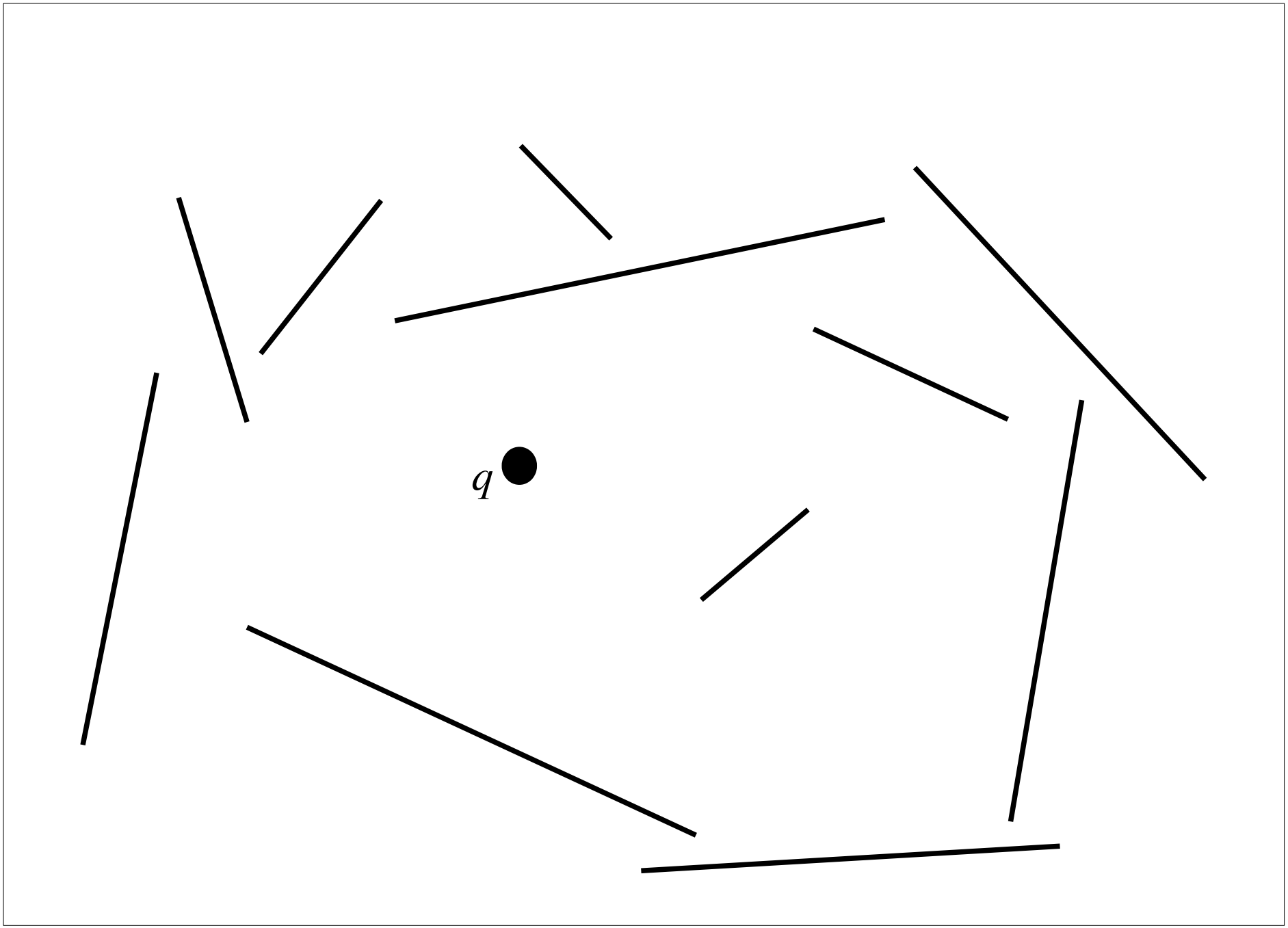
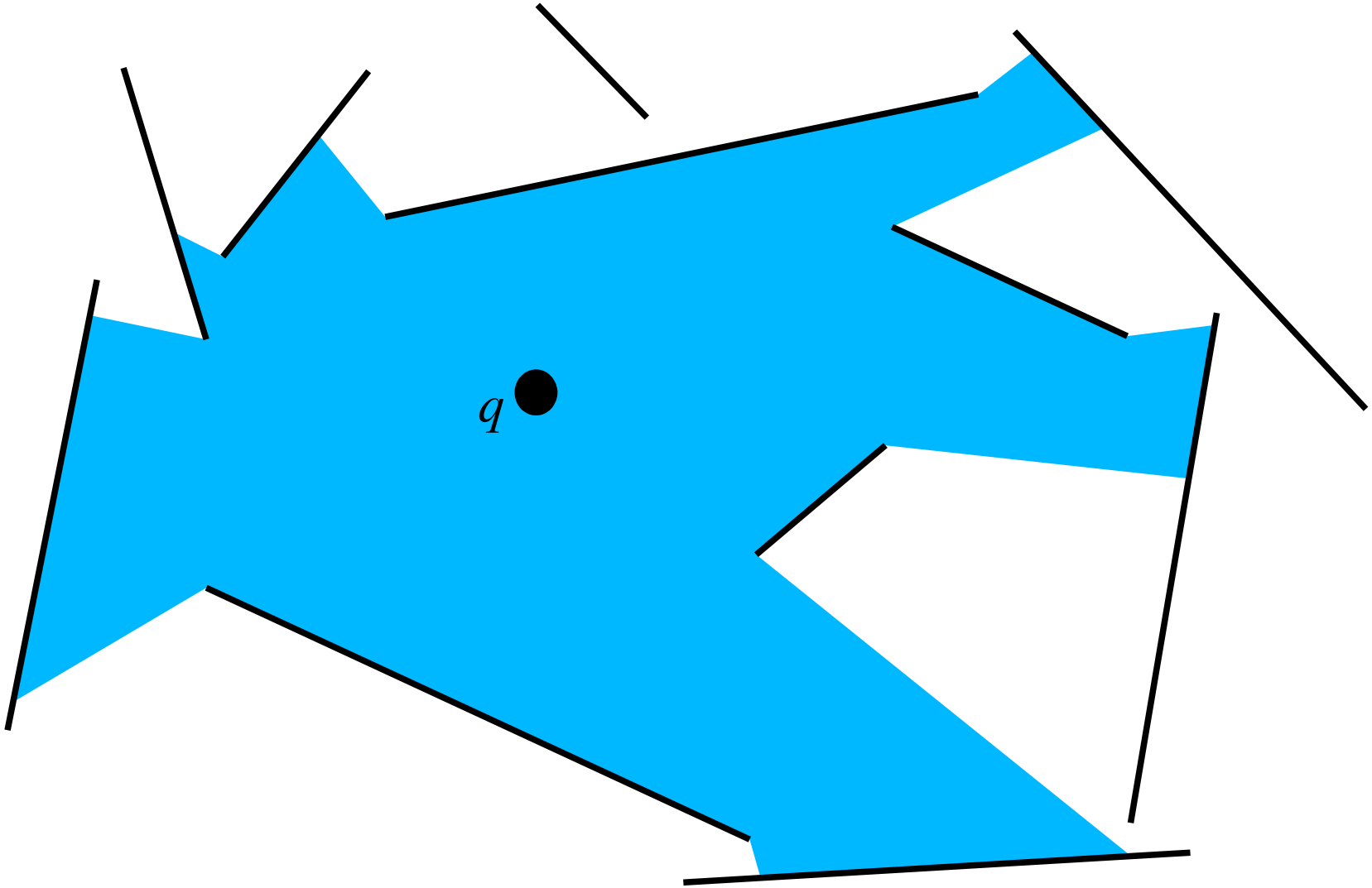
Sichtbarkeitsbereich eines Punktes

1. Beschreibung der Aufgabenstellung


q

Berechnet werden soll eine Beschreibung der Menge der Punkte, die von q aus gesehen werden.



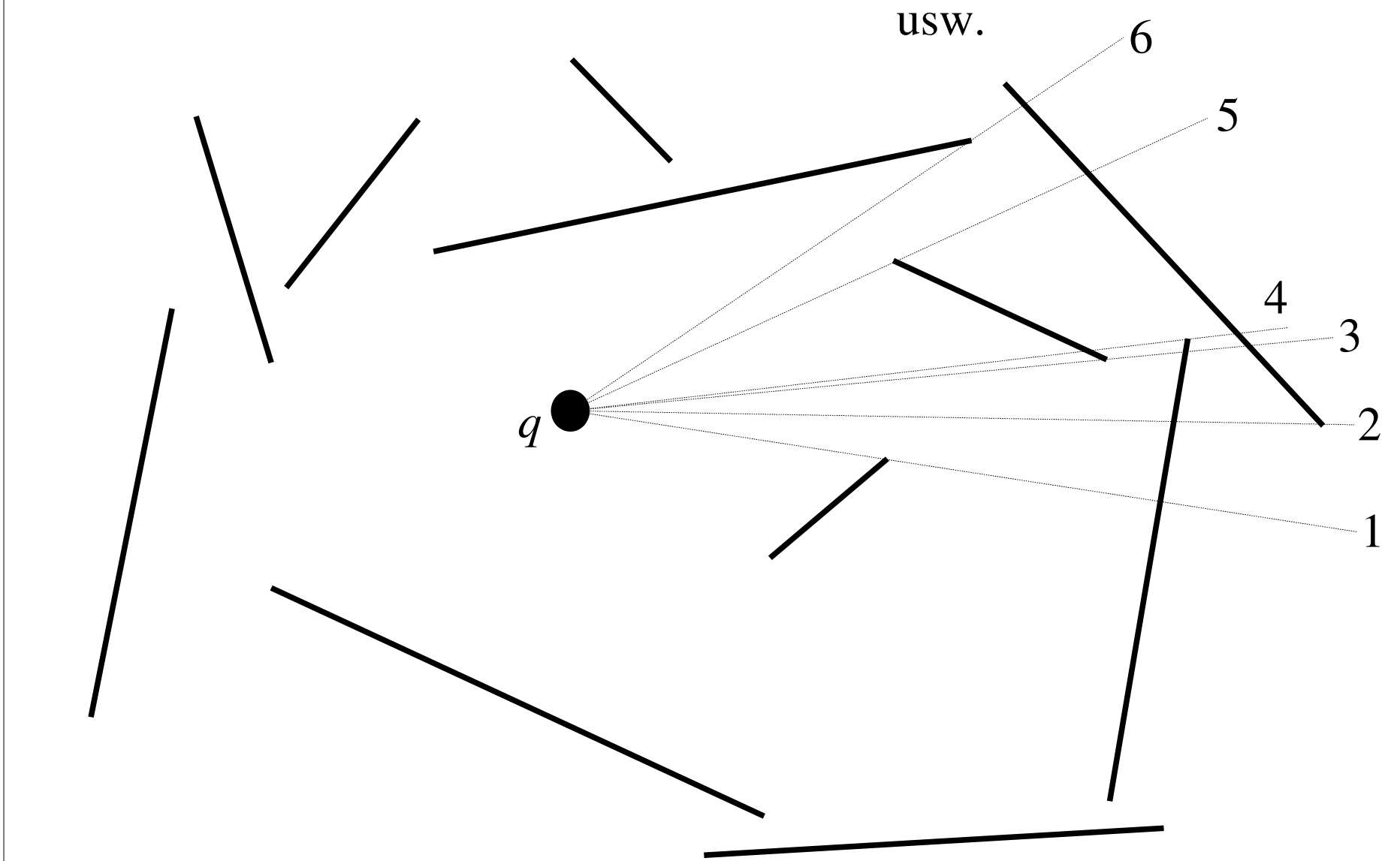
2. Idee für einen Algorithmus

Die Vorgehensweise ist ähnlich dem **Gleitgeradenverfahren** zur Bestimmung der Schnittpunkte von Strecken:

1. Sortieren der Streckenendpunkte nach Winkeln um q.

2. Gleitstrahl, der um q rotiert (**Winkelsweep**).

Sortierung der Endpunkte



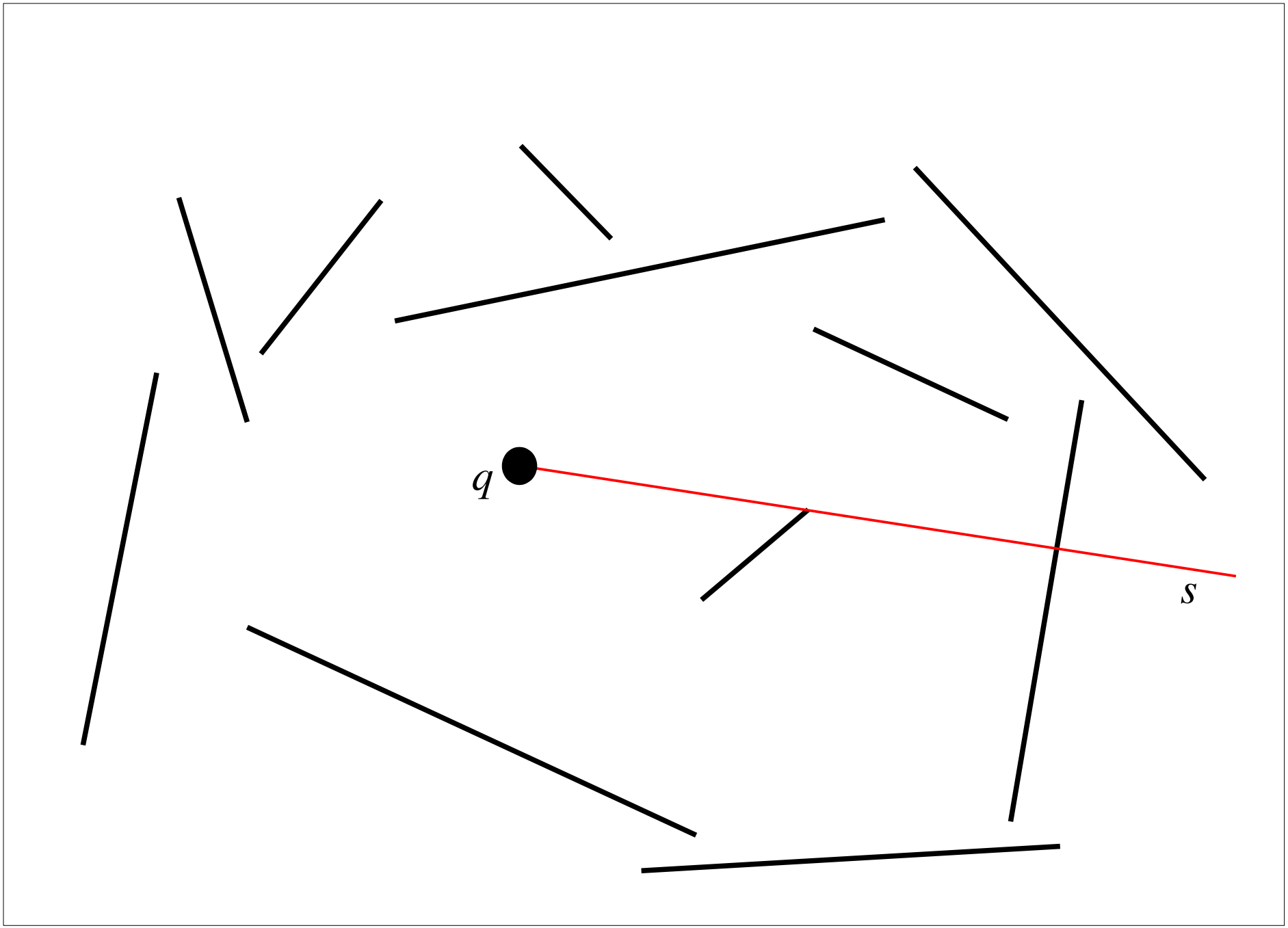
Grober Ablauf des Winkelsweep:

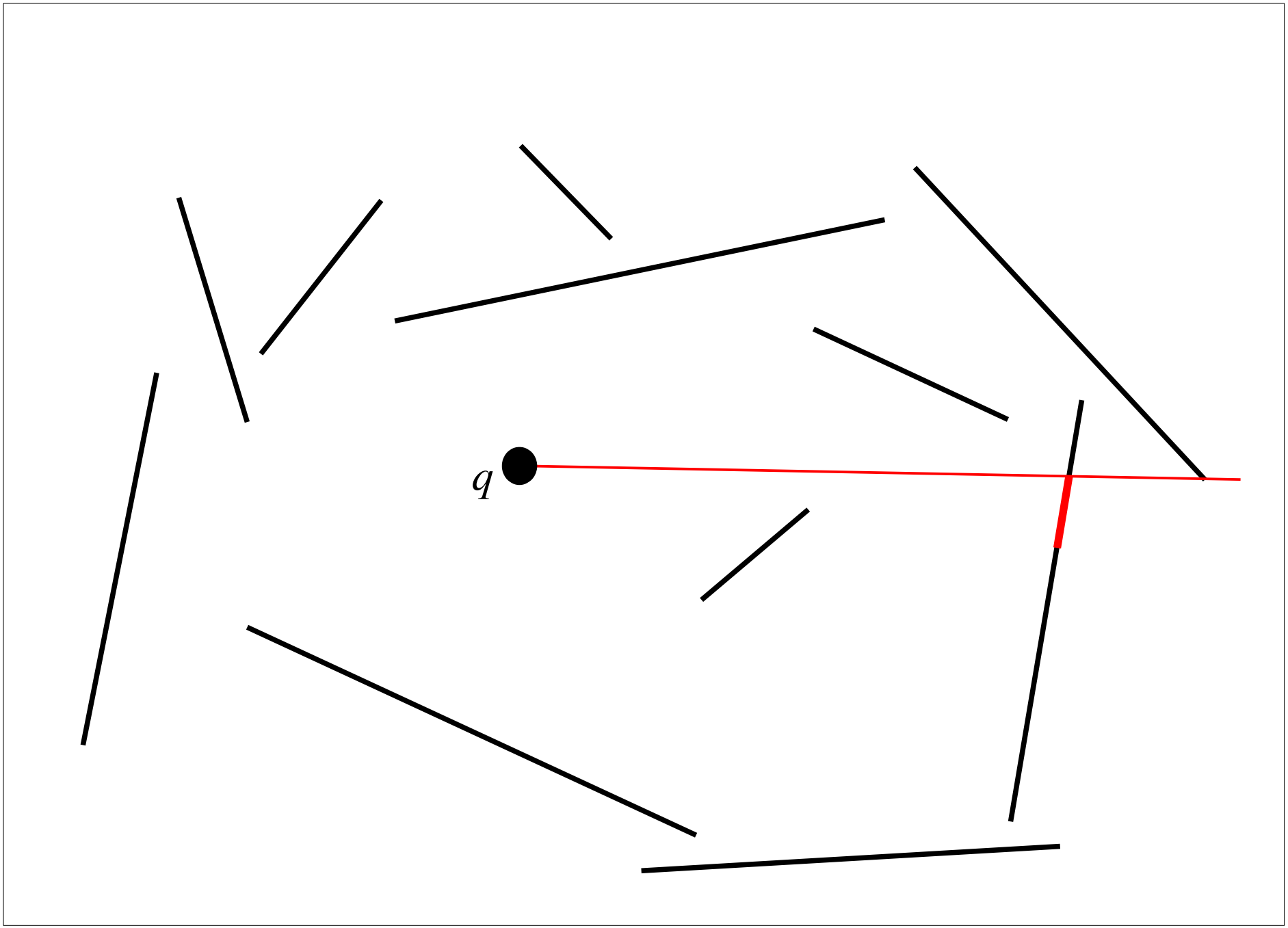
Wir starten bei Streckenendpunkt Nummer 1.

