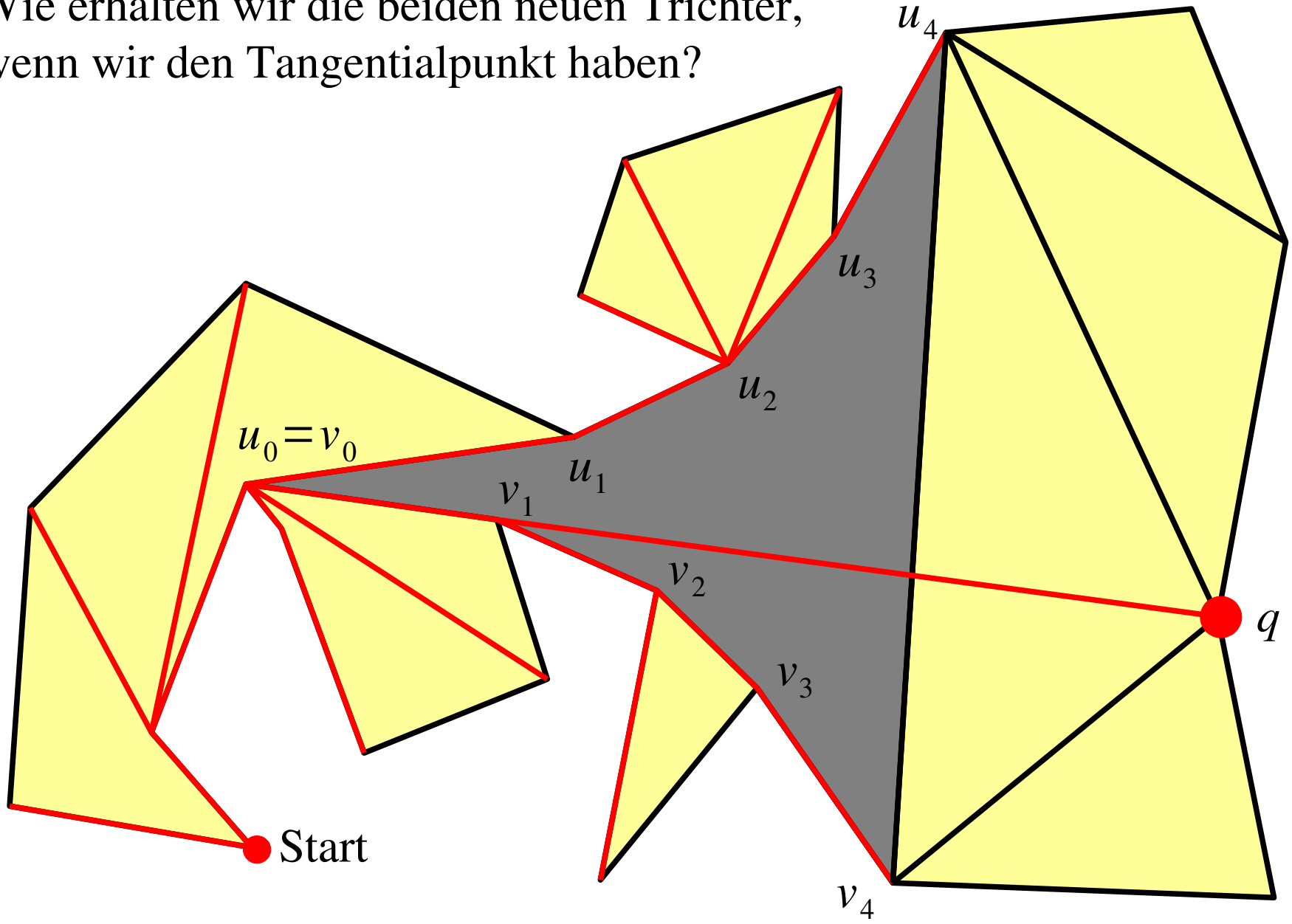
Wir legen die beiden Randketten eines Trichters jeweils in einem **dynamischen Array** ab.



Um den **Tangentialpunkt** auf einer effizient Randkette zu finden, führen wir eine **binäre Suche** durch (noch besser: positionssensitive Suche).

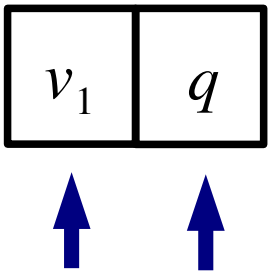


Wie erhalten wir die beiden neuen Trichter, wenn wir den Tangentialpunkt haben?

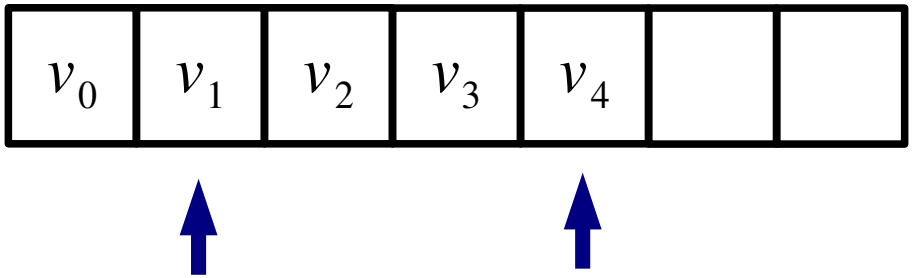


Zuerst wird der Trichter abgearbeitet, der eine **neue Spitze** hat. In unserem Beispiel also der untere Trichter.

Ein **neues Array** für die obere Kette:

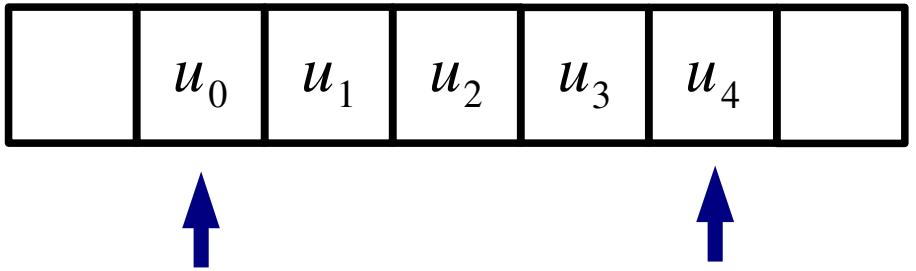


Virtuelles Abschneiden für die untere Kette:

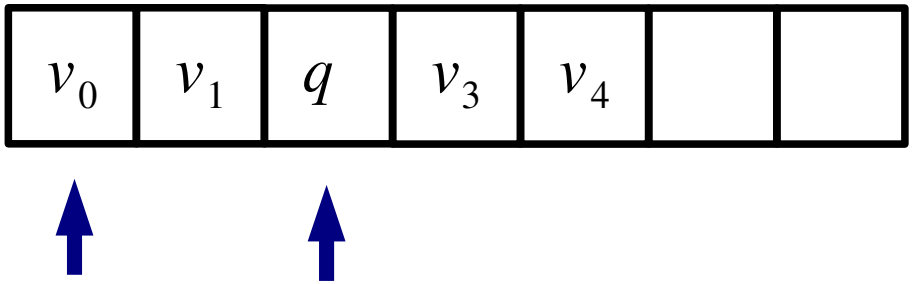


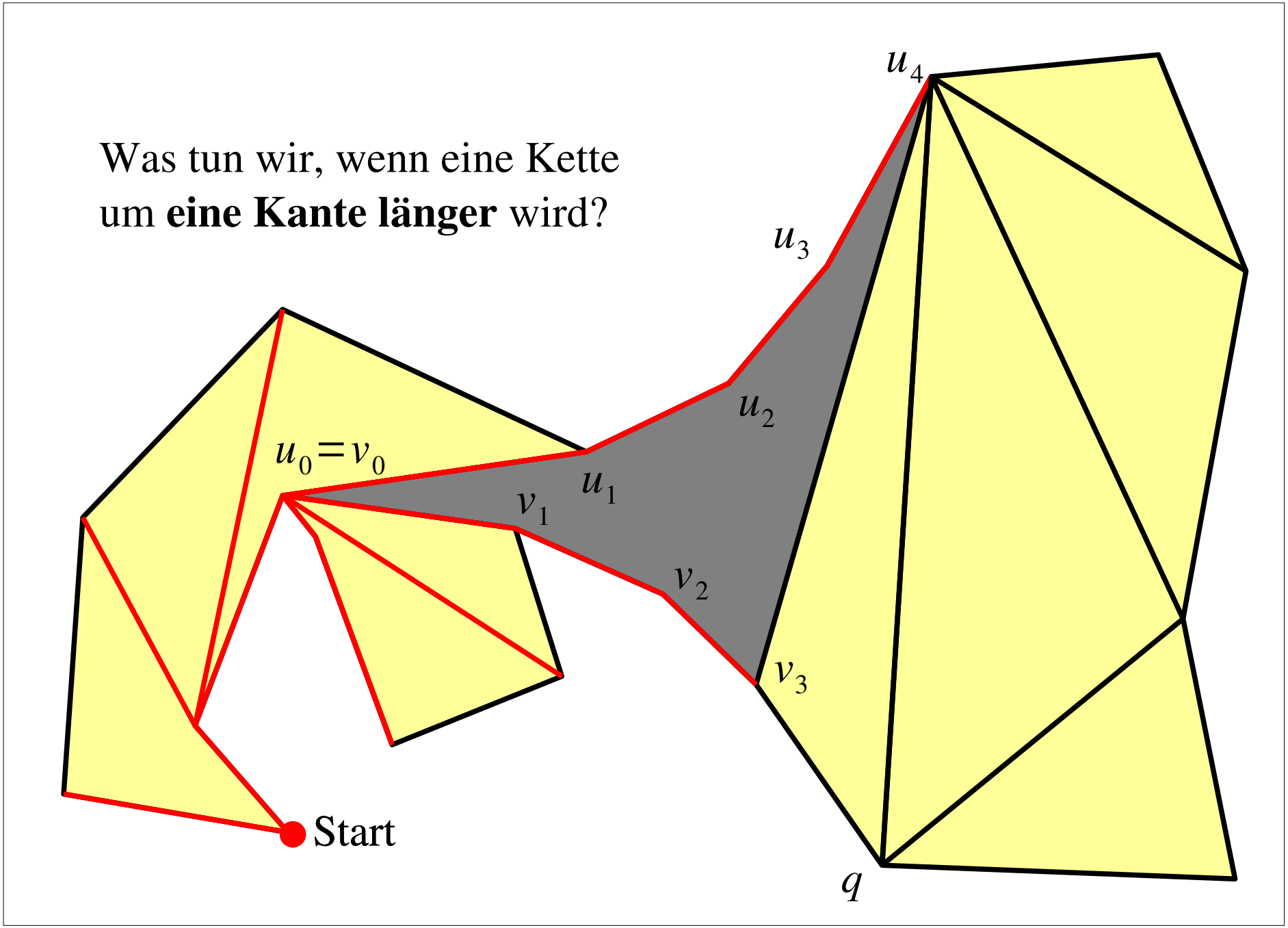
Nachdem der untere Trichter fertig abgearbeitet ist, kommt der obere Trichter dran.