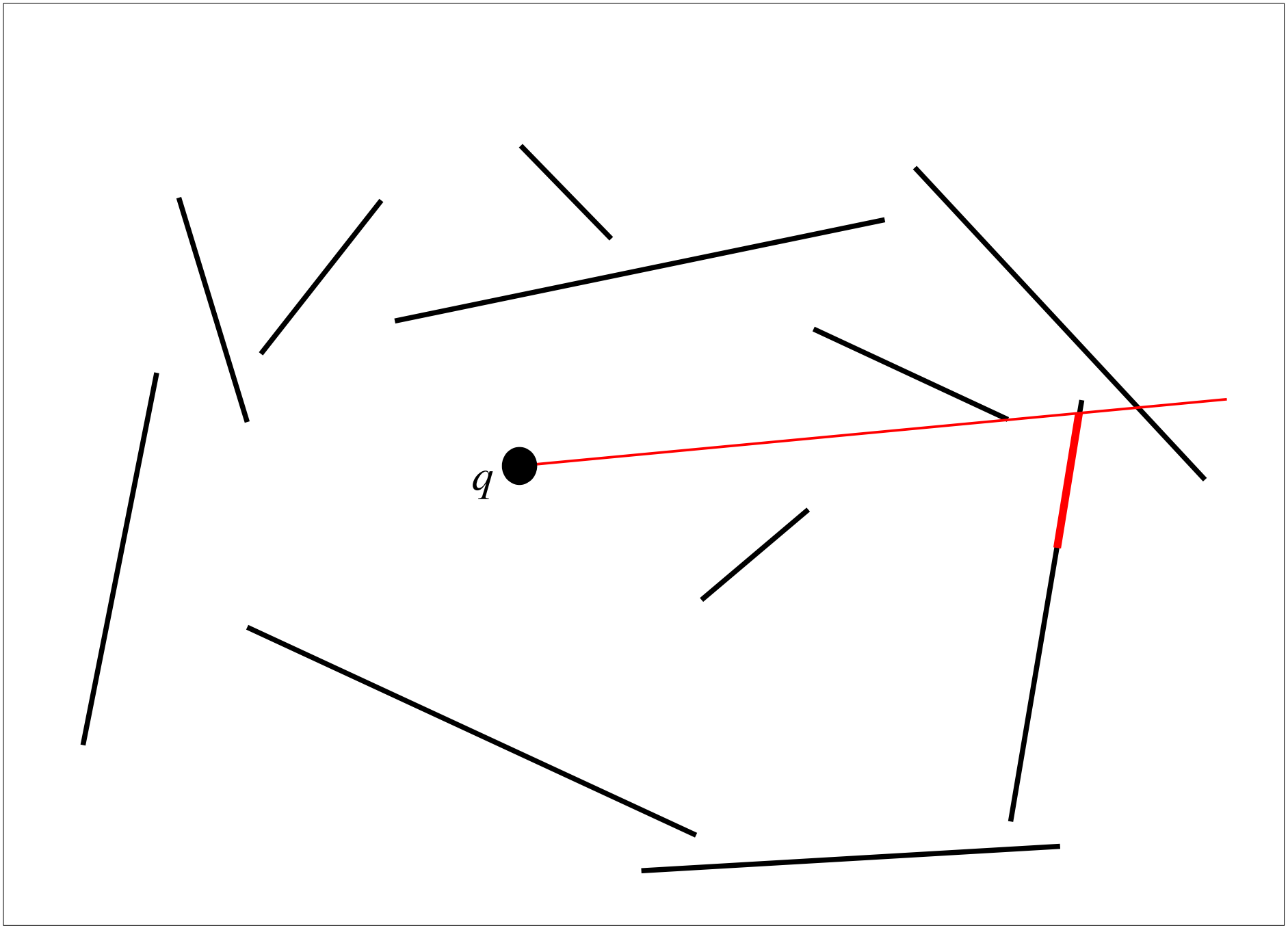
Zu jedem Zeitpunkt verwalten wir die vom Sweepstrahl *s* **geschnittenen Strecken** in einem Wörterbuch in der Reihenfolge, wie sie von *s* geschnitten werden.

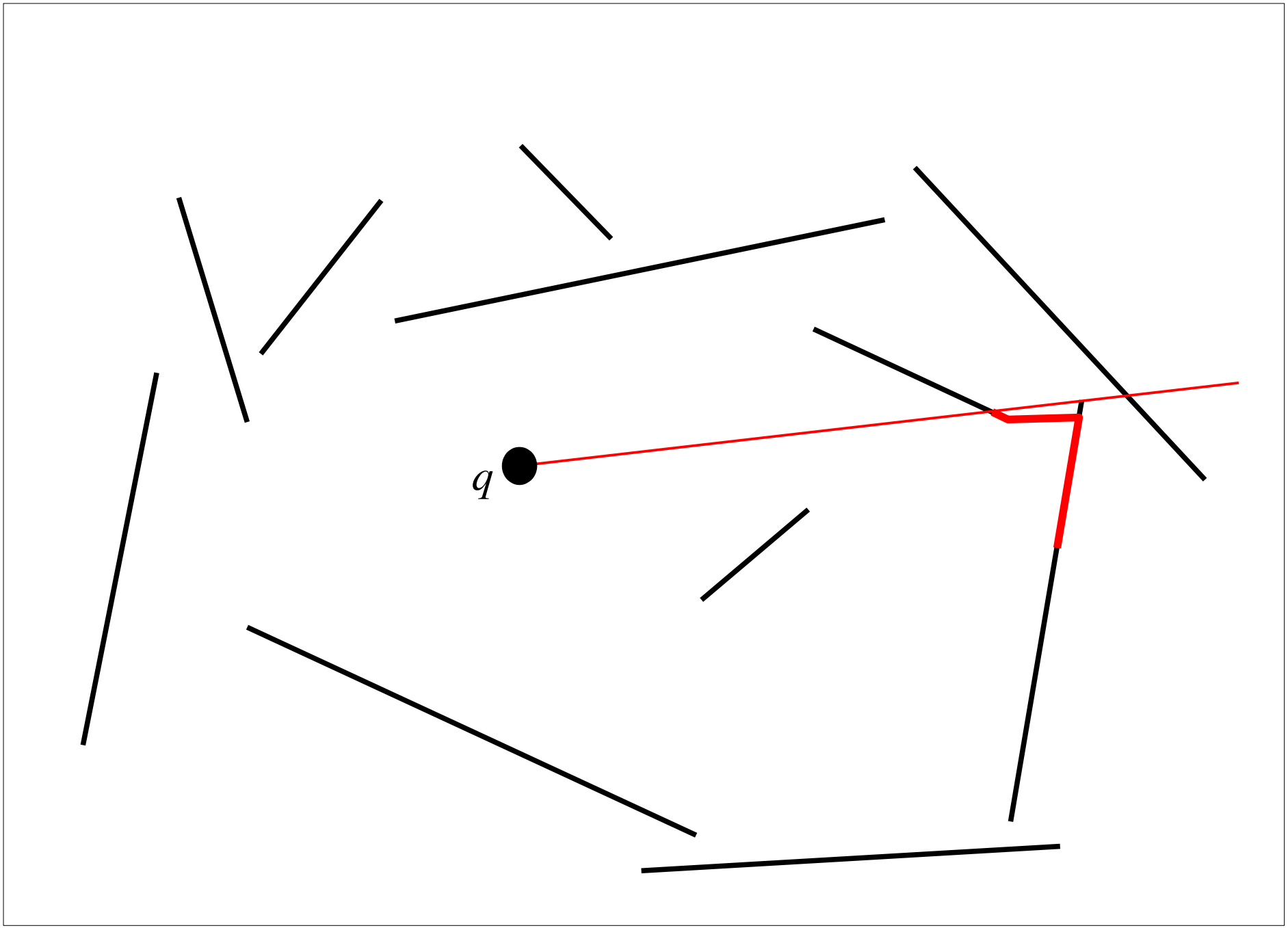
Das erste Element im Wörterbuch **ist sichtbar** von *q*.

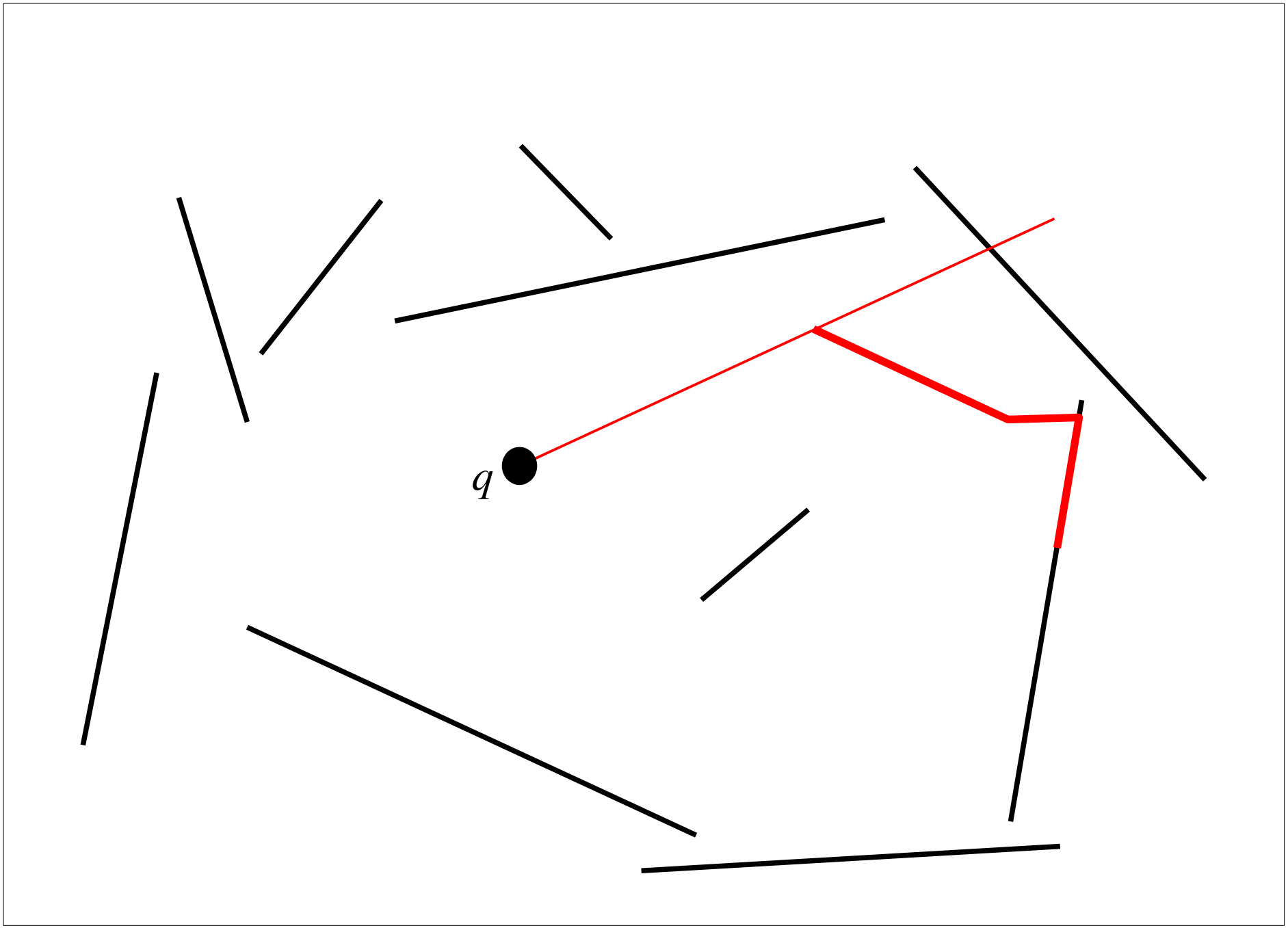
Natürlich **ändert** sich nur etwas an den Stellen, wo eine Strecke beginnt oder endet.