Die obere Kette **bleibt** wie sie ist:

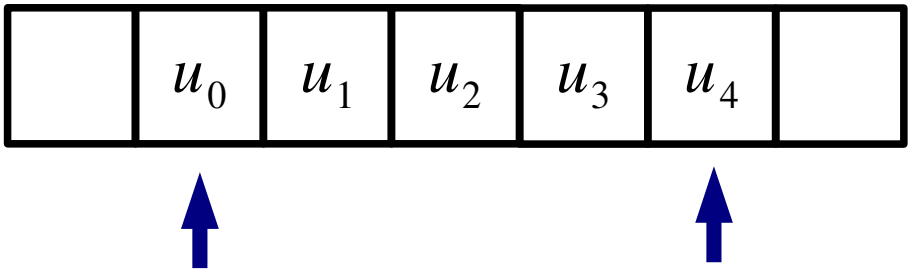


Die untere Kette wird **virtuell gekürzt** und aktualisiert:

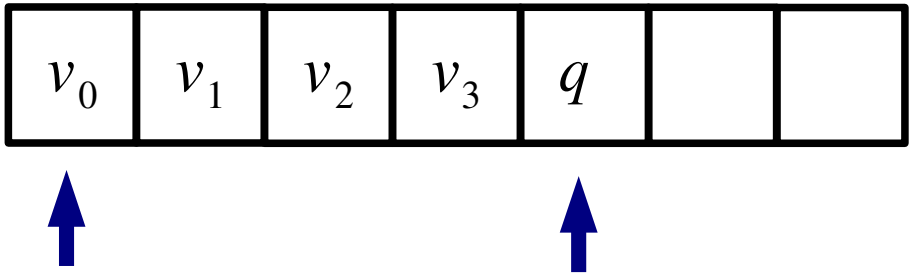



Was tun wir, wenn eine Kette
um **eine Kante länger** wird?
u_4
u_3
u_2
$u_0 = v_0$
u_1
v_1
v_2
v_3
Start
q

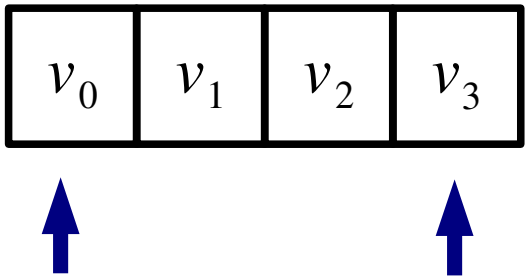
Die obere Kette bleibt wie sie ist:

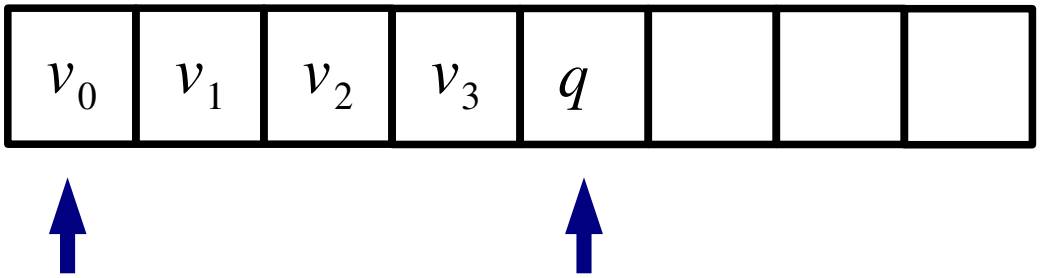


Die untere wird um eine Kante länger:



Falls im Array kein Platz mehr ist, wird alles in ein **neues Array doppelter Länge** kopiert:





Die **Laufzeit** des Algorithmus ergibt sich folgendermaßen:

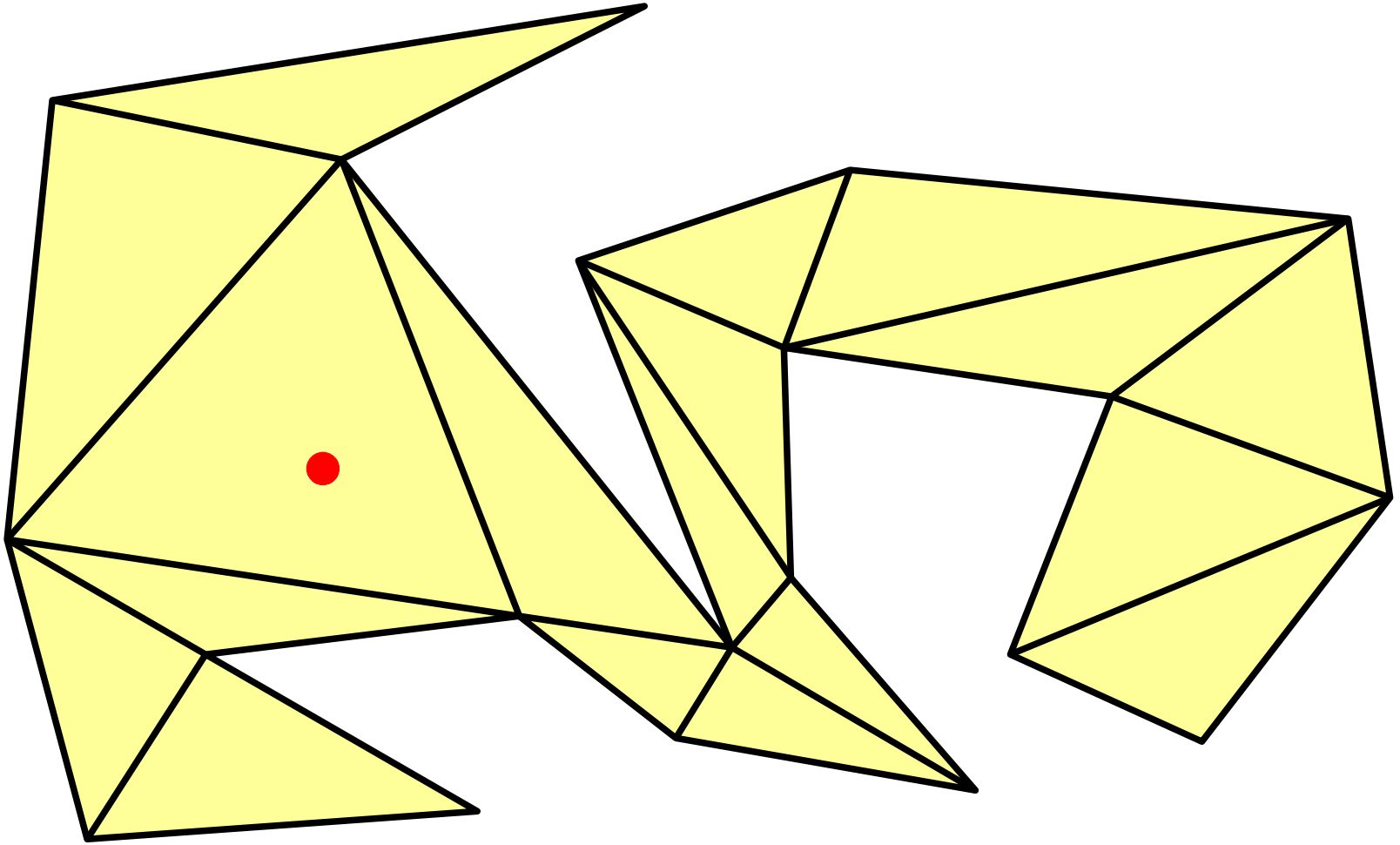
Triangulieren:	$O(n \log n)$
Finden eines Tangentialpunkts:	$O(\log n)$
Aktualisieren eines Trichters:	$O(1)$ (amortisiert)
Anzahl der Punkte q:	$O(n)$

Laufzeit insgesamt: $O(n \log n)$

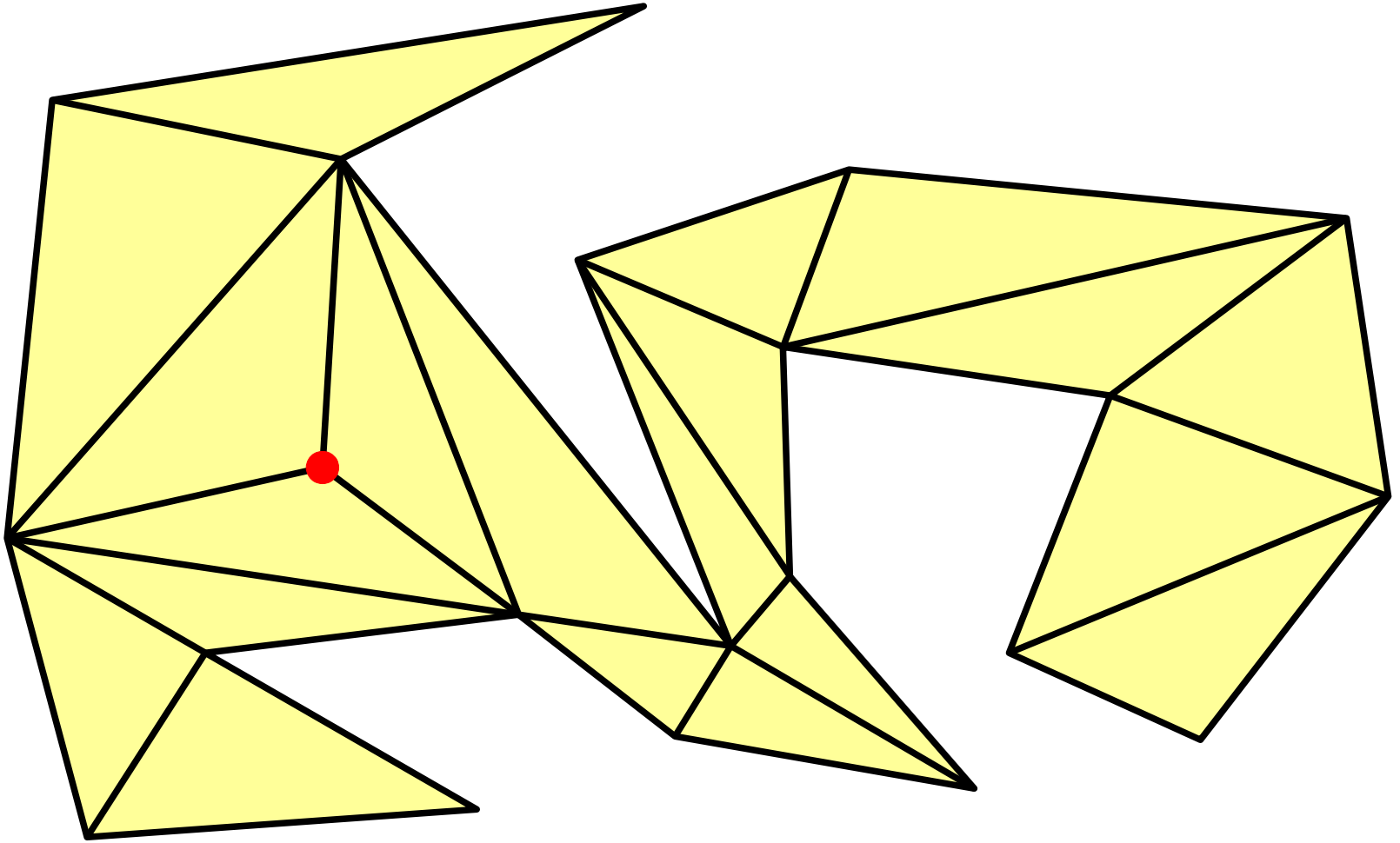
Eine detailliertere Analyse liefert sogar eine Laufzeit in $O(n)$.

4. Start-/Zielpunkte, die keine Ecken sind

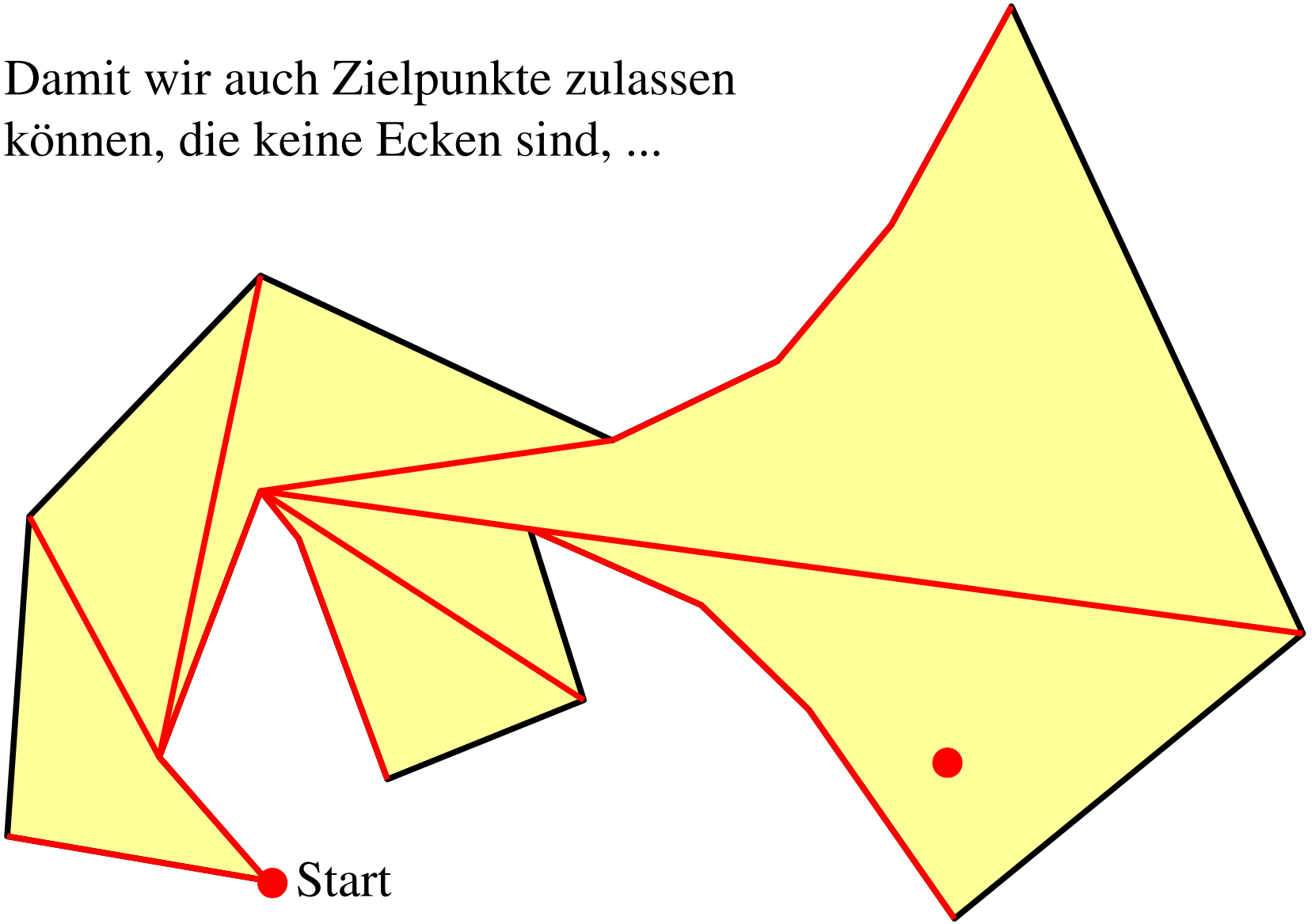
Wenn der Startpunkt kein Eckpunkt ist...



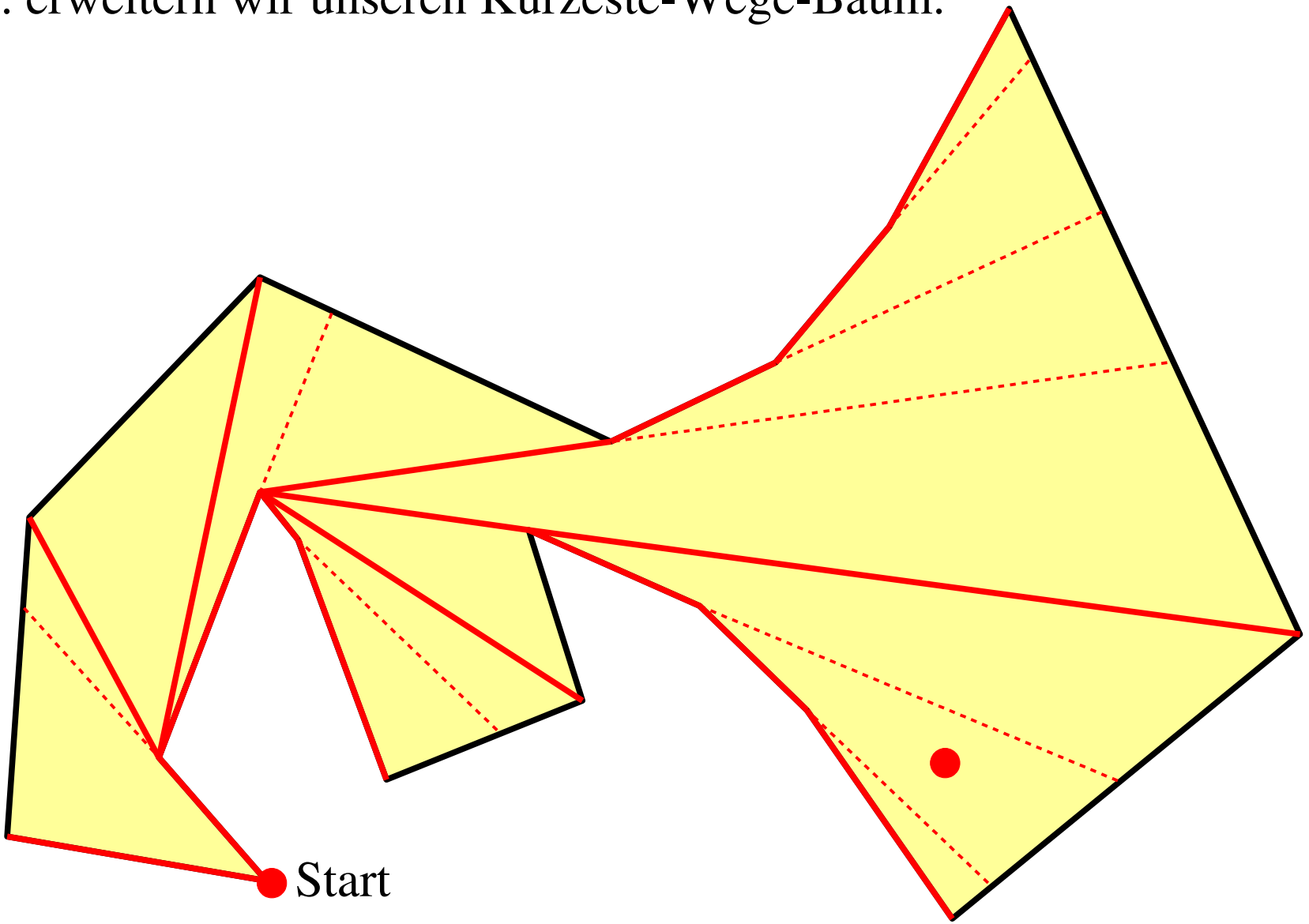
...dann lokalisieren wir das Dreieck, in dem der Startpunkt liegt, und zerlegen es in drei Dreiecke.



Damit wir auch Zielpunkte zulassen können, die keine Ecken sind, ...