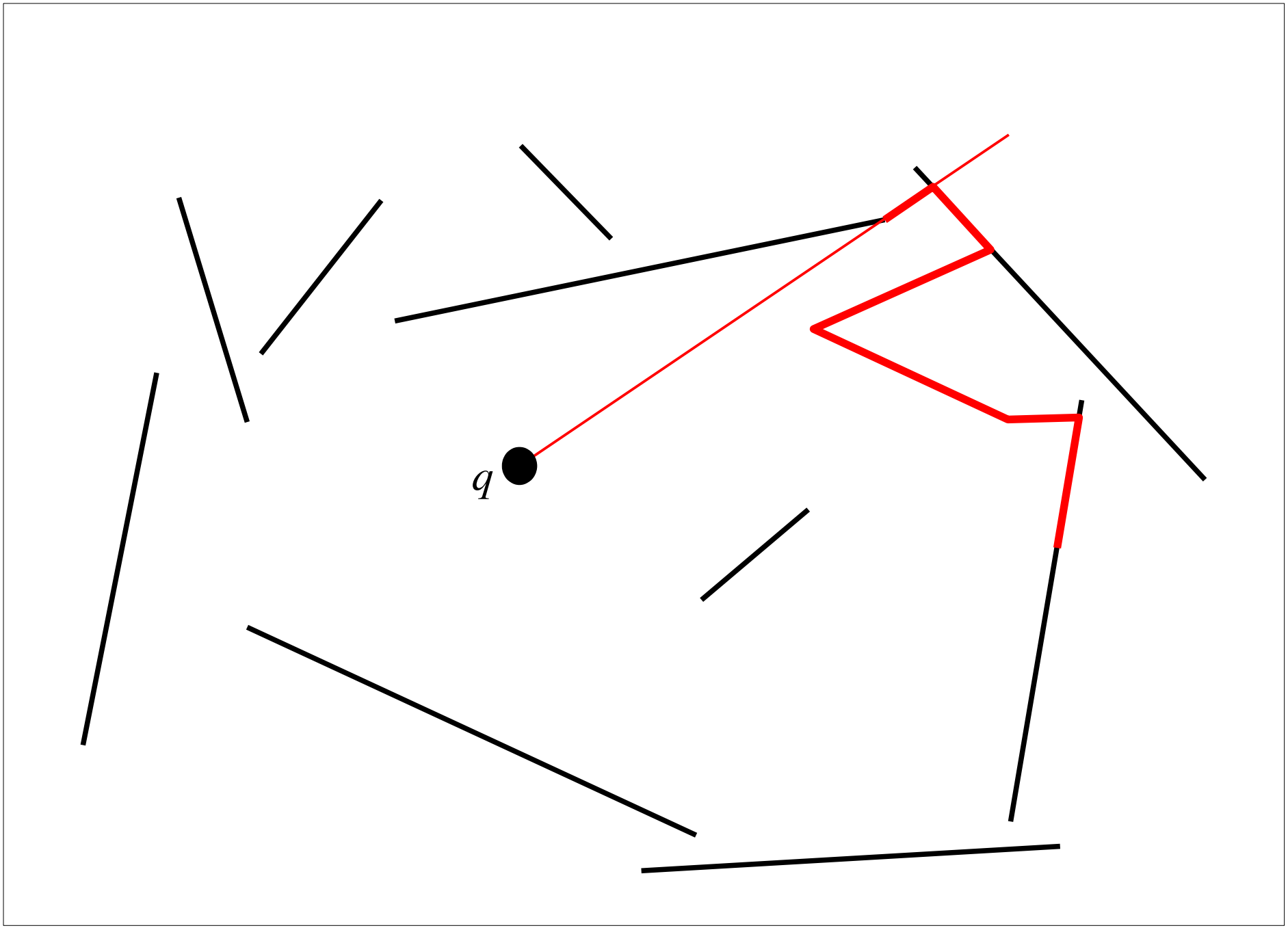

q
s

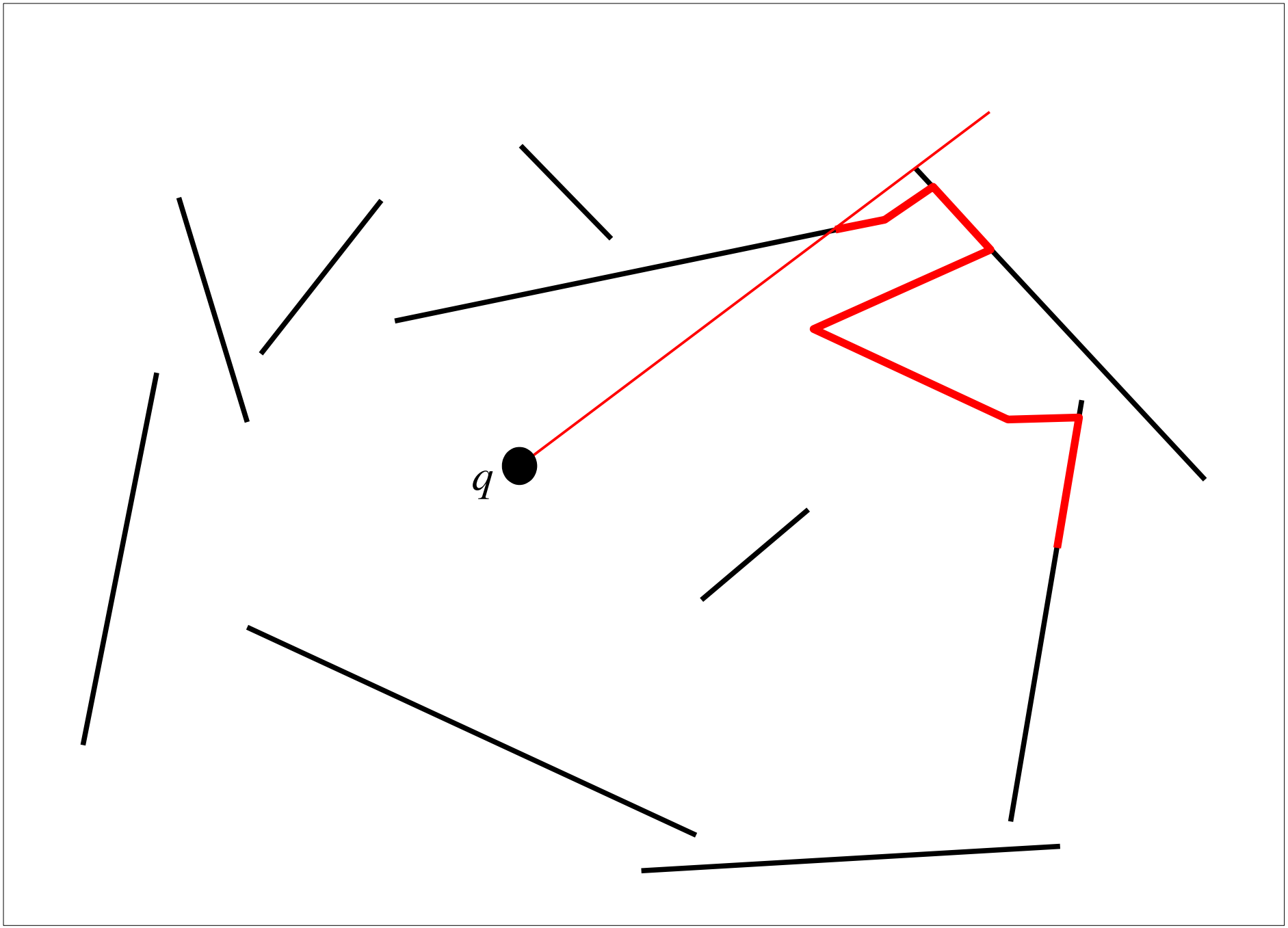

q

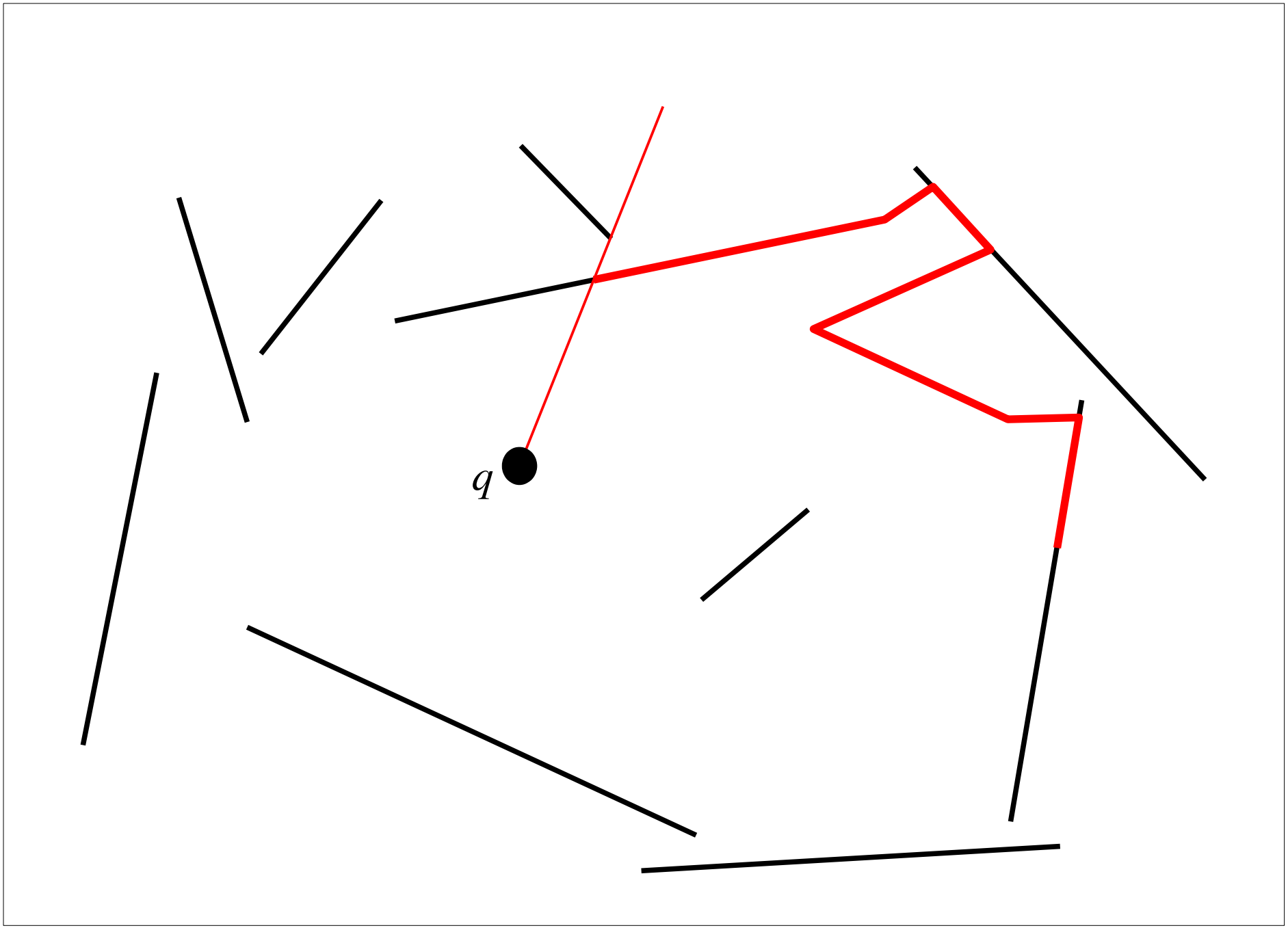

q

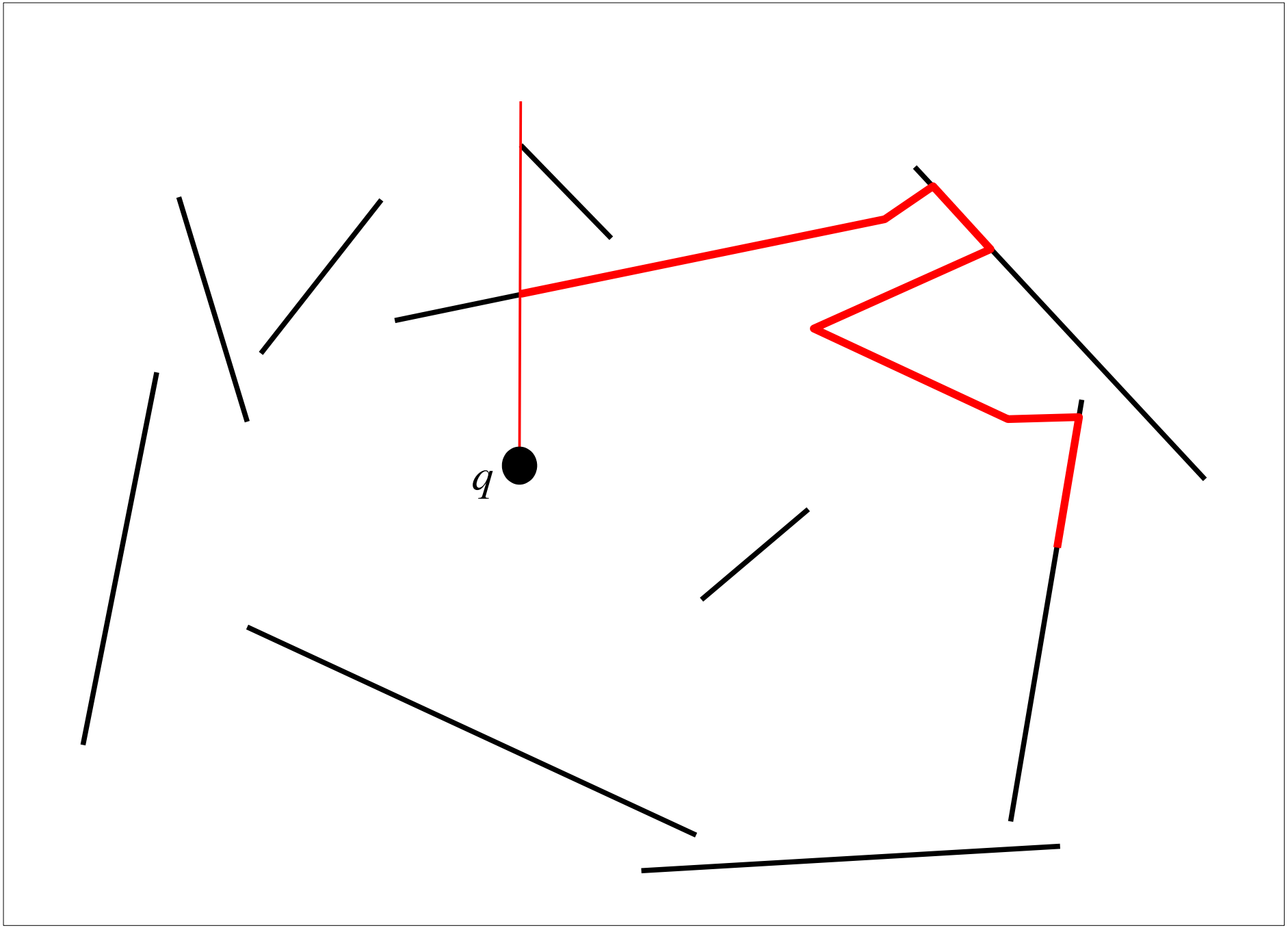

q

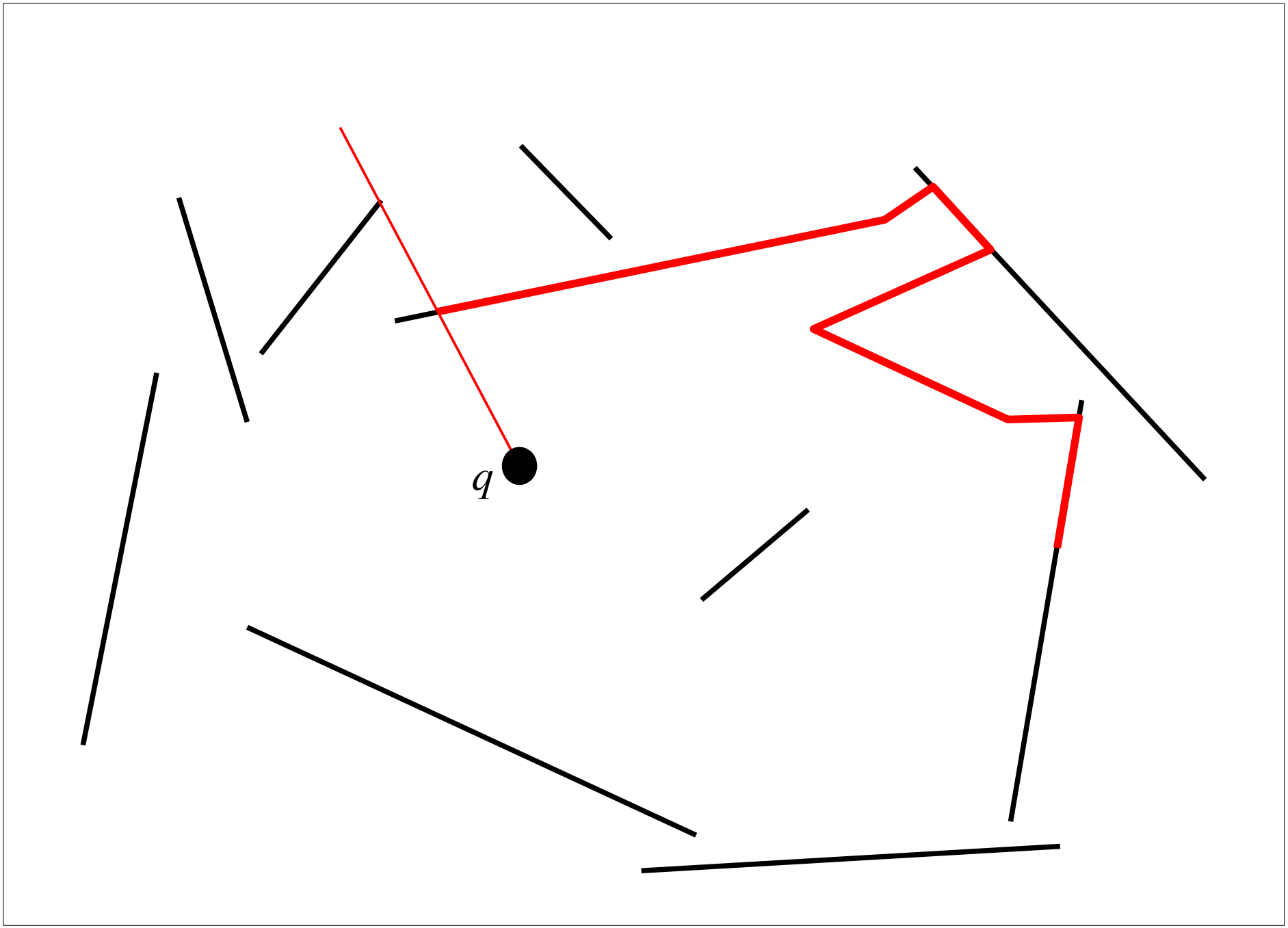

q


q


q


q


q


q

Zu überlegen ist noch, wie genau man

- die **Sweepstatus**-Datenstruktur und
- die **Ereignispunkte**-Datenstruktur

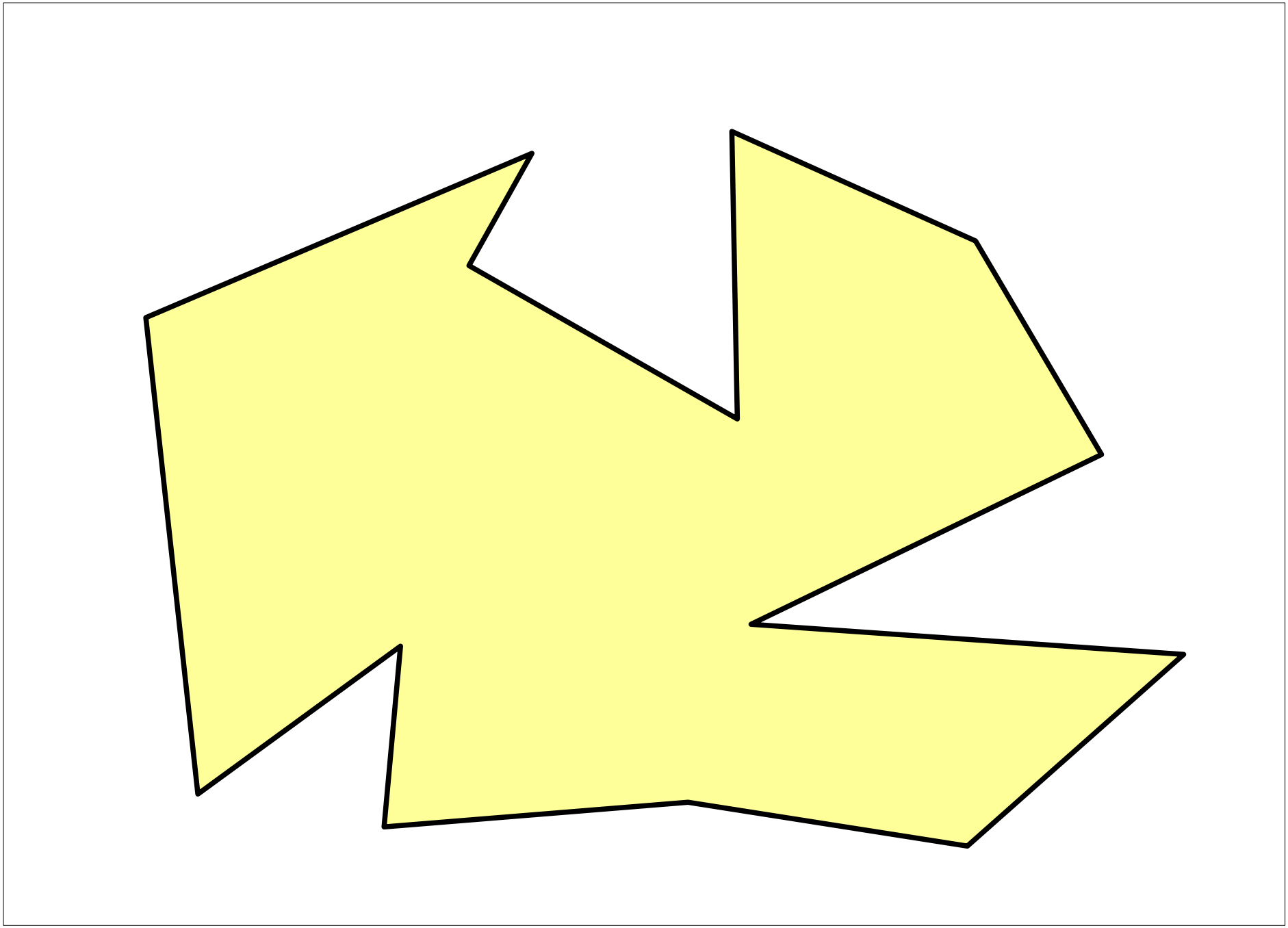
umsetzt. Man kann erreichen:

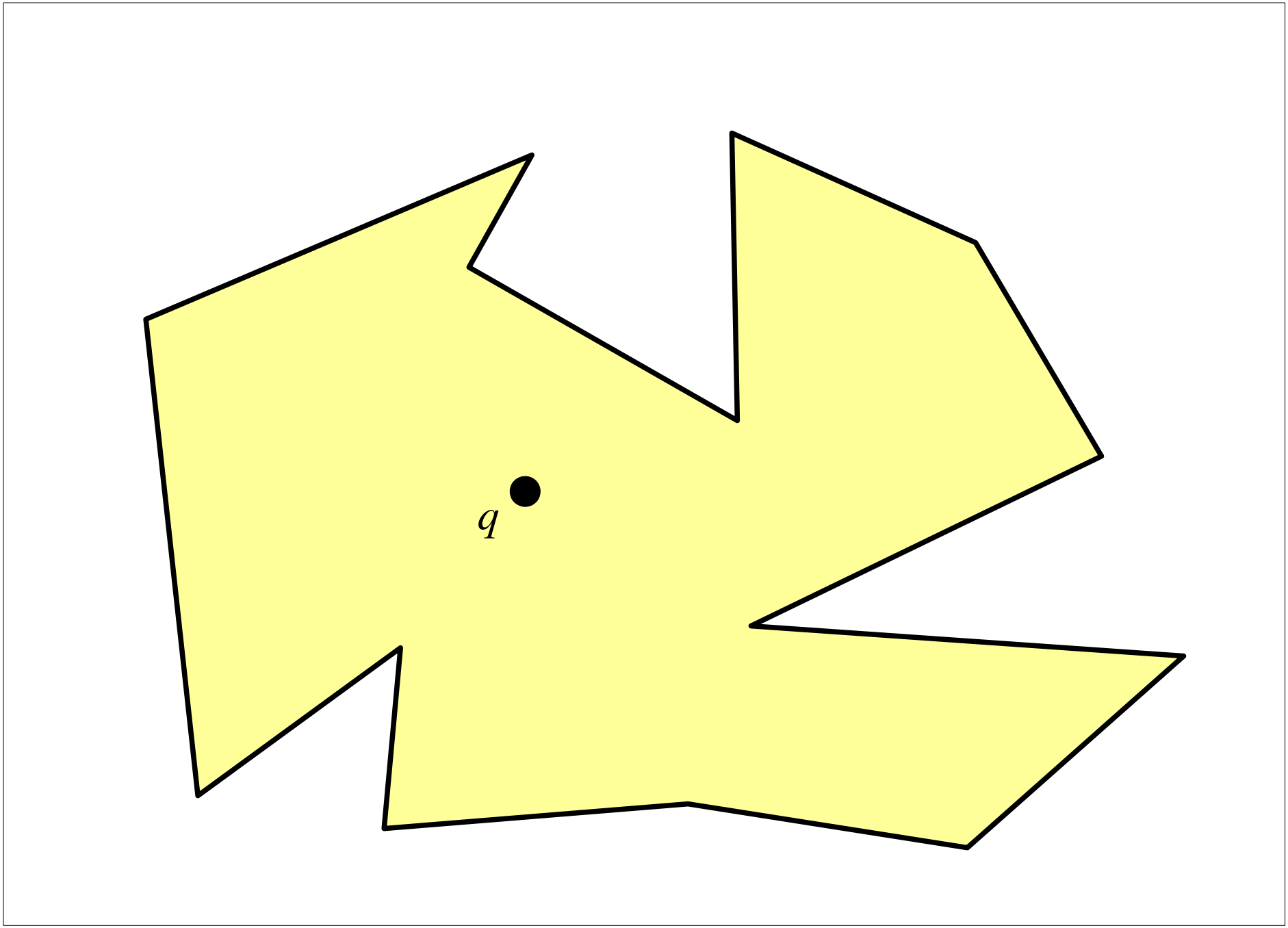
Pro Ereignispunkt: $O(\log n)$