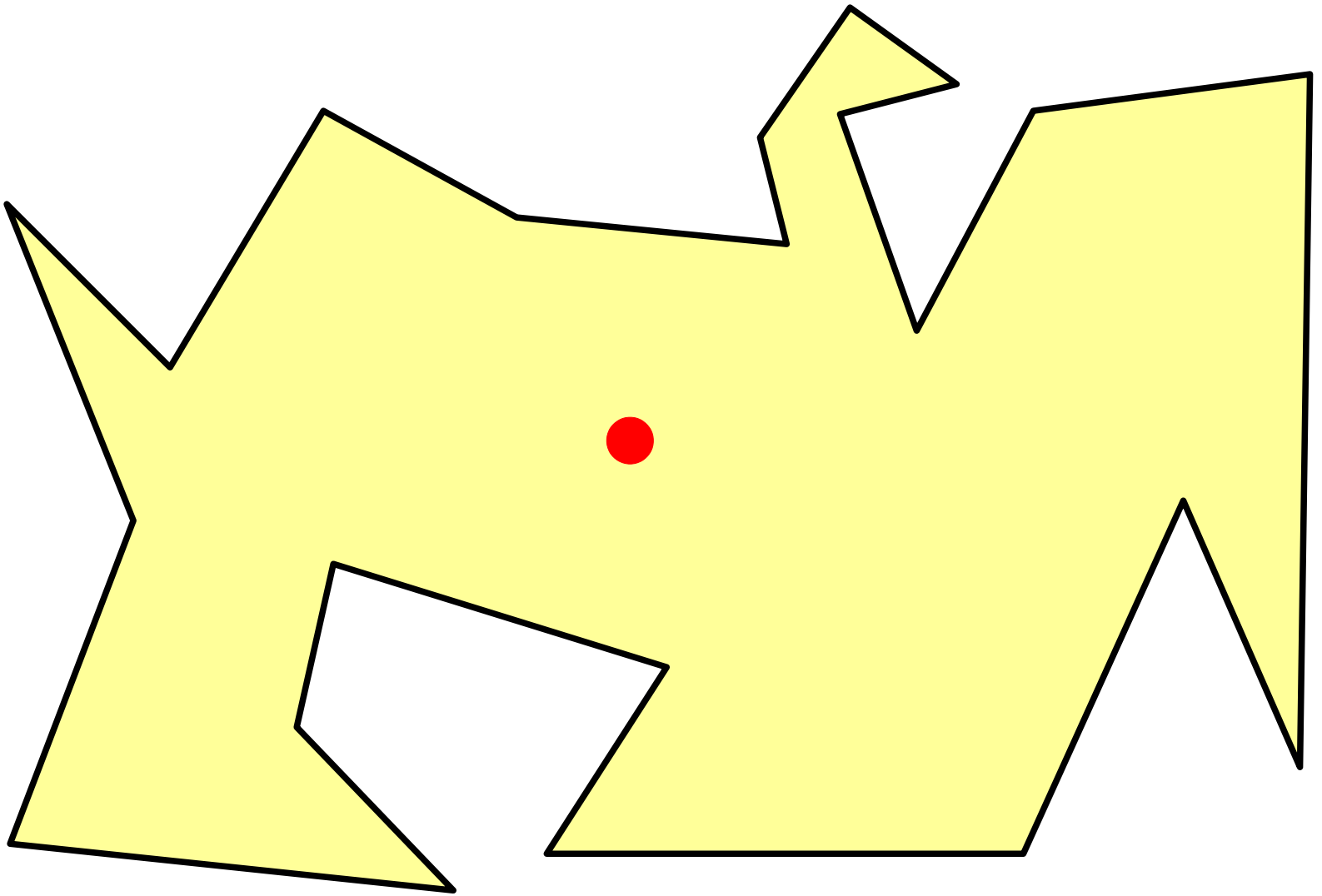
... erweitern wir unseren Kürzeste-Wege-Baum.



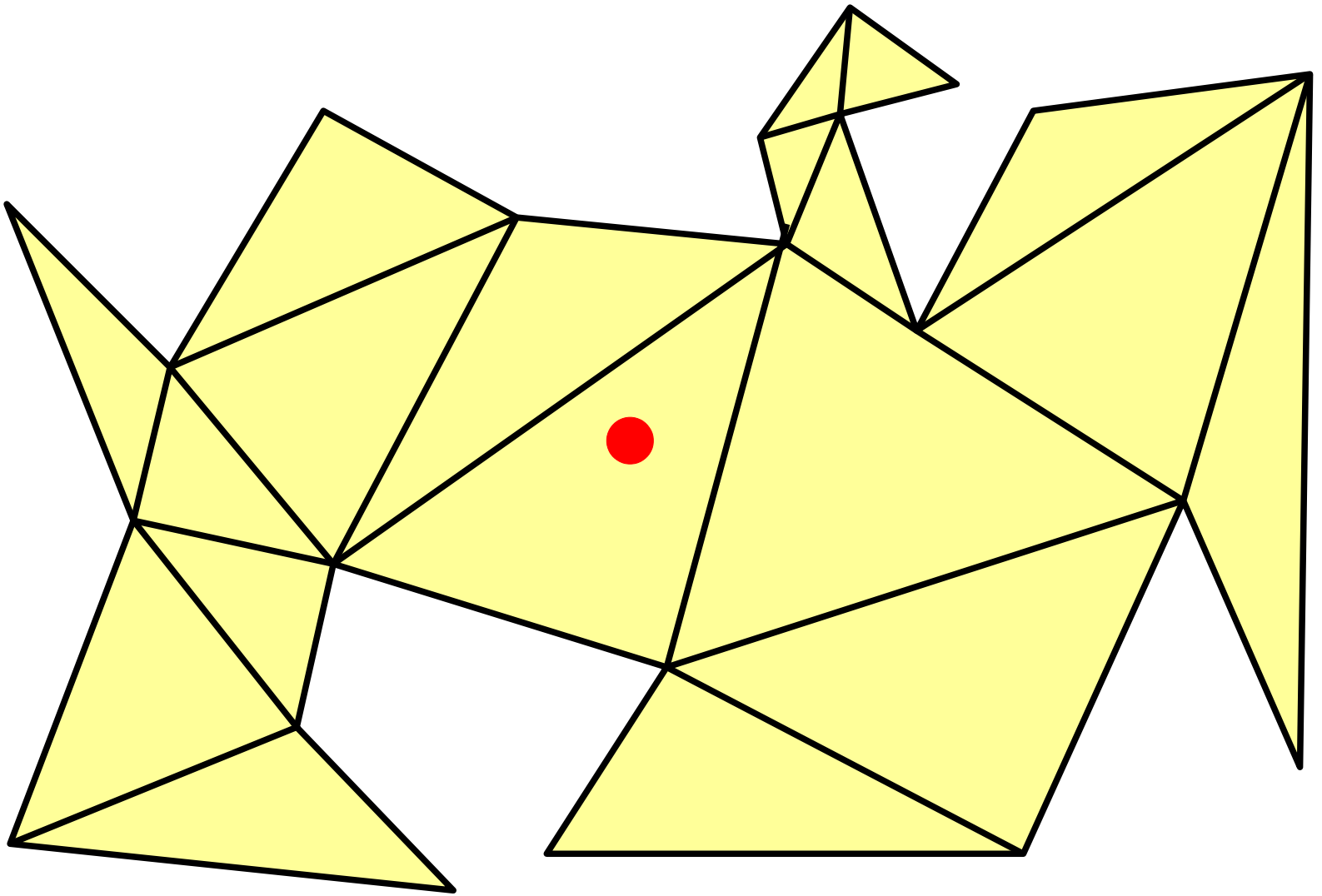
Man kann den Kürzeste-Wege-Baum zu einer **Datenstruktur** ausbauen, mit deren Hilfe man für beliebige vom Nutzer vorgegebene Zielpunkte effizient den kürzesten Weg erhält.

5. Anwendungen des Kürzeste-Wege-Baums

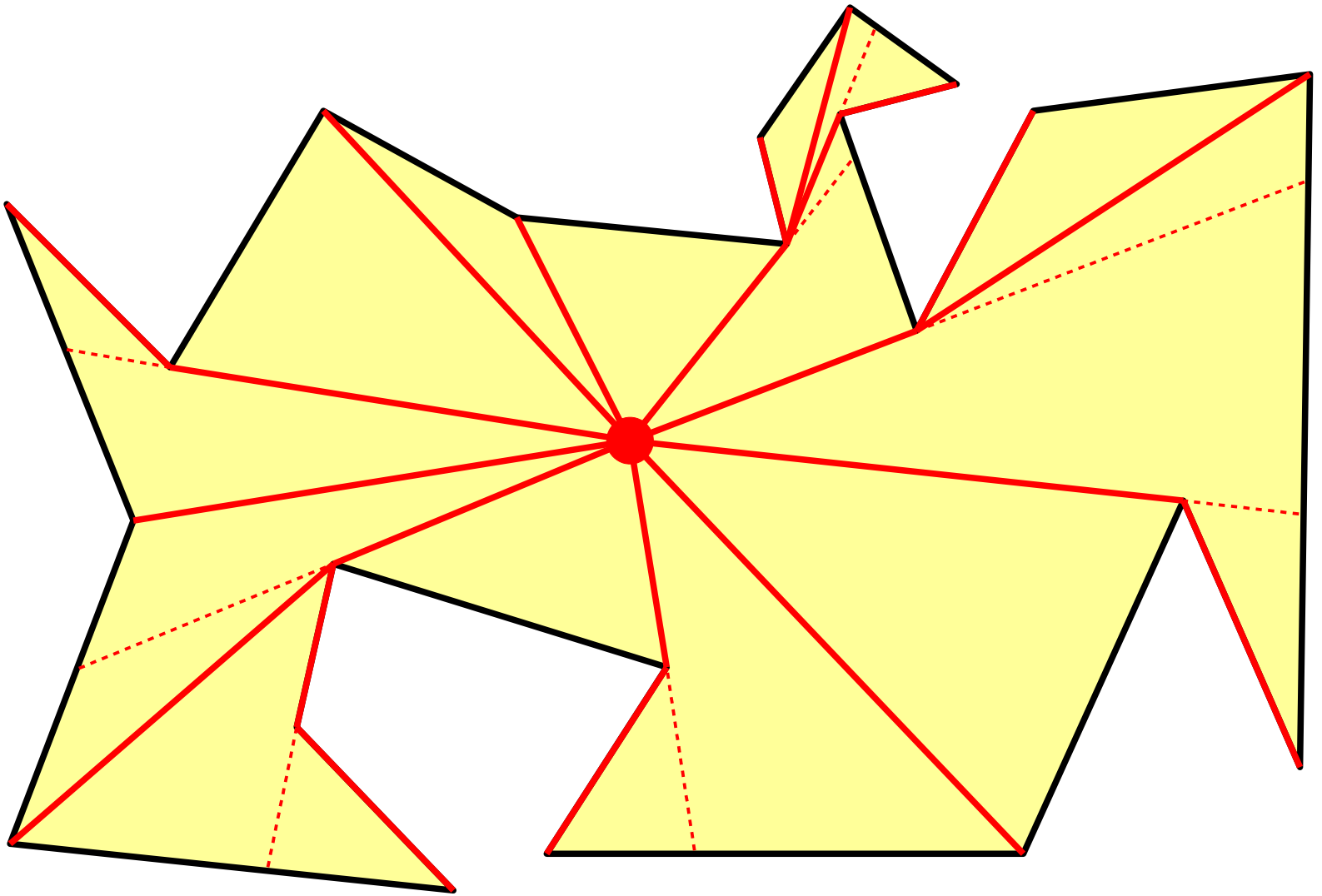
Sichtbarkeit von einem Punkt



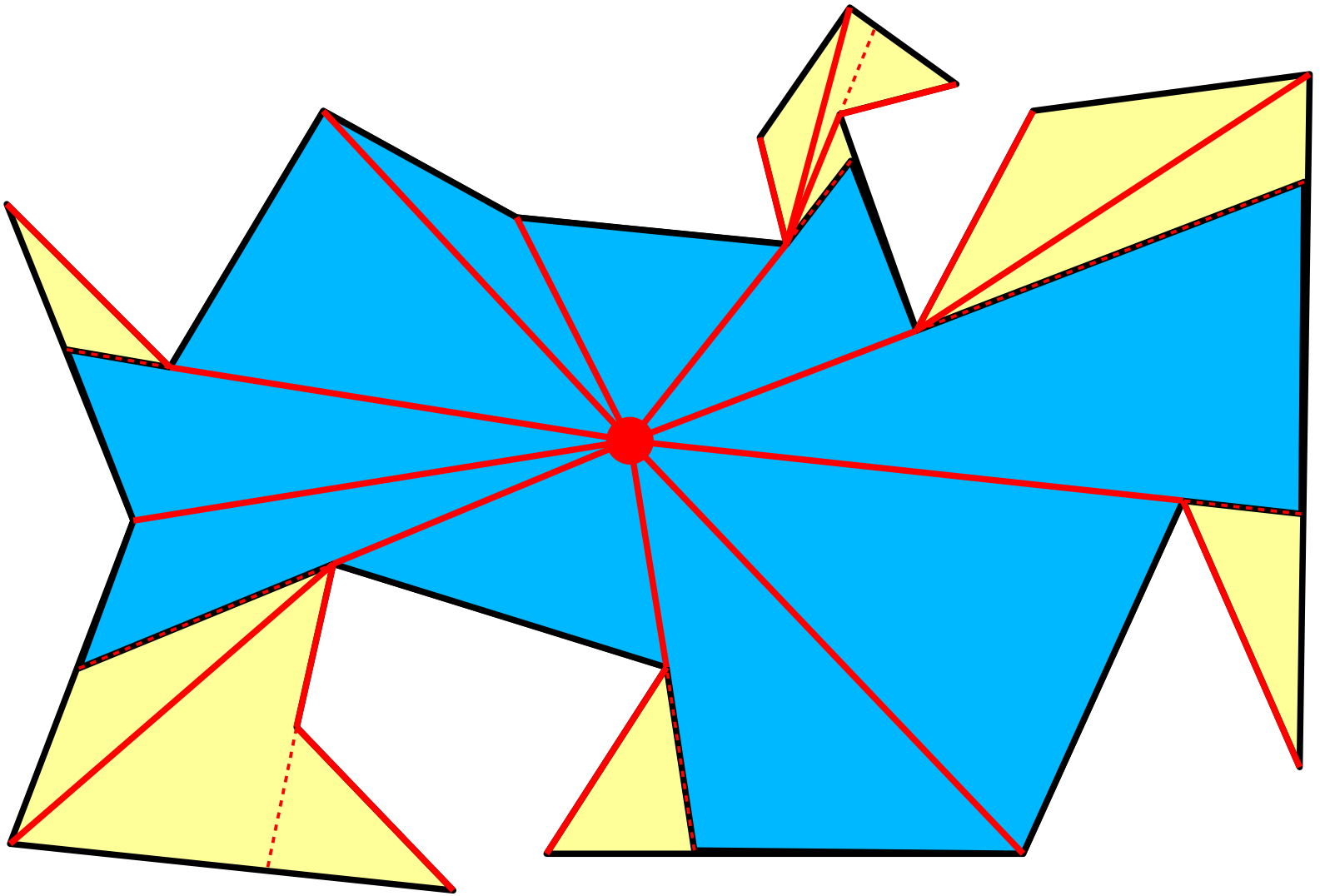
Als Erstes wird trianguliert.



Dann wird der erweiterte Kürzeste-Wege-Baum berechnet.

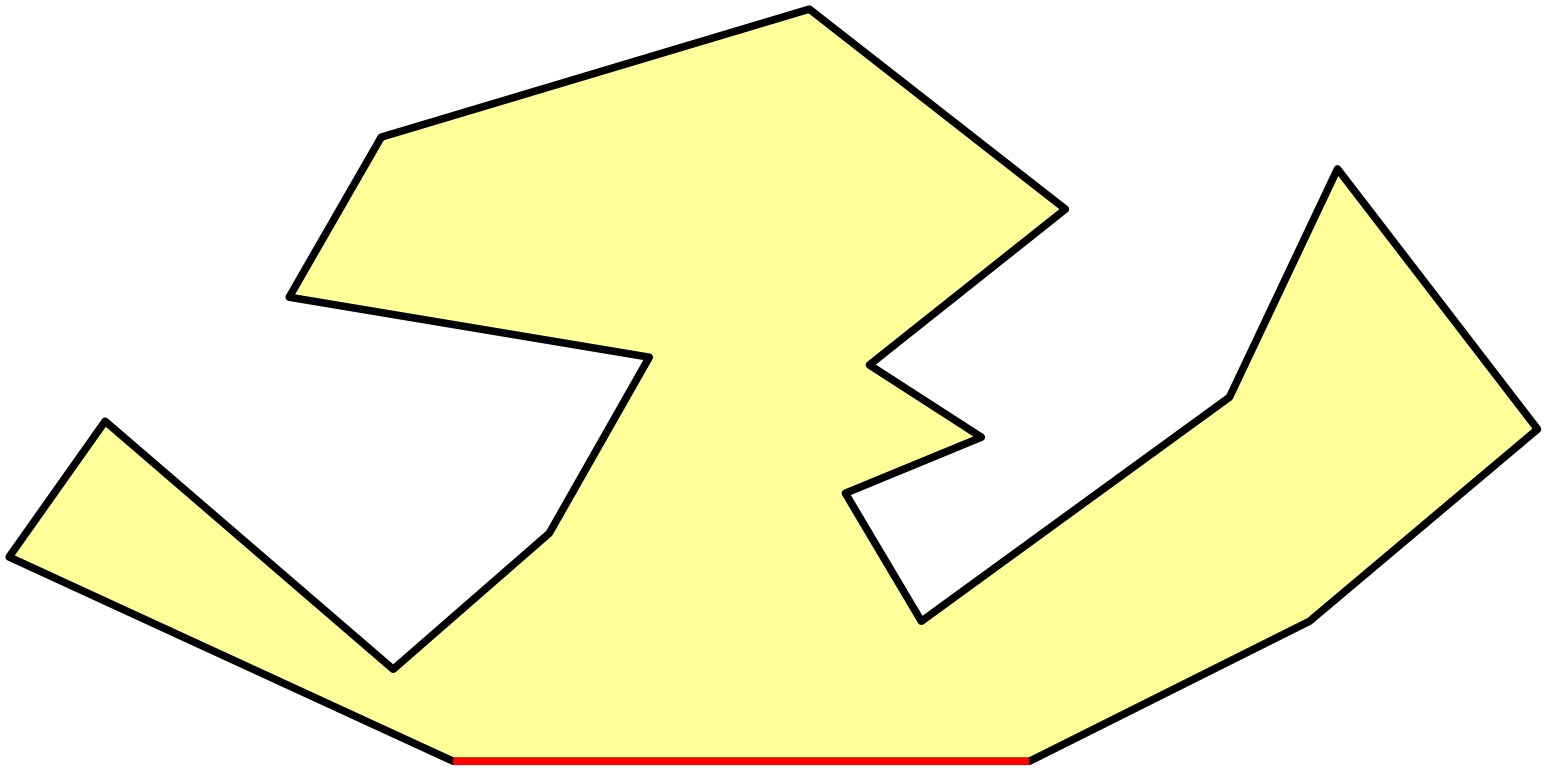


Damit kann man das Sichtbarkeitspolygon in $O(n)$ auslesen.



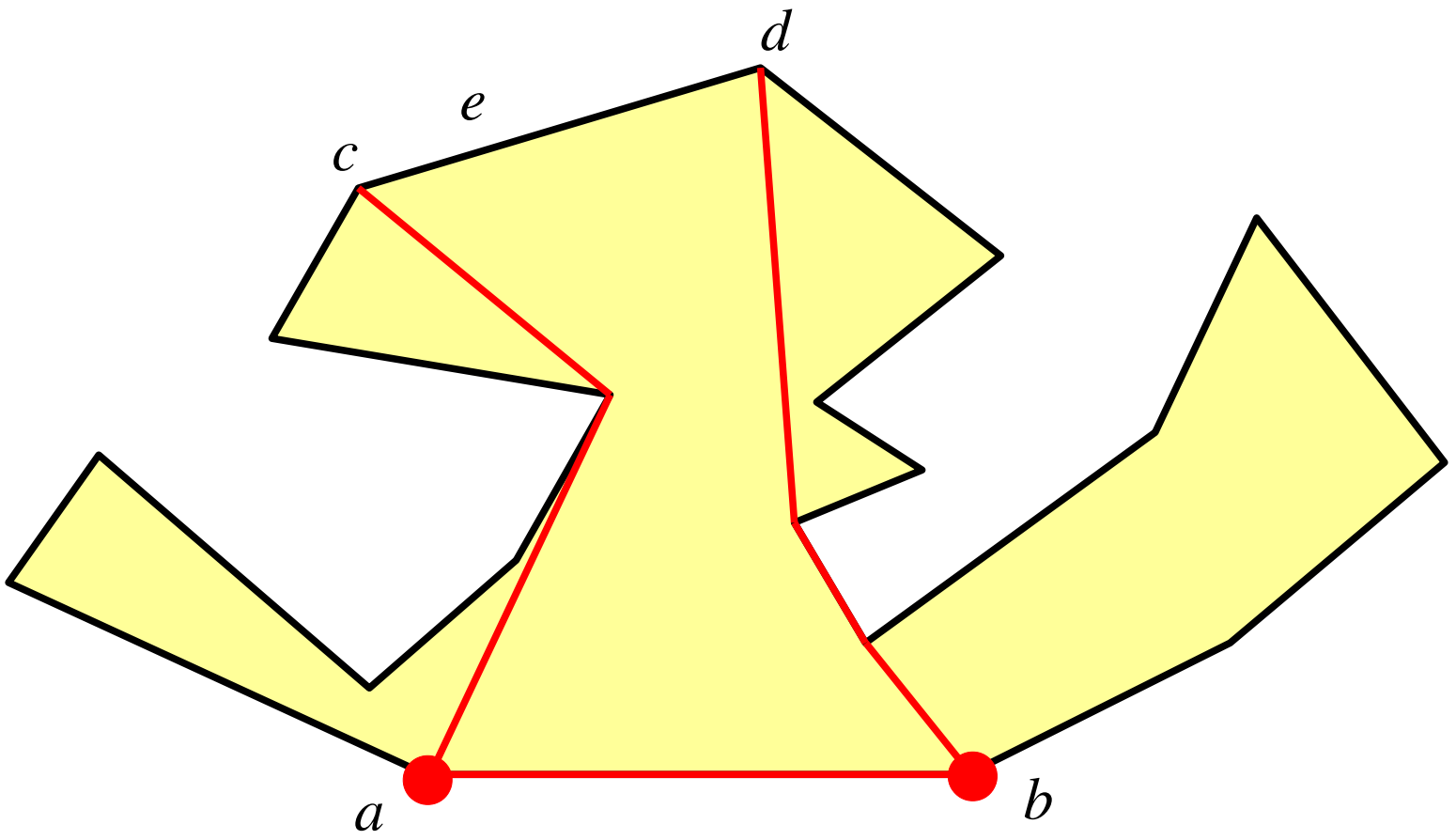
Sichtbarkeit von einer Strecke

Es soll der Bereich eines einfachen Polygons berechnet werden, der von einem Gerät überwacht wird, wenn es sich entlang einer Kante **bewegen kann**.