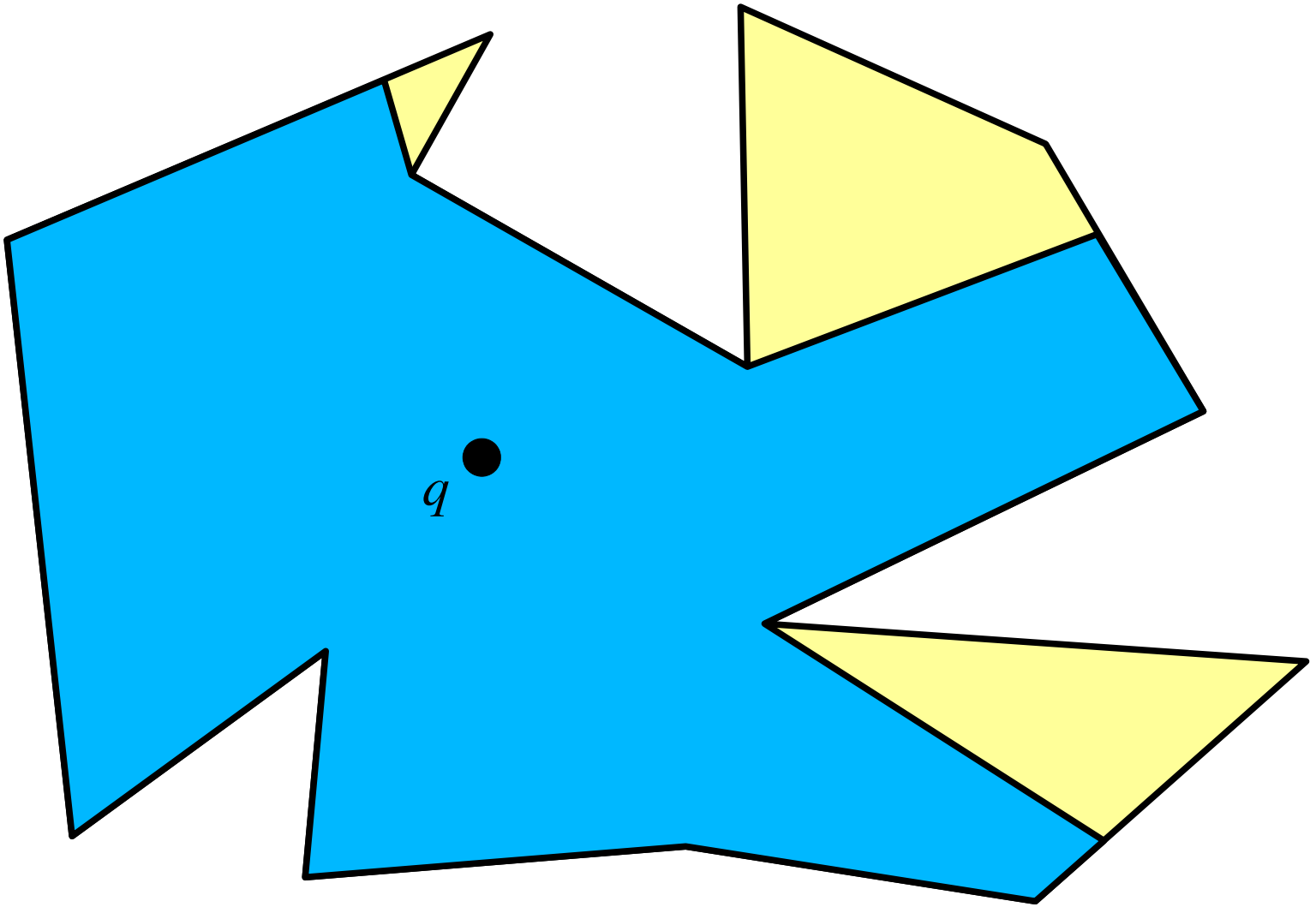
Sortieren zu Beginn: $O(n \log n)$

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

3. Sichtbarkeitsbereich eines Punktes in einem einfachen Polygon




q

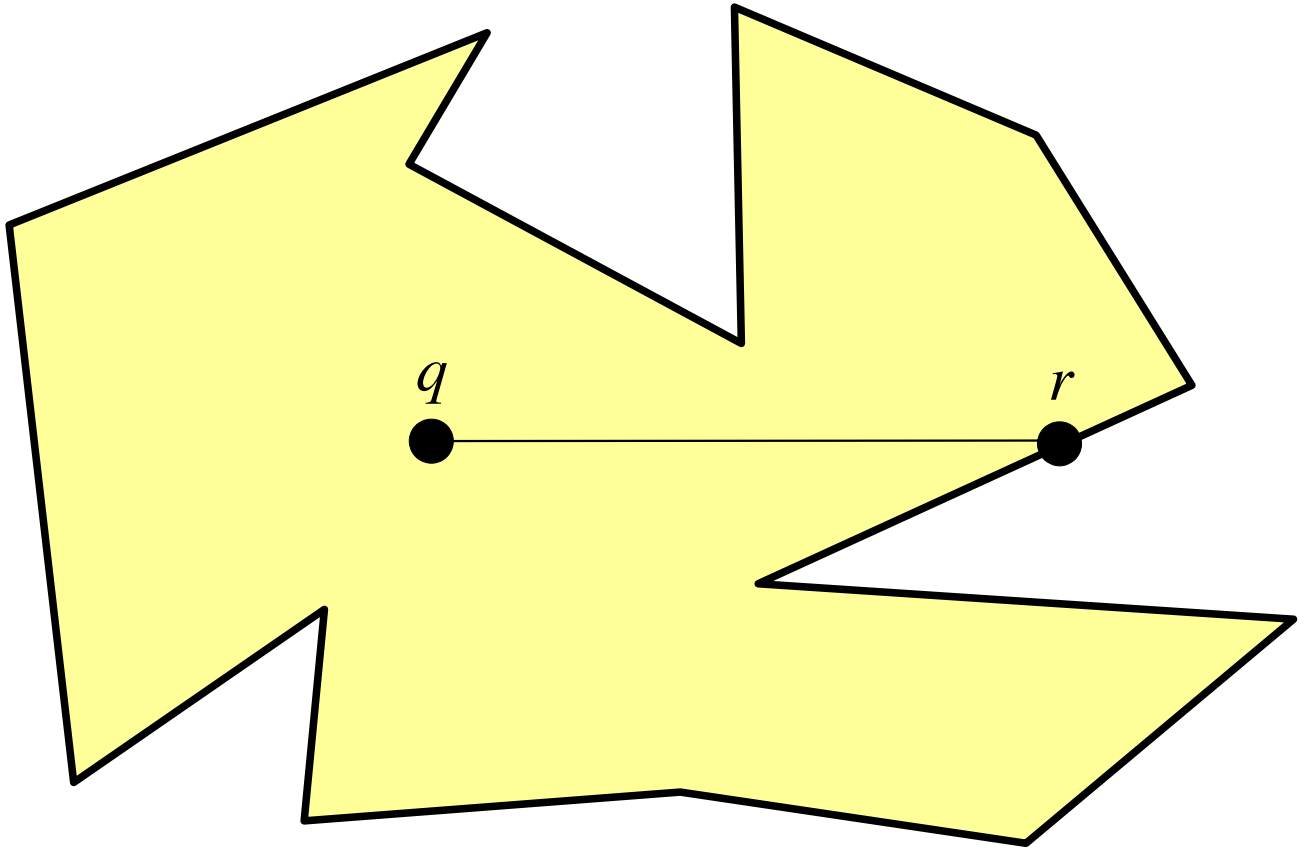

q

Man kann natürlich den bereits behandelten Algorithmus für eine allgemeine Menge von Strecken verwenden.

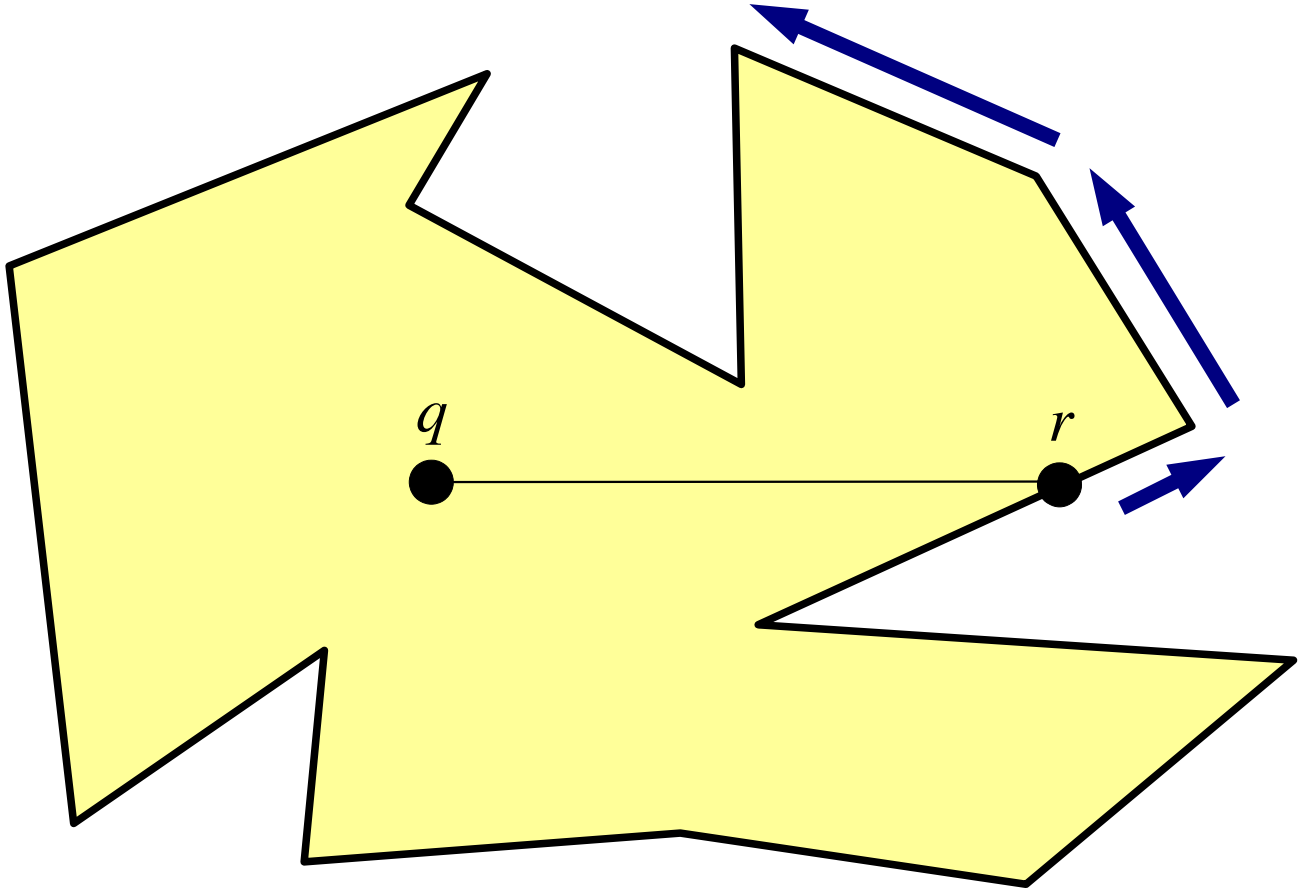
Unser Ziel:
Wir wollen **ohne Sortieren** auskommen und eine Laufzeit in $O(n)$ erreichen.