Sei *e* eine Kante, die zumindest **teilweise überwacht** wird.

Wir betrachten die kürzesten Wege von *a* nach *c* und von *b* nach *d*.



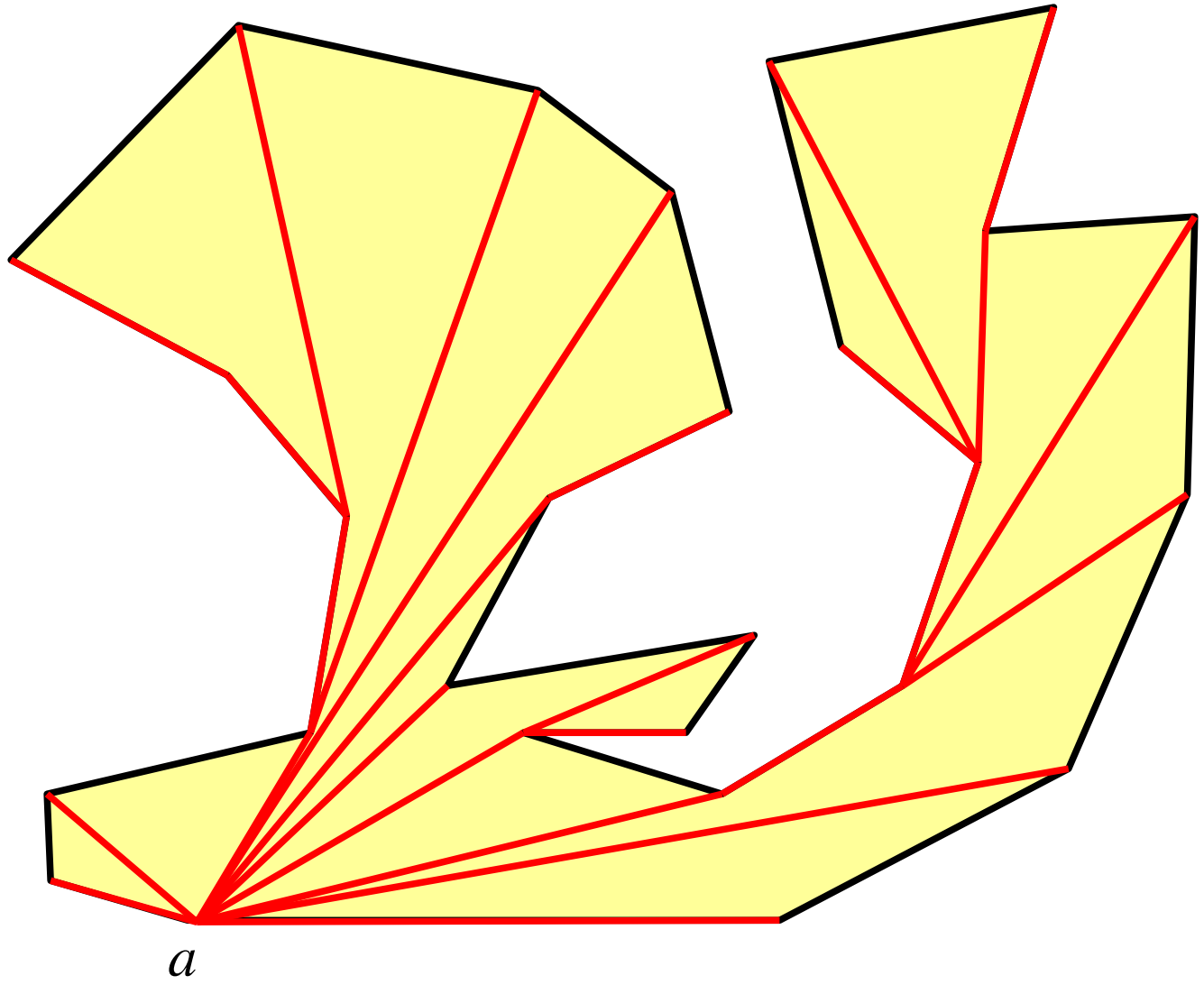
Wenn ein Punkt von *e* sichtbar ist, dann sind der kürzeste Weg von *a* nach *c* und der kürzeste Weg von *b* nach *d* entsprechend verlaufende **reflexe Ketten**.

Wichtig für uns ist die **Umkehrung**:

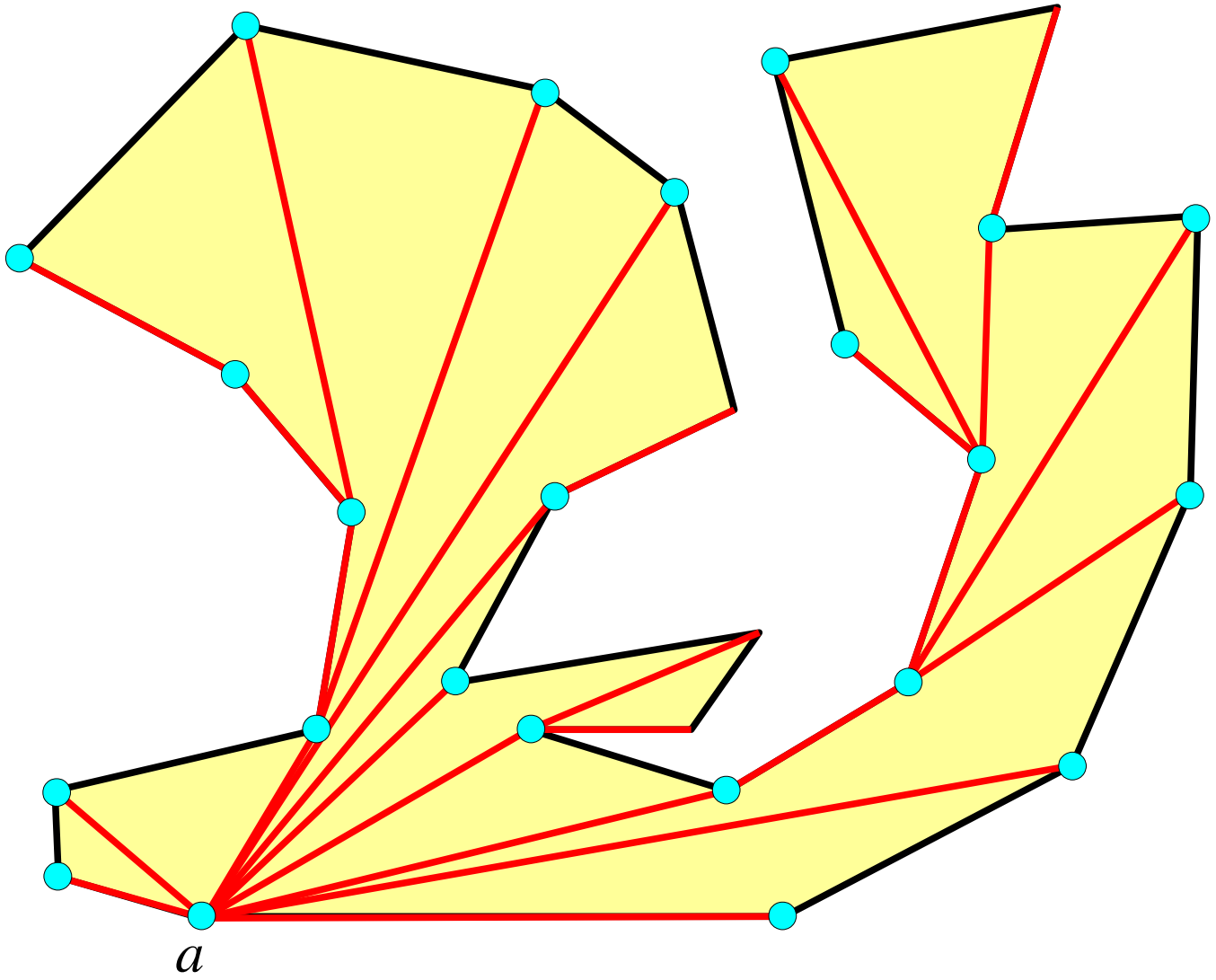
Wenn der kürzeste Weg von *a* nach *c* oder der kürzeste Weg von *b* nach *d* keine entsprechend verlaufende reflexe Kette ist, dann ist kein Punkt auf *e* sichtbar.

Es liegt also nahe, die beiden Kürzeste-Wege-Bäume von *a* und *b* aus zu betrachten und dann die Kanten des Polygons durchzugehen.