4. Beschreibung des Algorithmus

1. Gehe von q aus nach rechts, bis zum ersten Punkt r auf dem Rand von P.



2. Durchlaufe von r aus den Rand von P gegen den Uhrzeigersinn.



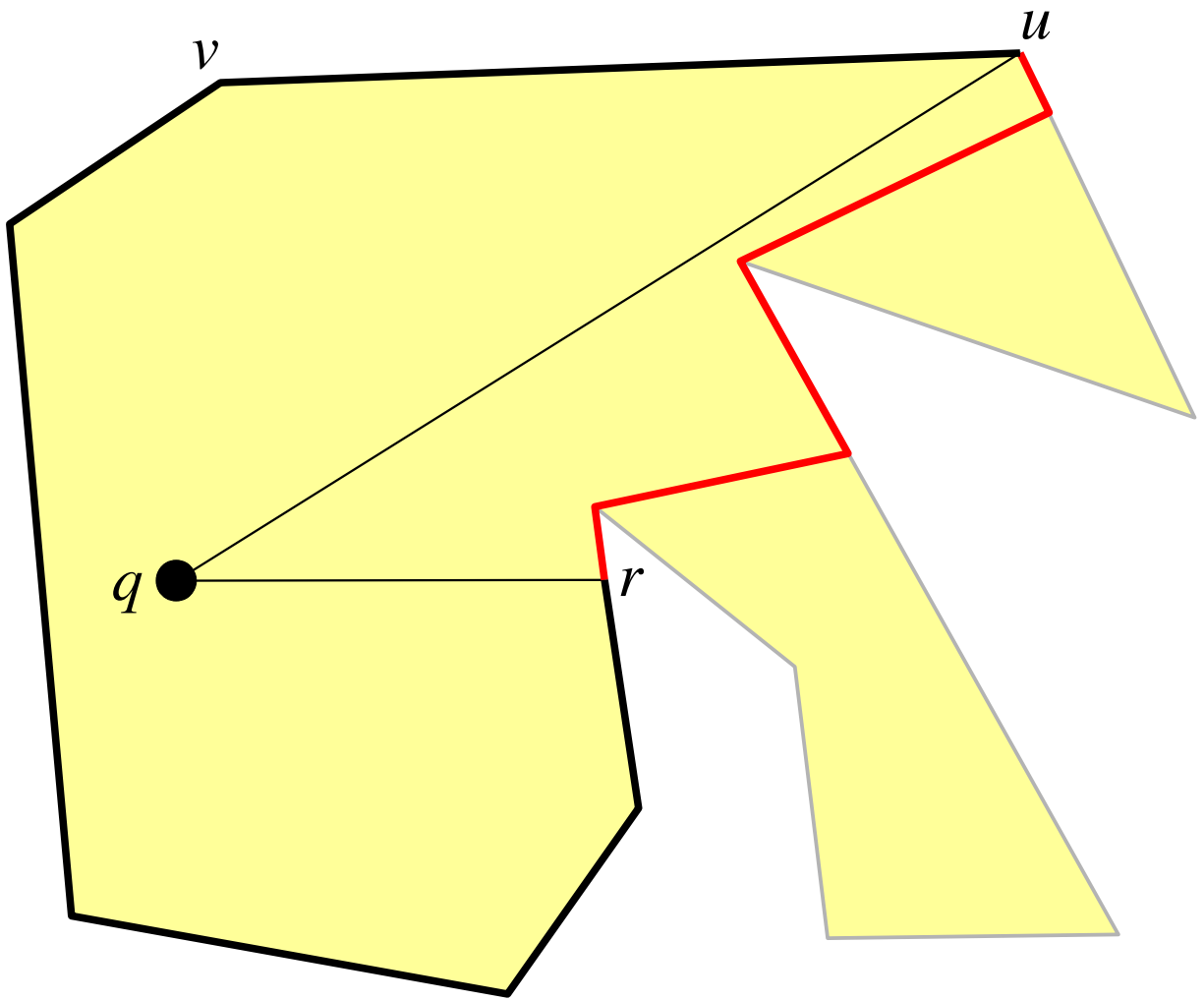
Bemerkungen:

Natürlich können wir den Rand von *P* nicht kontinuierlich durchlaufen, sondern wir gehen in **diskreten Schritten** von Eckpunkt zu Eckpunkt.

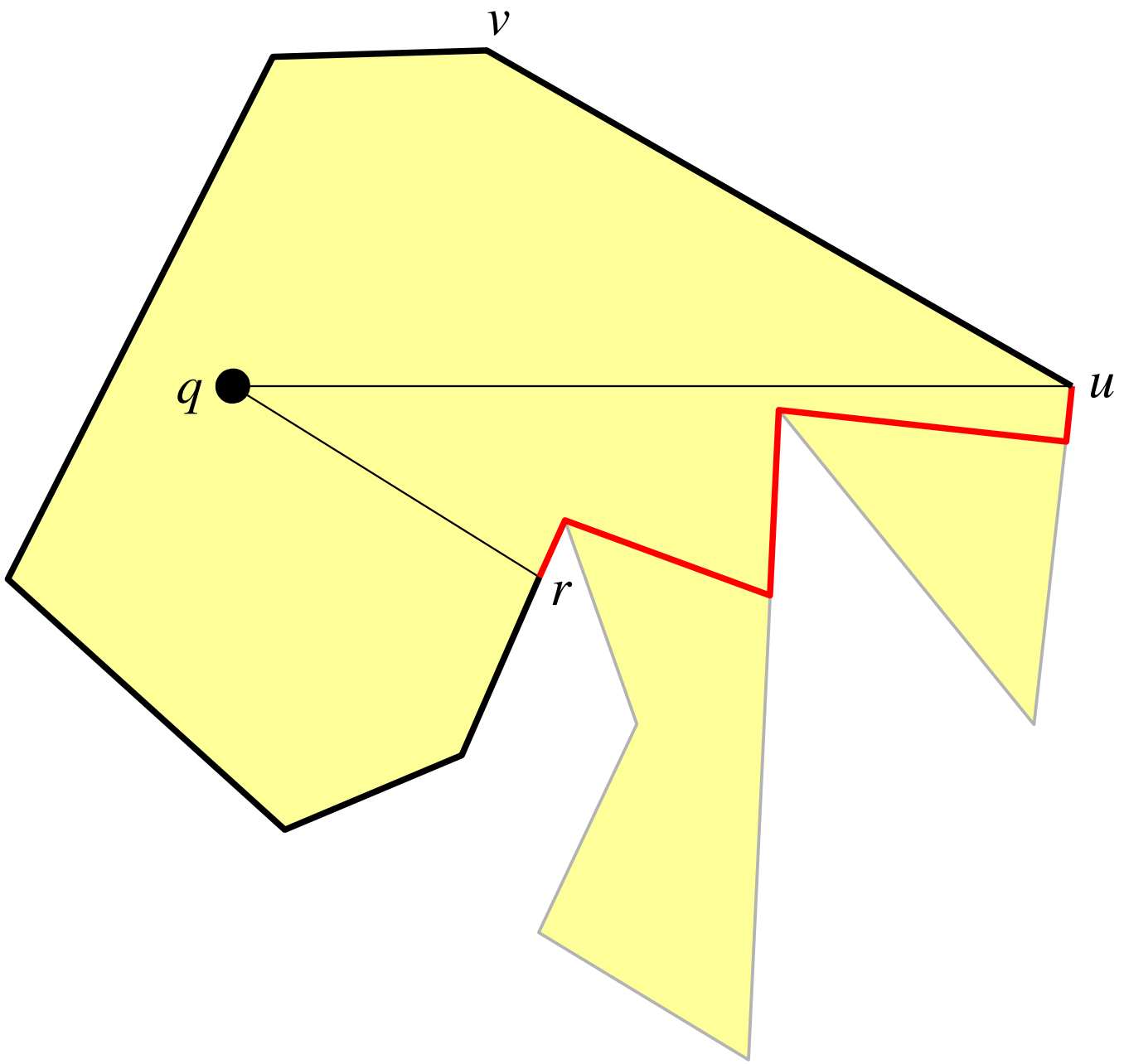
Den Teil des bis zu einem bestimmten Zeitpunkt abgelaufenen Randes von *P,* welcher möglicherweise von *q* aus sichtbar ist, verwalten wir in einem **Stapel**.

Wir gehen im Folgenden davon aus, dass wir die **Ecke *u* als letzte** betrachtet haben und müssen uns überlegen, welche Möglichkeiten es gibt, wenn wir die **nächste Ecke *v*** abarbeiten wollen.

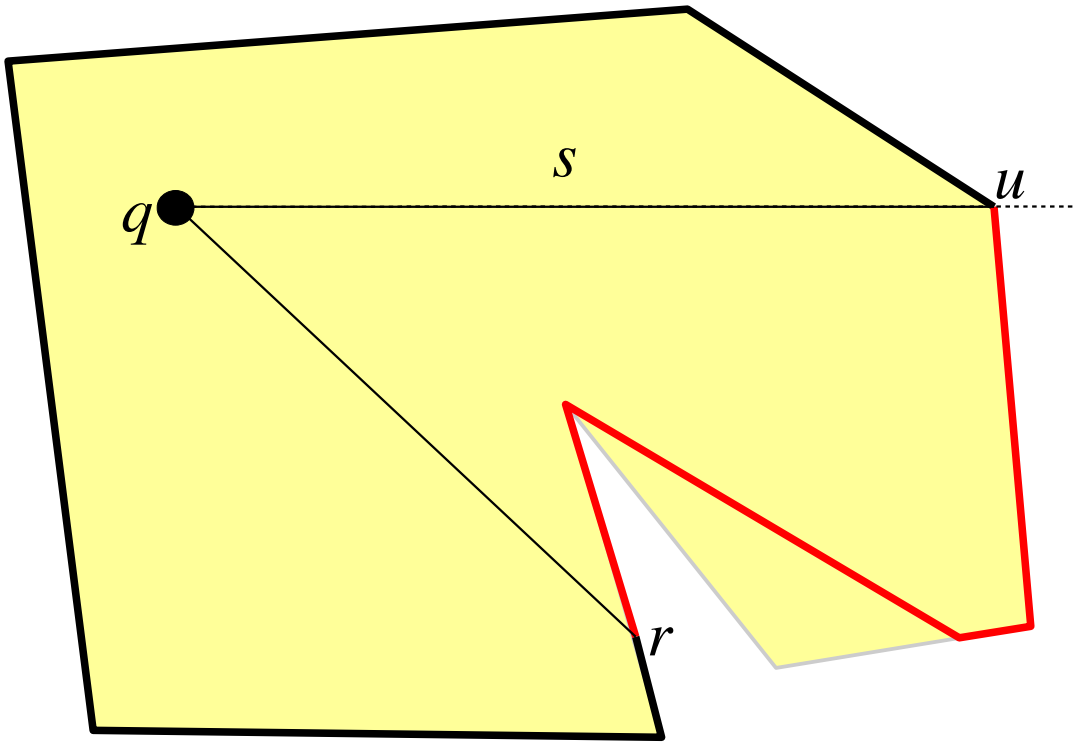
Beispiel für die vorliegende Situation:



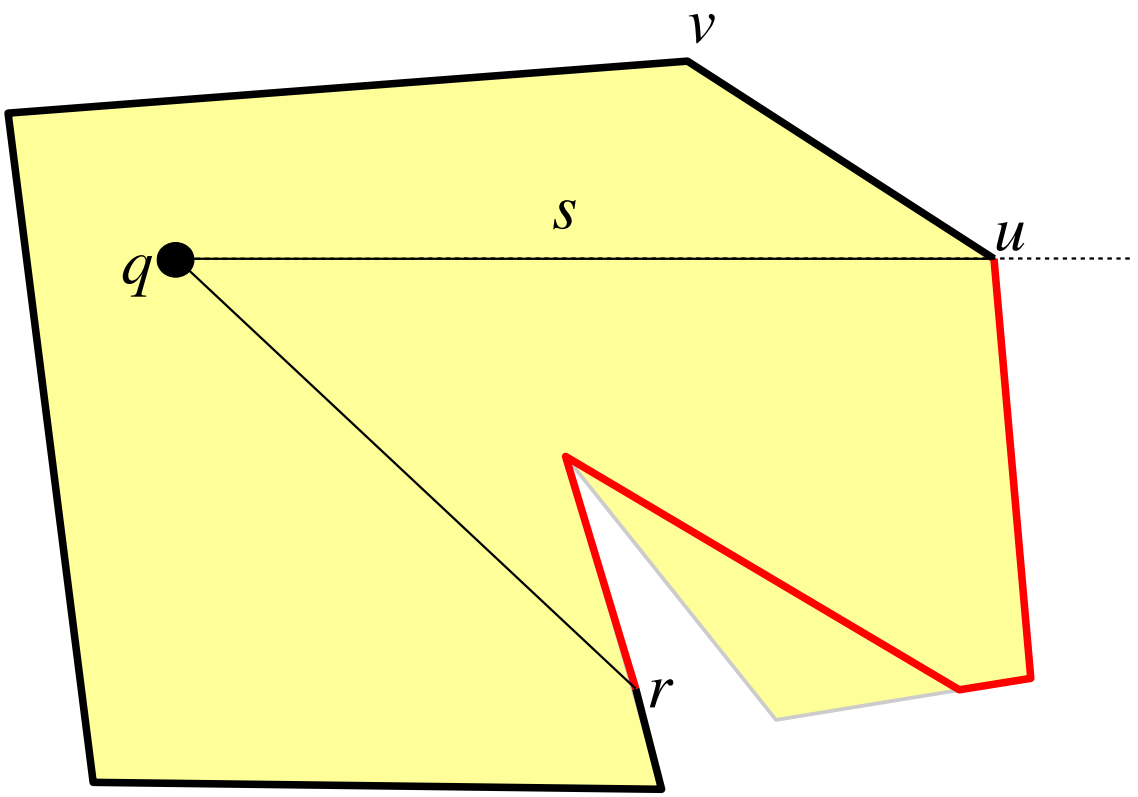
Der Einfachheit halber gehen wir davon aus, dass wir das Polygon so “hinlegen”, dass die Strecke zwischen q und u **waagerecht** verläuft.



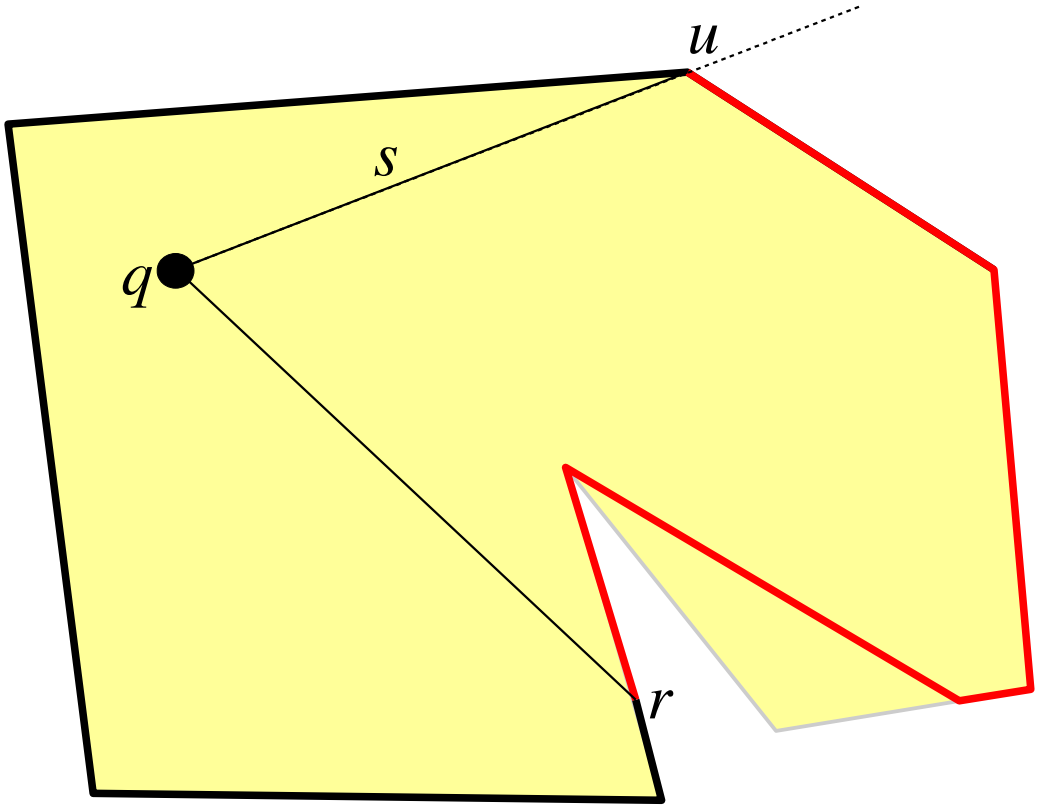
1. Fall: Der Rand von P trifft bei u von unten auf den Strahl s mit Startpunkt q durch u.



1.1. Unterfall: v liegt oberhalb des Strahls s.



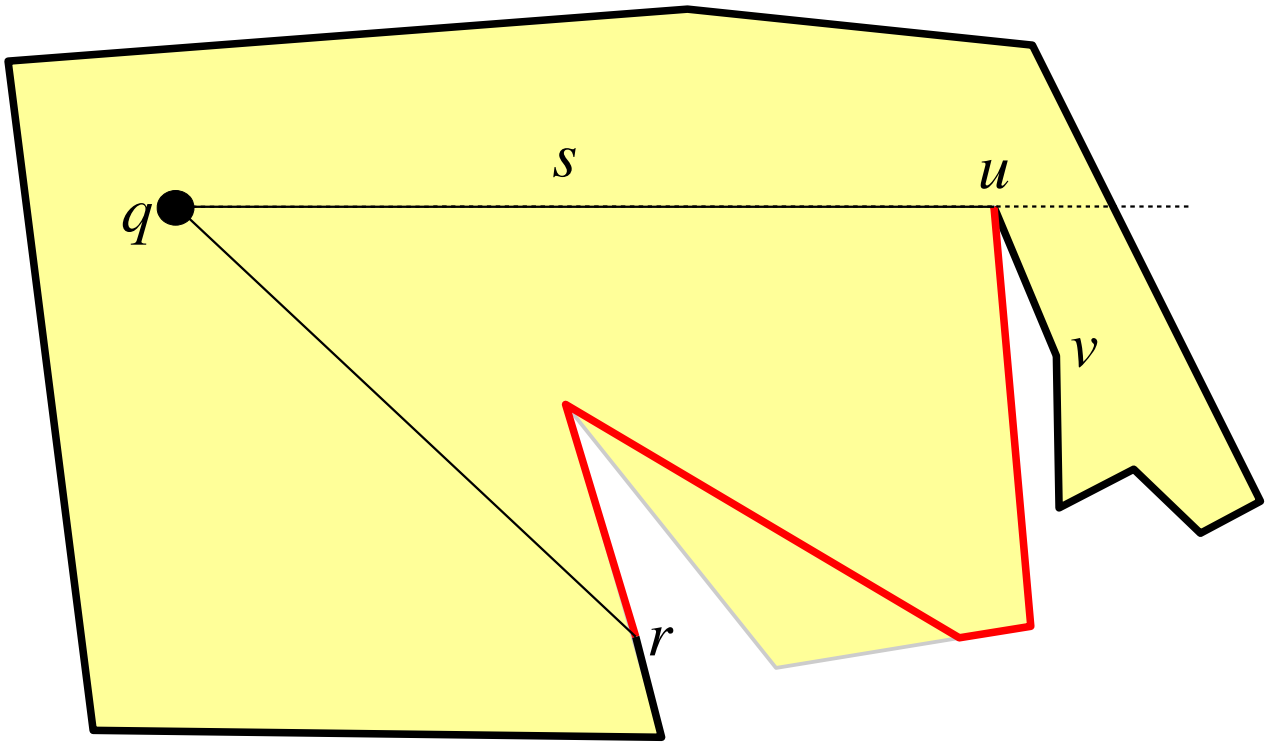
1.1. Unterfall: v liegt oberhalb des Strahls s.



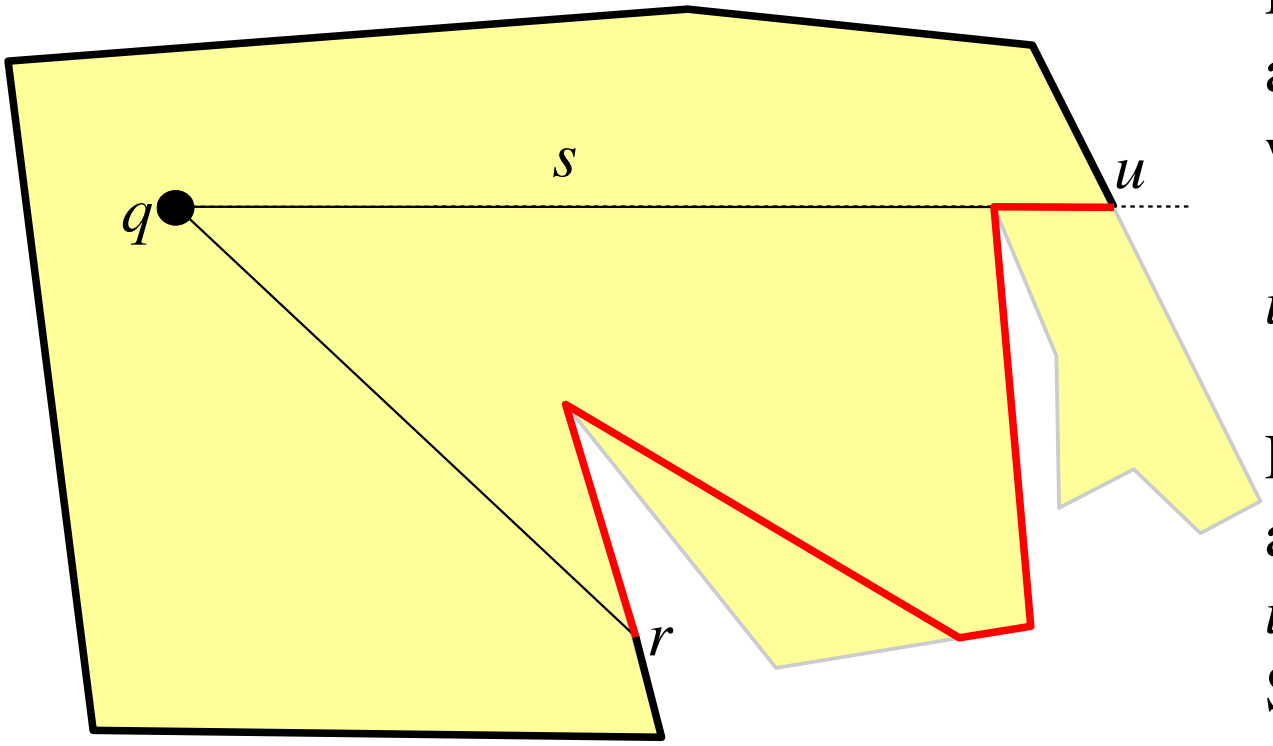
Die Kante zwischen u und v wird auf den Stapel gelegt.

u wird aktualisiert.

1.2. Unterfall: *v* liegt unterhalb von *s* und der Rand macht bei *u* eine Rechtsdrehung.



1.2. Unterfall: v liegt unterhalb von s und der Rand macht bei u eine Rechtsdrehung.



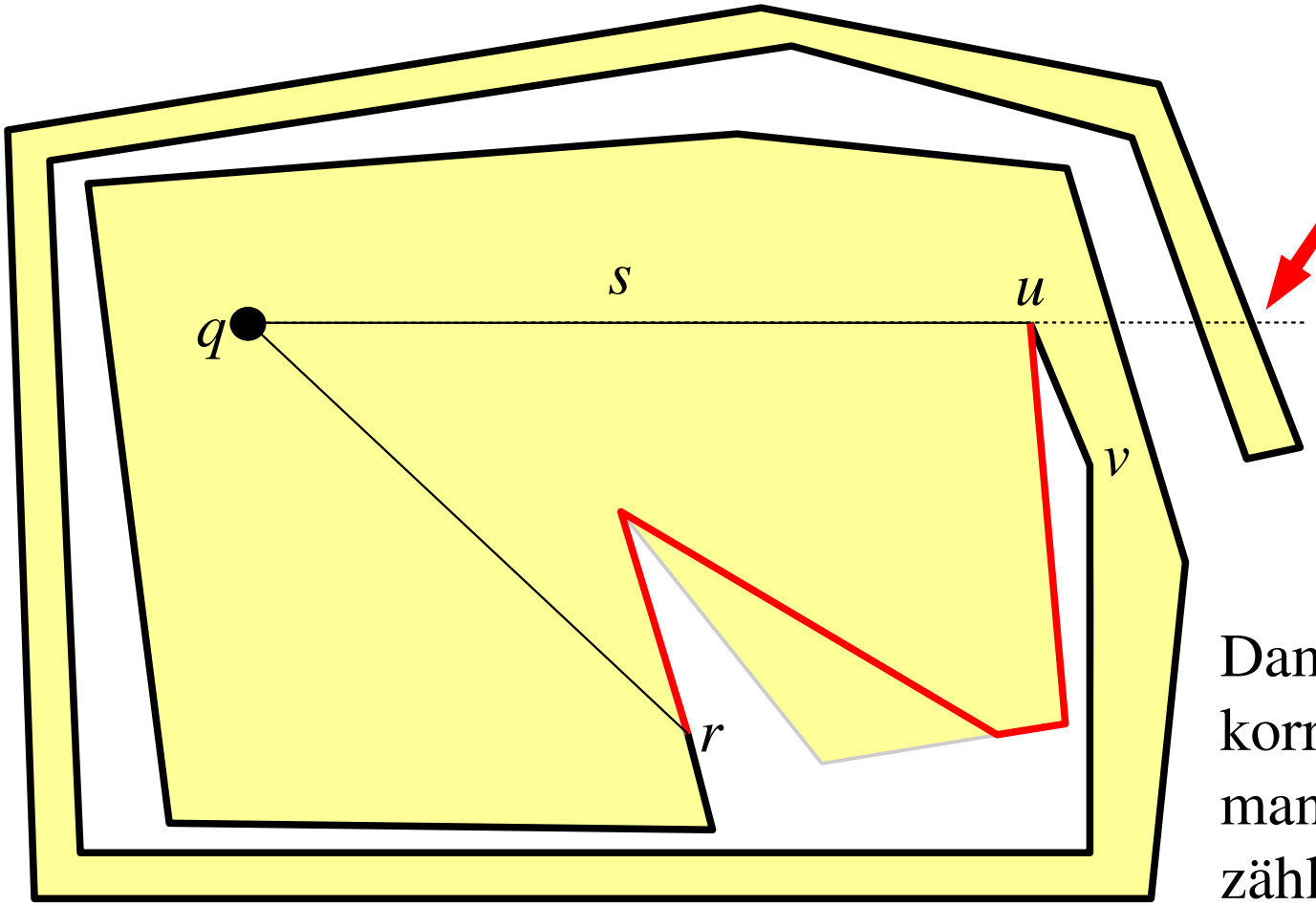
Wir verfolgen den Rand von P von v aus, bis er hinter u wieder auftaucht.

u wird aktualisiert.

Die Strecke vom alten zum neuen u wird auf den Stapel gelegt.

ACHTUNG FALLSTRICK!

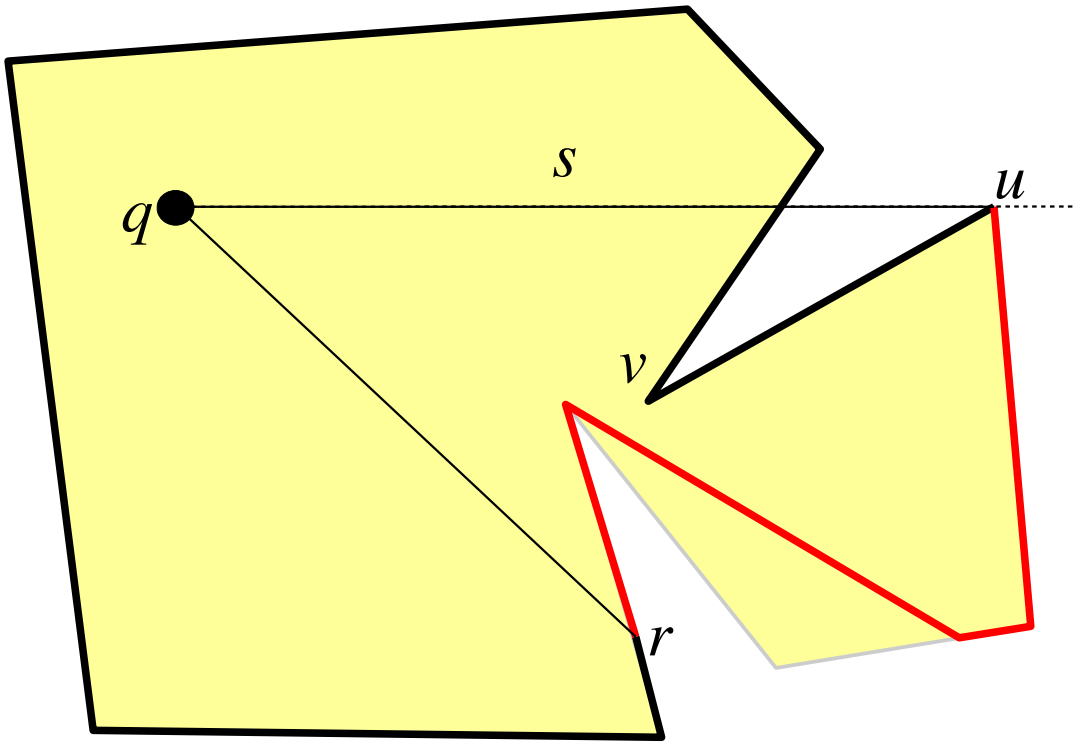
1.2. Unterfall: v liegt unterhalb von s und der Rand macht bei u eine Rechtsdrehung.



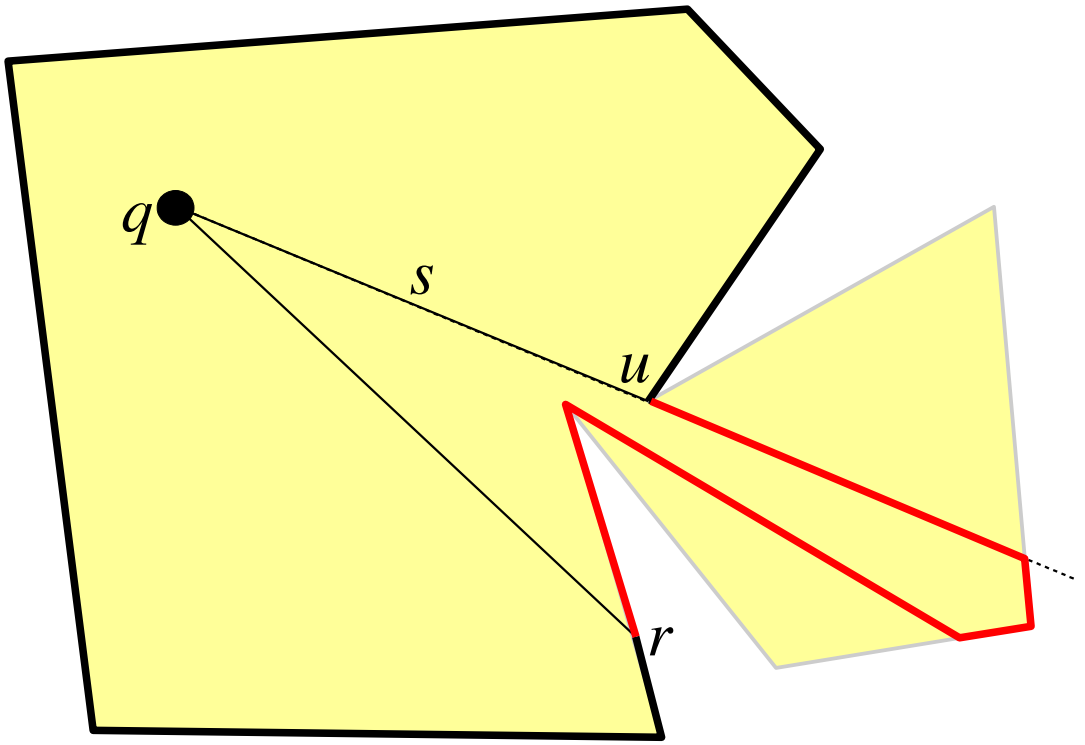
Wäre falsches neues u!

Damit das Verfahren korrekt arbeitet, muss man einen Winkelzähler mitführen.