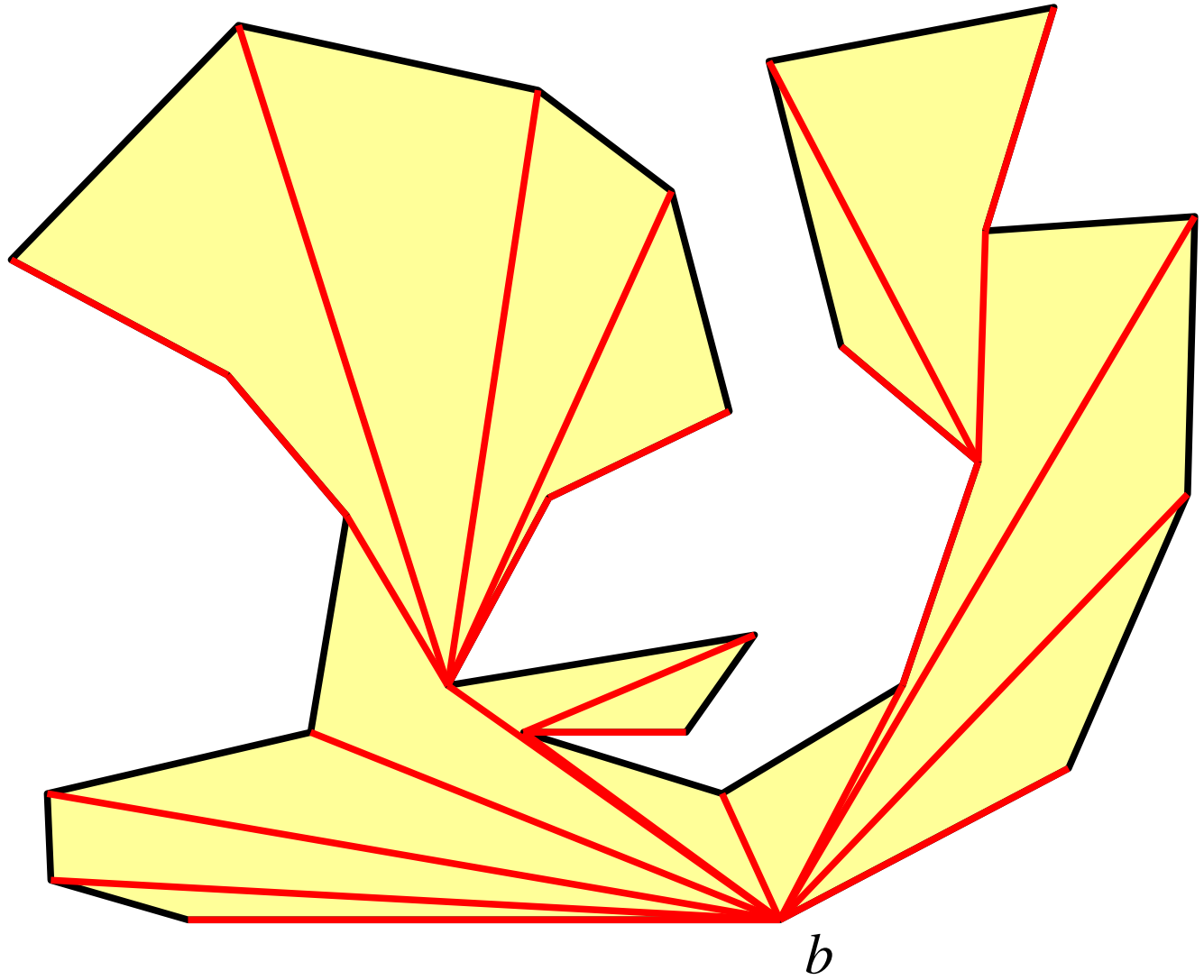
Der Kürzeste-Wege-Baum von a.



Wir markieren die Knoten, die man von a über entsprechend verlaufende reflexe Ketten erreicht.

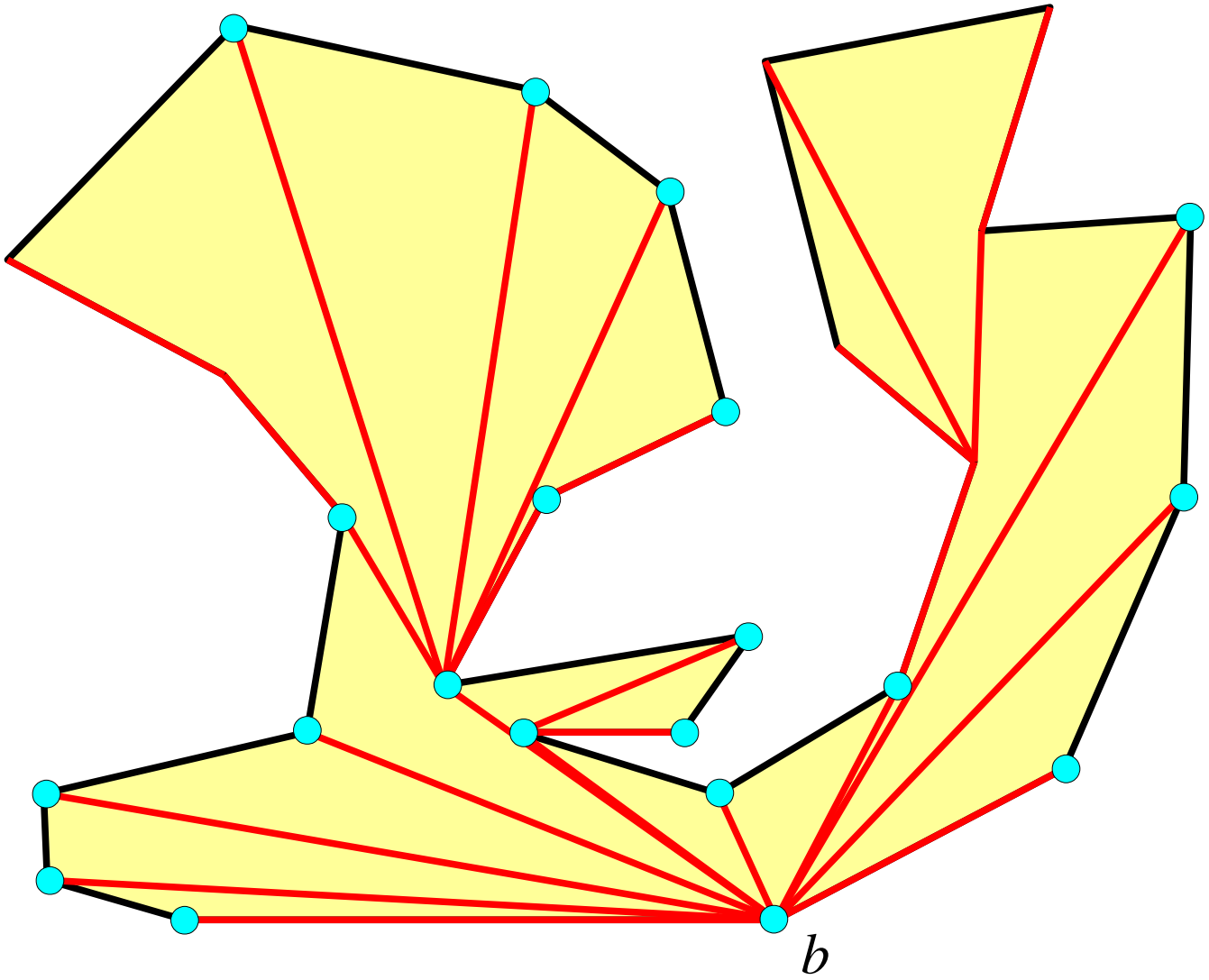


Der Kürzeste-Wege-Baum von b.