1.3. Unterfall: v liegt unterhalb von s und der Rand macht bei u eine Linksdrehung.



1.3.1. Unterfall: Die Strecke zwischen u und v schneidet kein Stapelelement.

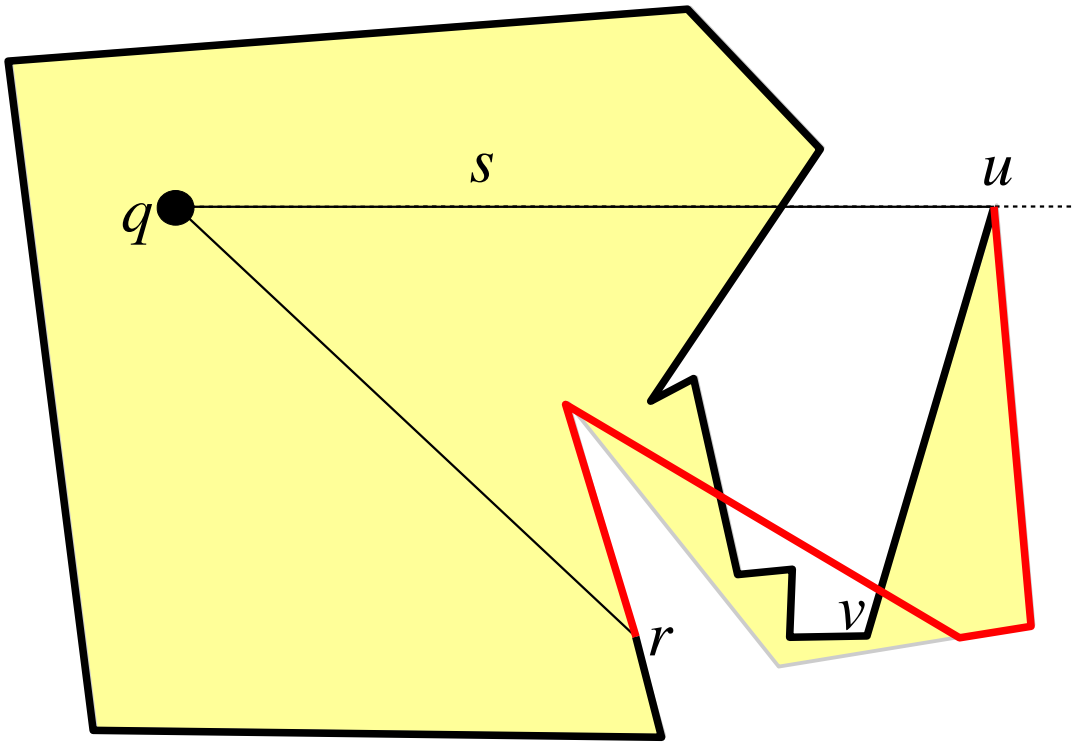


Wir nehmen alles vom Stapel herunter, was von der Kante zwischen u und v verdeckt wird.

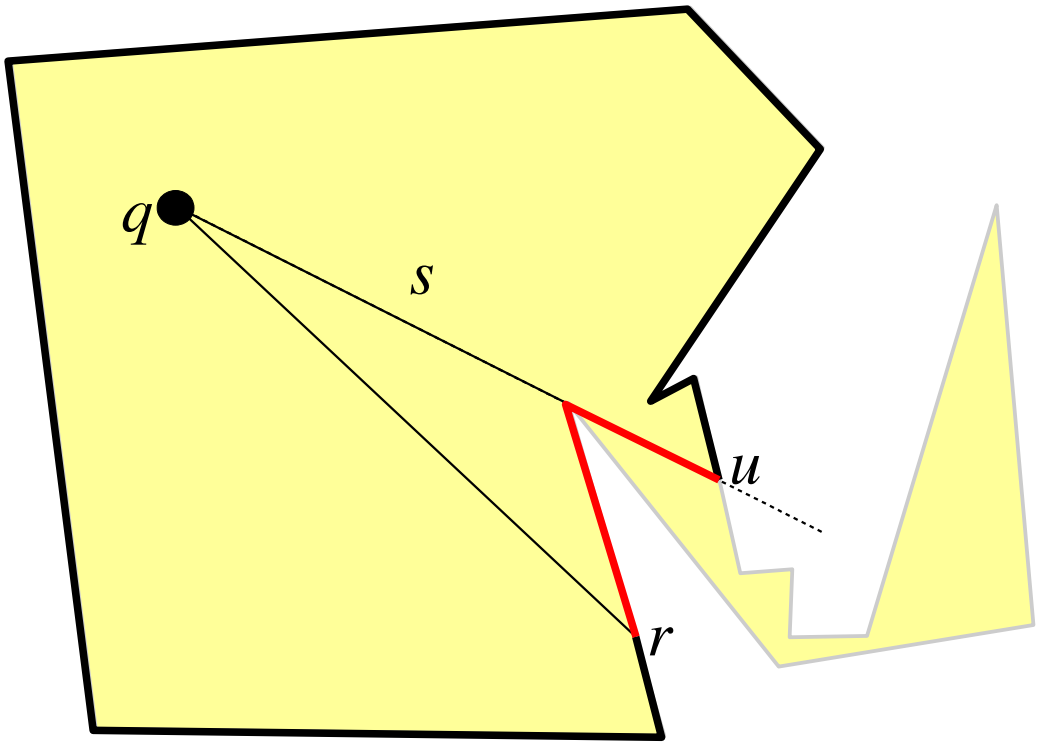
u wird aktualisiert.

Der Stapel wird entsprechend aktualisiert.

1.3.2. Unterfall: Die Strecke zwischen u und v schneidet ein Stapelelement.



1.3.2. Unterfall: Die Strecke zwischen u und v schneidet ein Stapelelement.

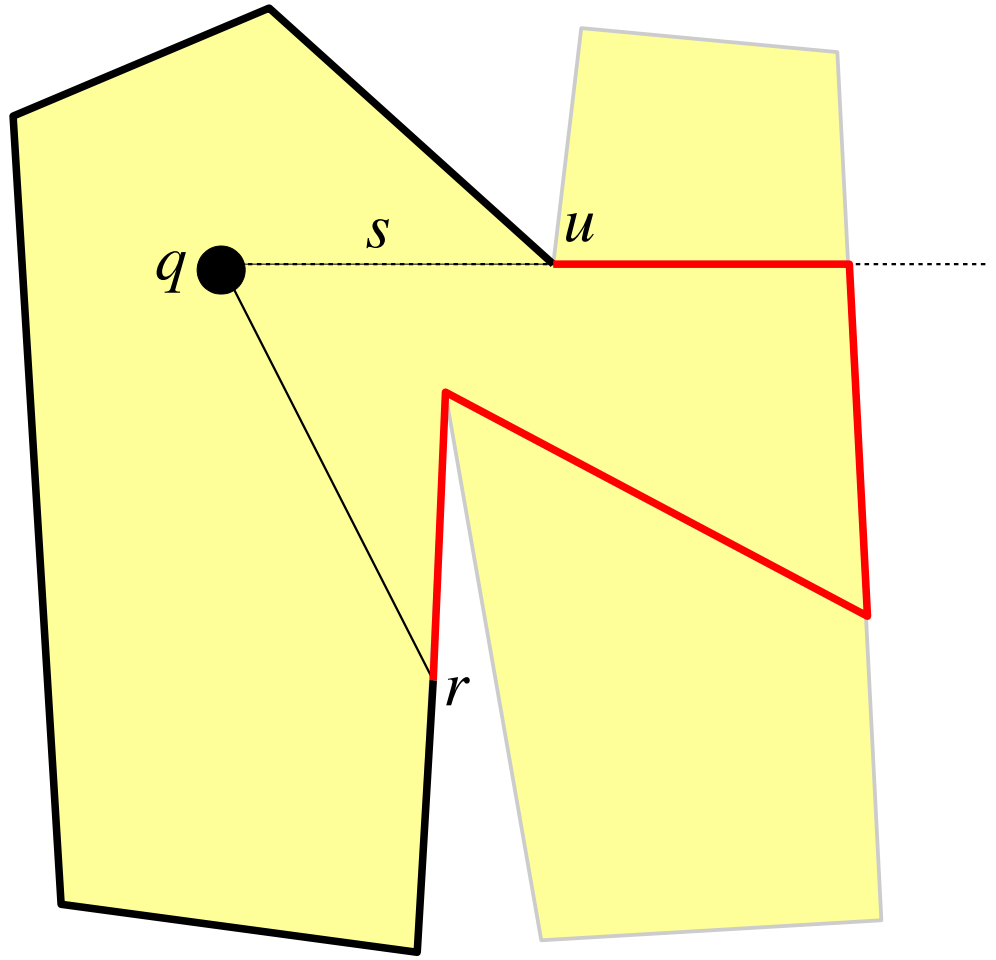


Wir nehmen alles vom Stapel herunter, was von der Kante zwischen u und v verdeckt wird.

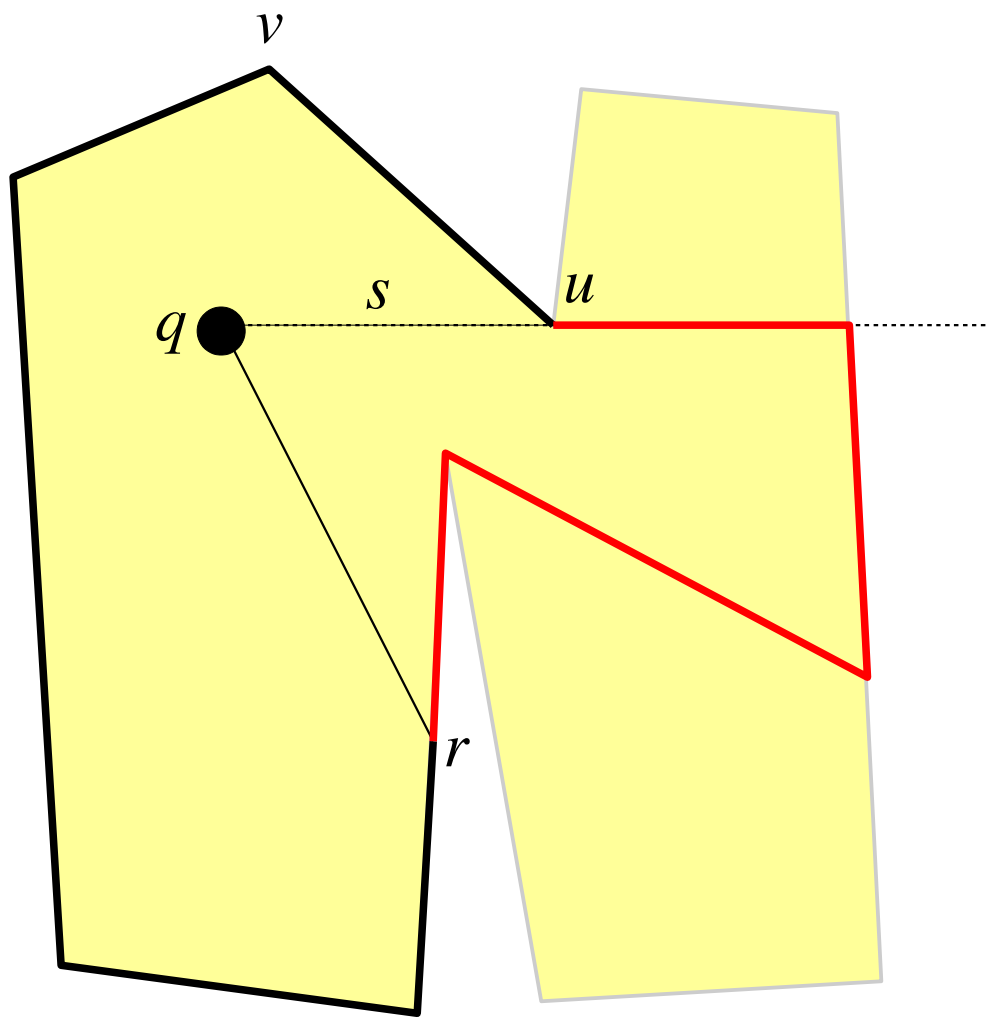
Wir verfolgen den Rand von v aus vorwärts bis er wieder auftaucht.

u und der Stapel werden aktualisiert.

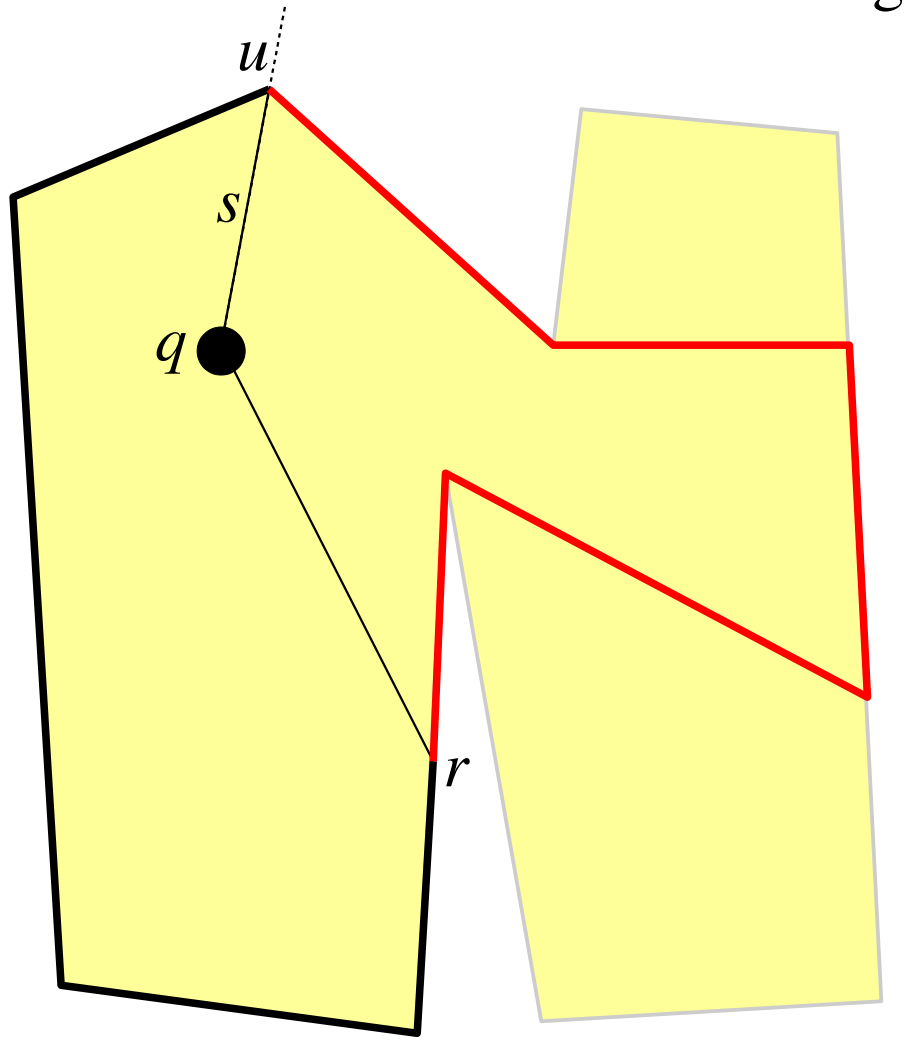
2. Fall: Der Rand von P trifft bei u von oben auf den Strahl s.



2.1. Unterfall: v liegt oberhalb von s und der Rand macht bei u eine Rechtsdrehung.



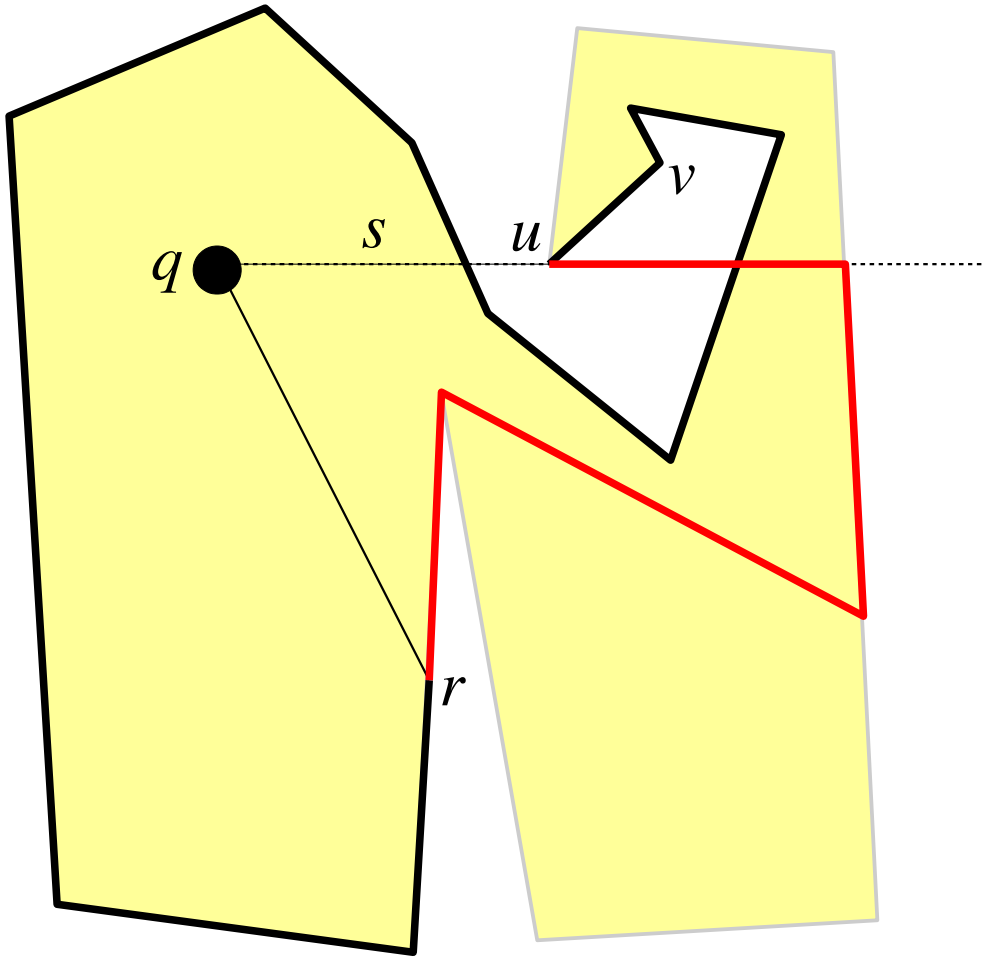
2.1. Unterfall: *v* liegt oberhalb von *s* und der Rand macht bei *u* eine Rechtsdrehung.