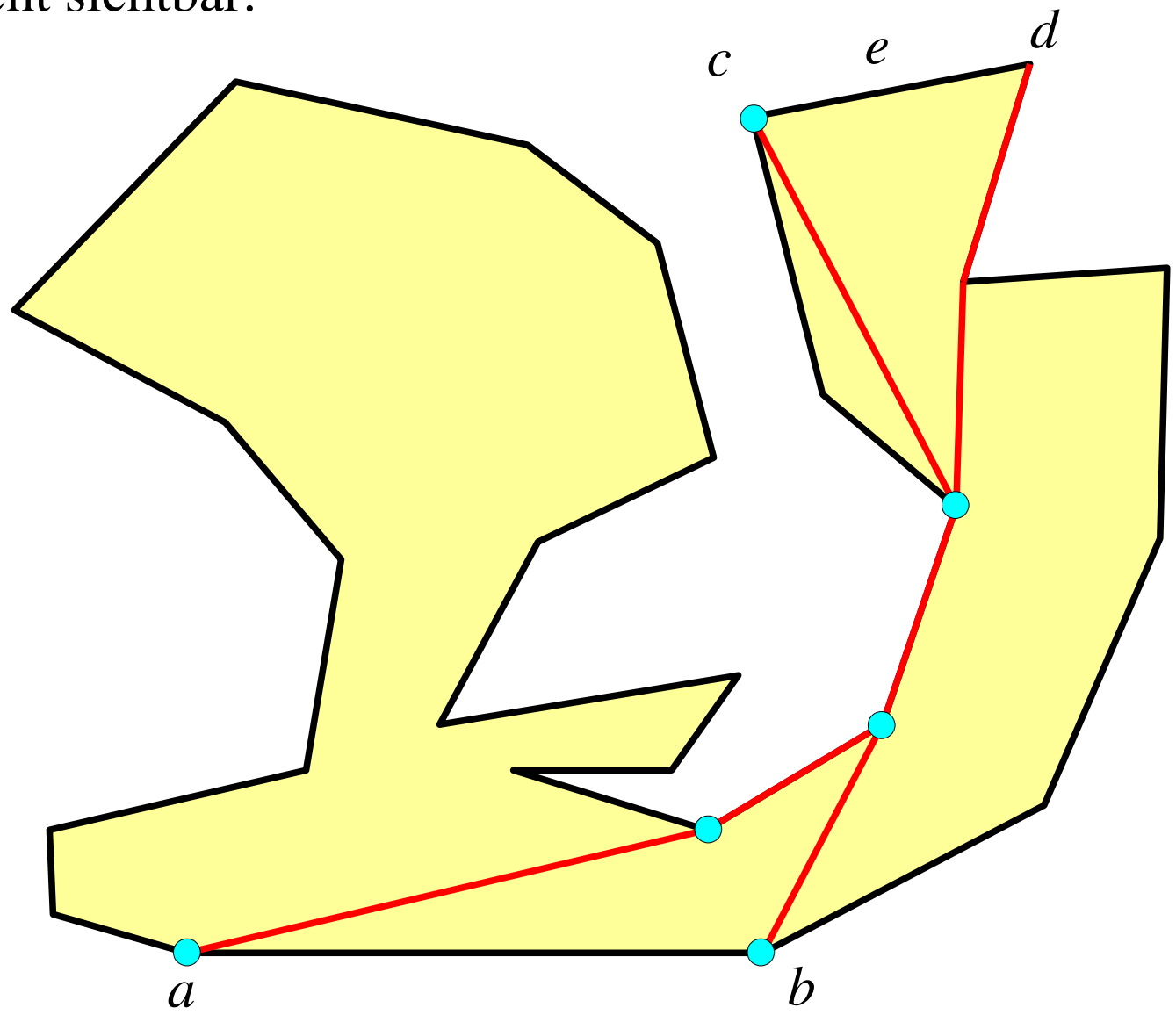


Algorithmus

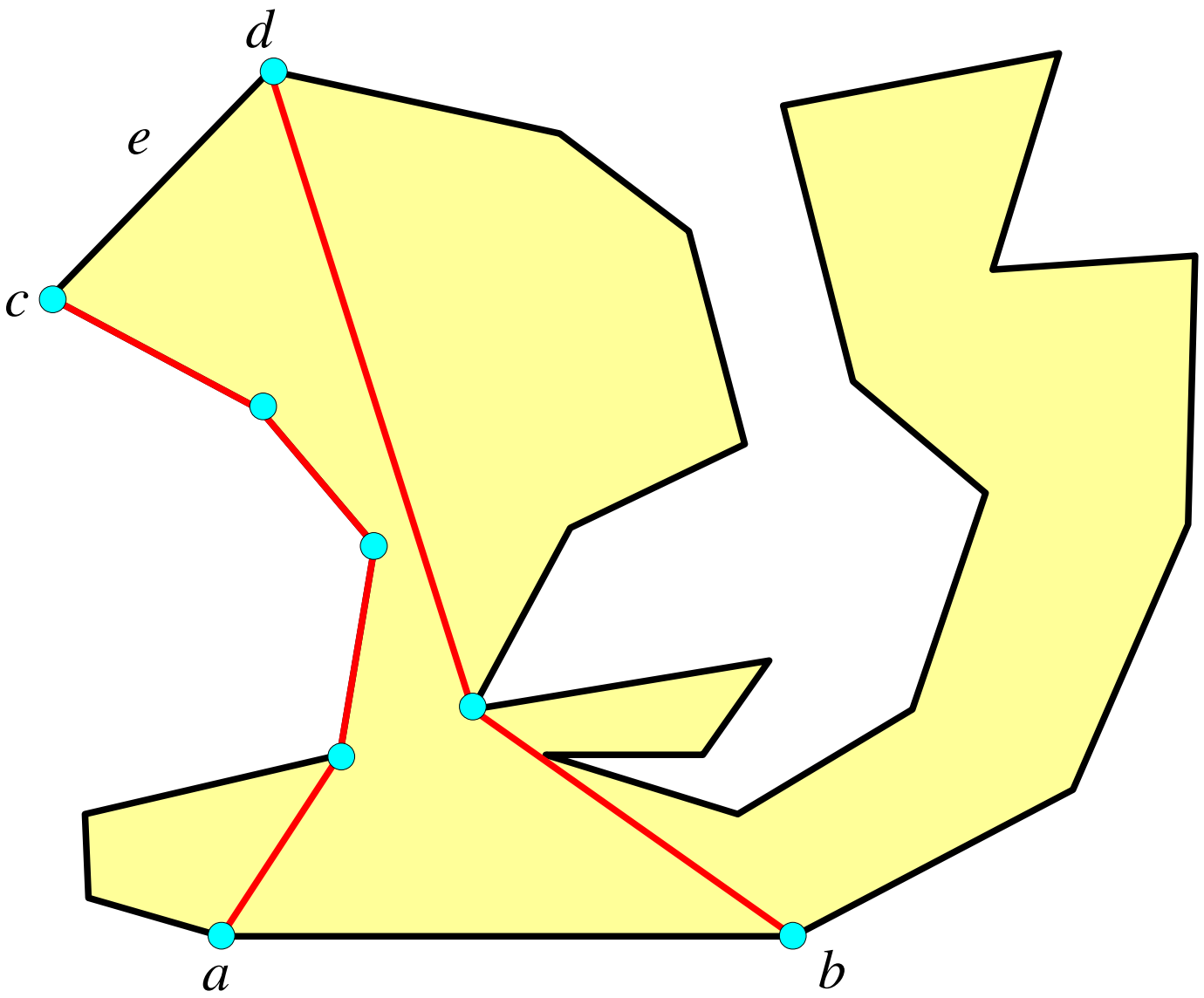
Wir markieren die Knoten, die man von b über entsprechend verlaufende reflexe Ketten erreicht.



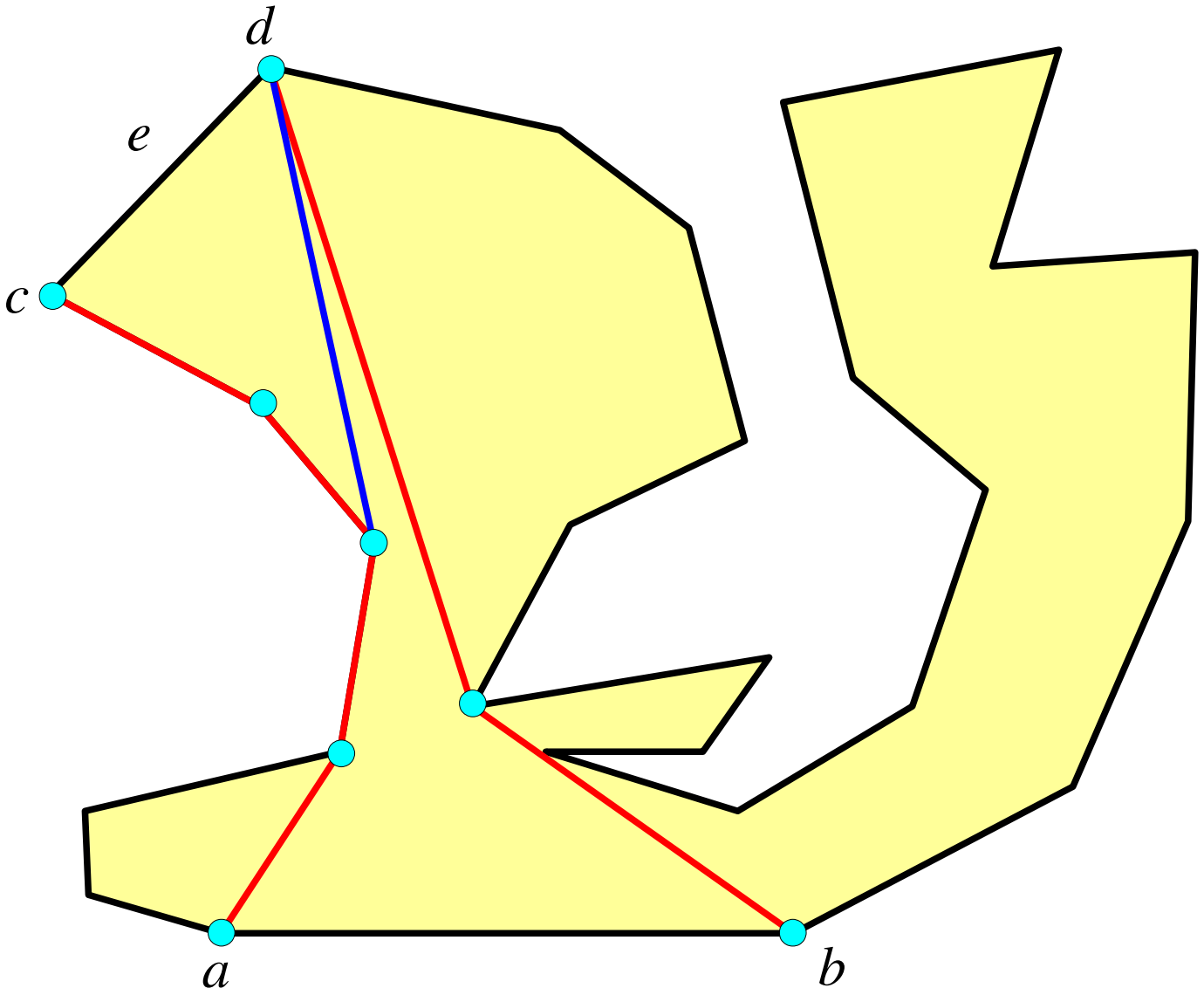
e ist nicht sichtbar.



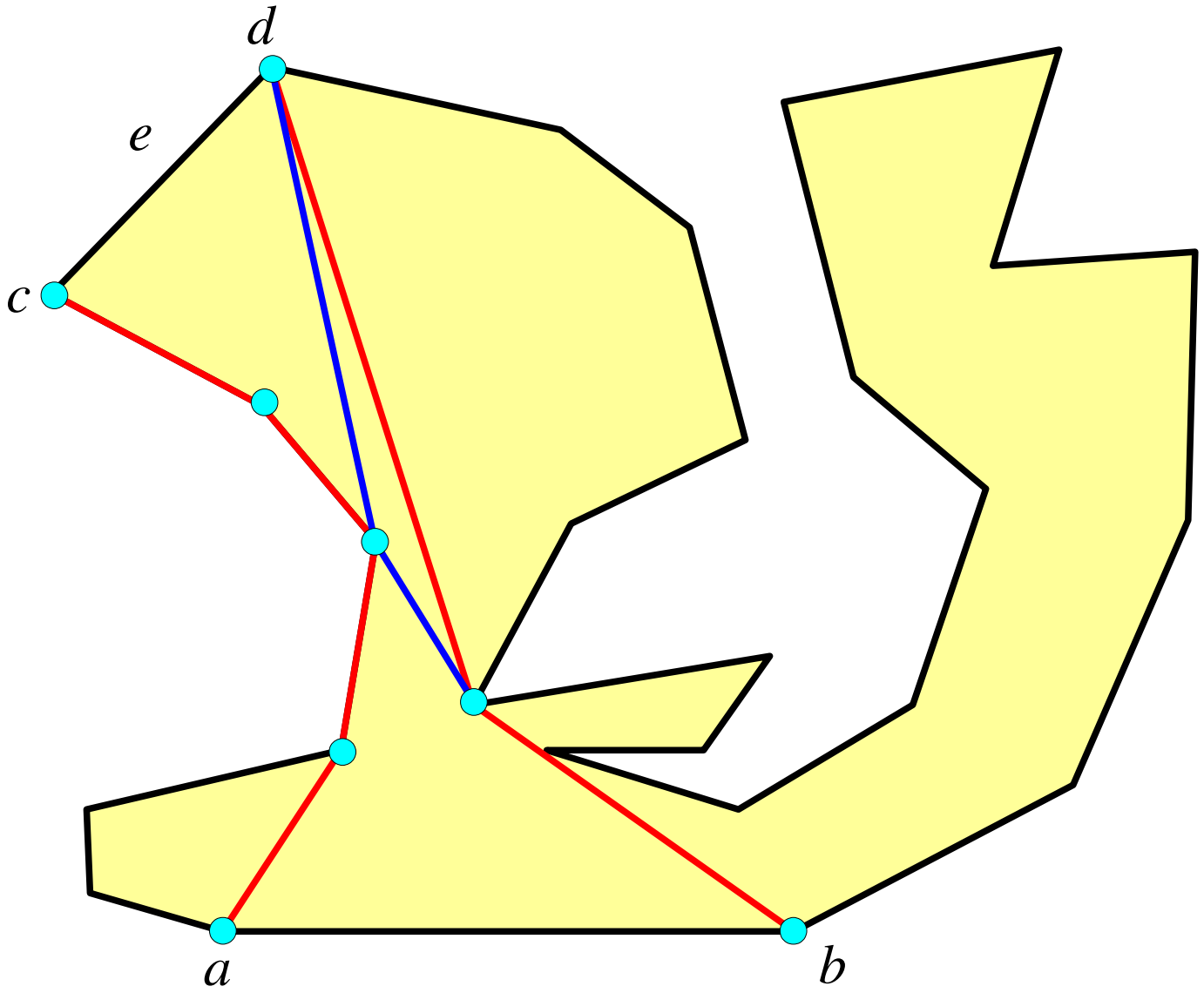
Ein Teil von *e* ist sichtbar.



Wir betrachten nun die Ecke, wo der kürzeste Weg von *a* nach *d* abzweigt...



...und die Ecke, wo der kürzeste Weg von *b* nach *c* abzweigt.



Damit gewinnen wir den sichtbaren Teil von e.