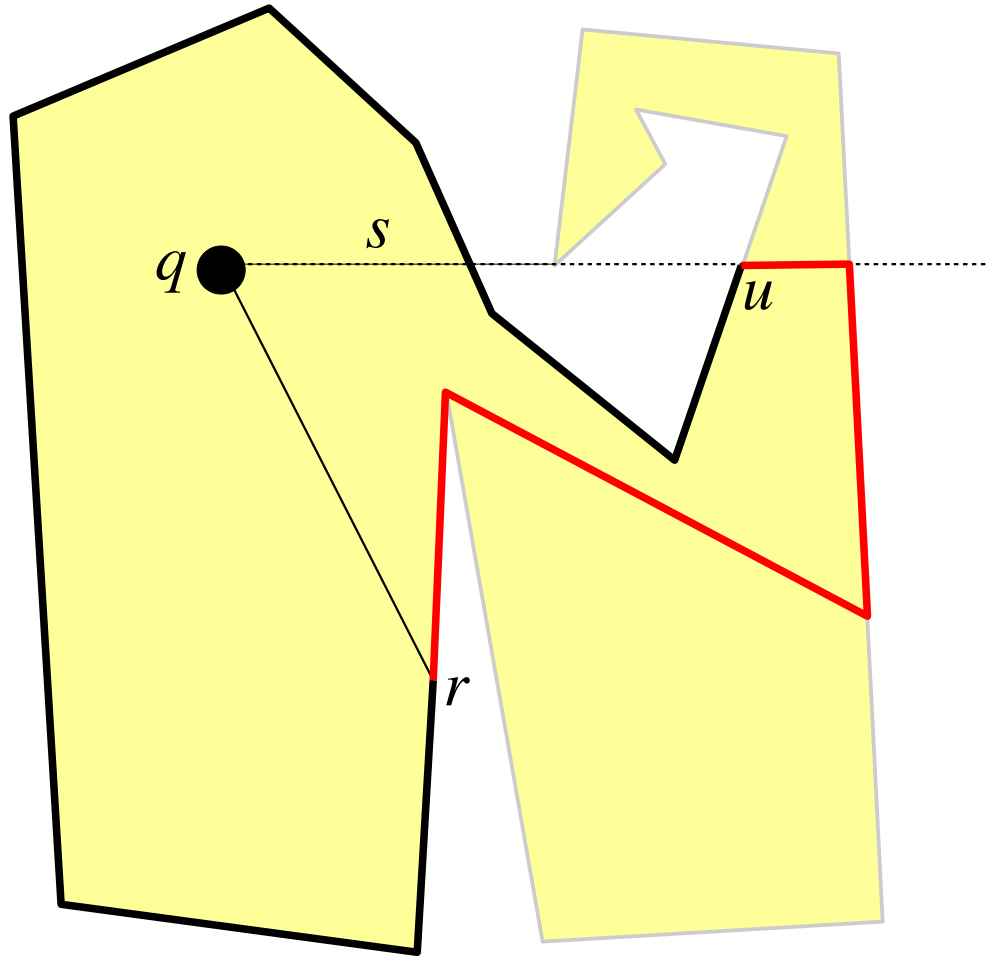
Die Kante zwischen *u* und *v* wird auf den Stapel gelegt.

u wird aktualisiert.

2.2. Unterfall: v liegt oberhalb von s und der Rand macht bei u eine Linksdrehung.



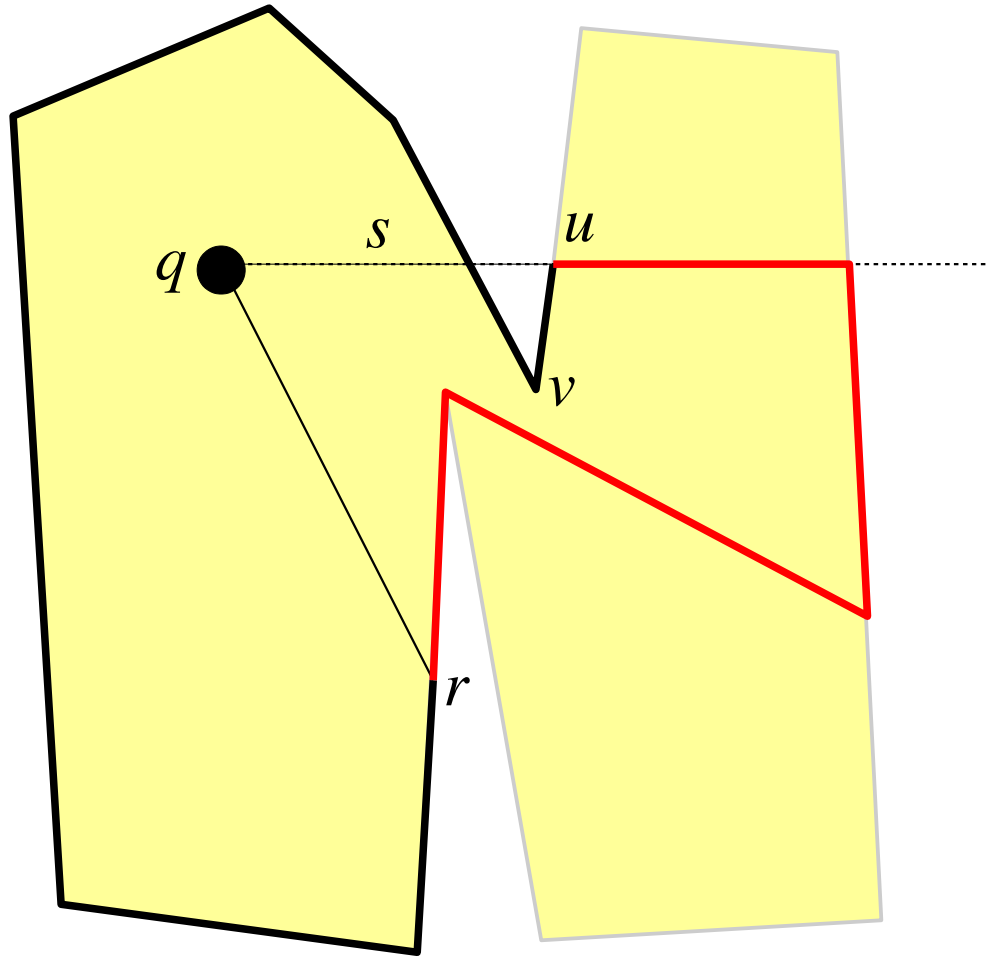
2.2. Unterfall: v liegt oberhalb von s und der Rand macht bei u eine Linksdrehung.



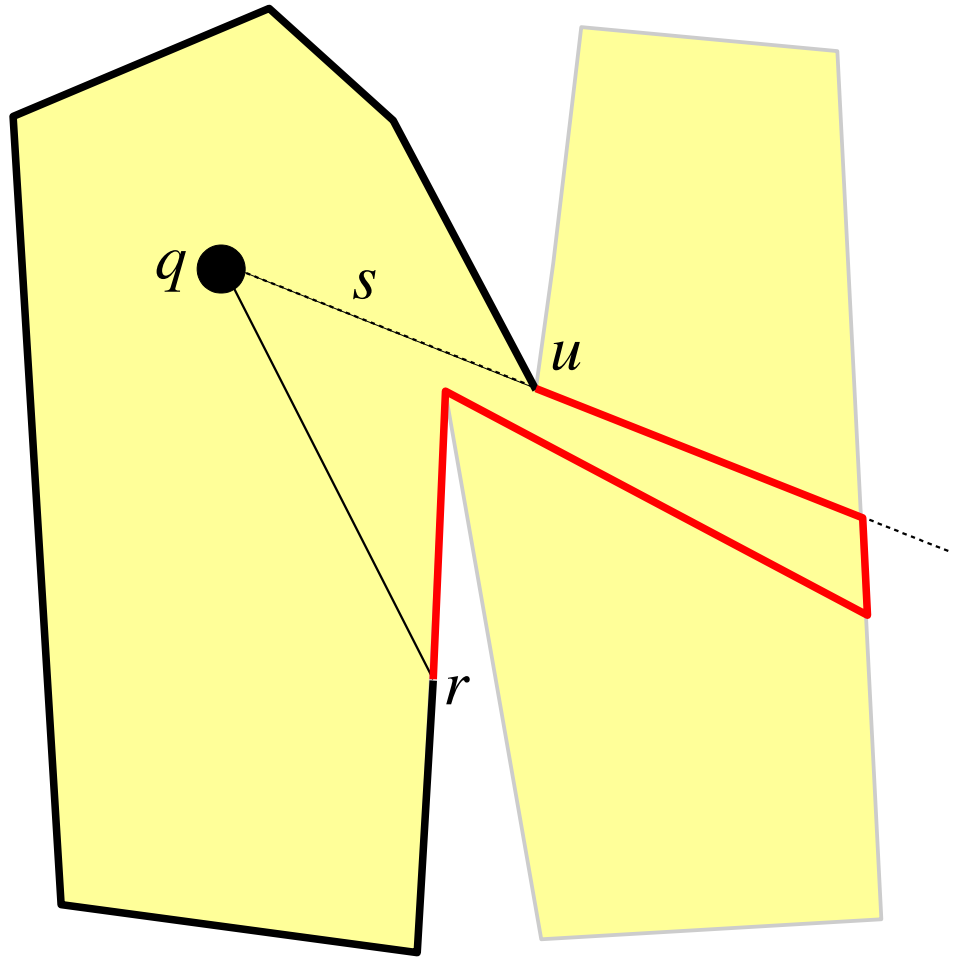
Wir verfolgen den Rand von P von v aus, bis er hinter u wieder auftaucht.

u und der Stapel werden aktualisiert.

2.3. Unterfall: v liegt unterhalb von s.



2.3.1. Unterfall: Die Strecke zwischen u und v schneidet kein Stapelelement.

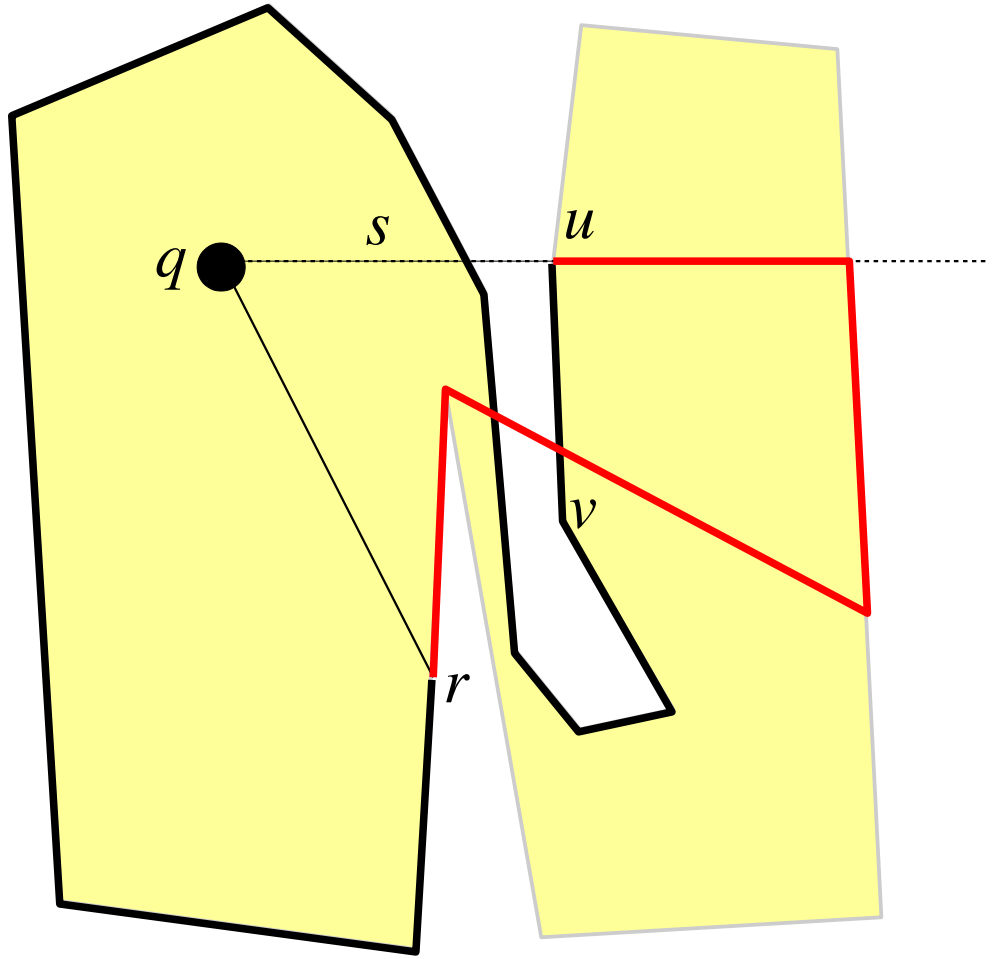


Wir nehmen alles vom Stapel herunter, was von der Kante zwischen u und v verdeckt wird.

u und der Stapel werden aktualisiert.

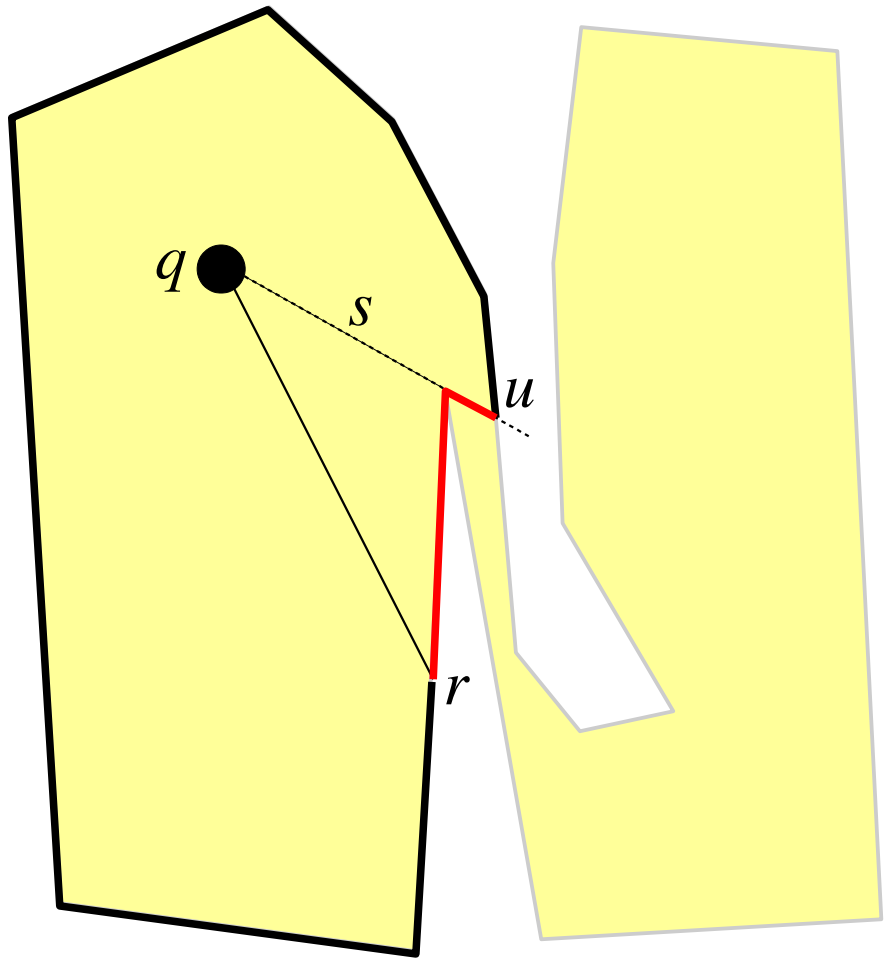
2.3.2. Unterfall:

Die Strecke zwischen u und v schneidet ein Stapelelement.



2.3.2. Unterfall:

Die Strecke zwischen u und v schneidet ein Stapelelement.

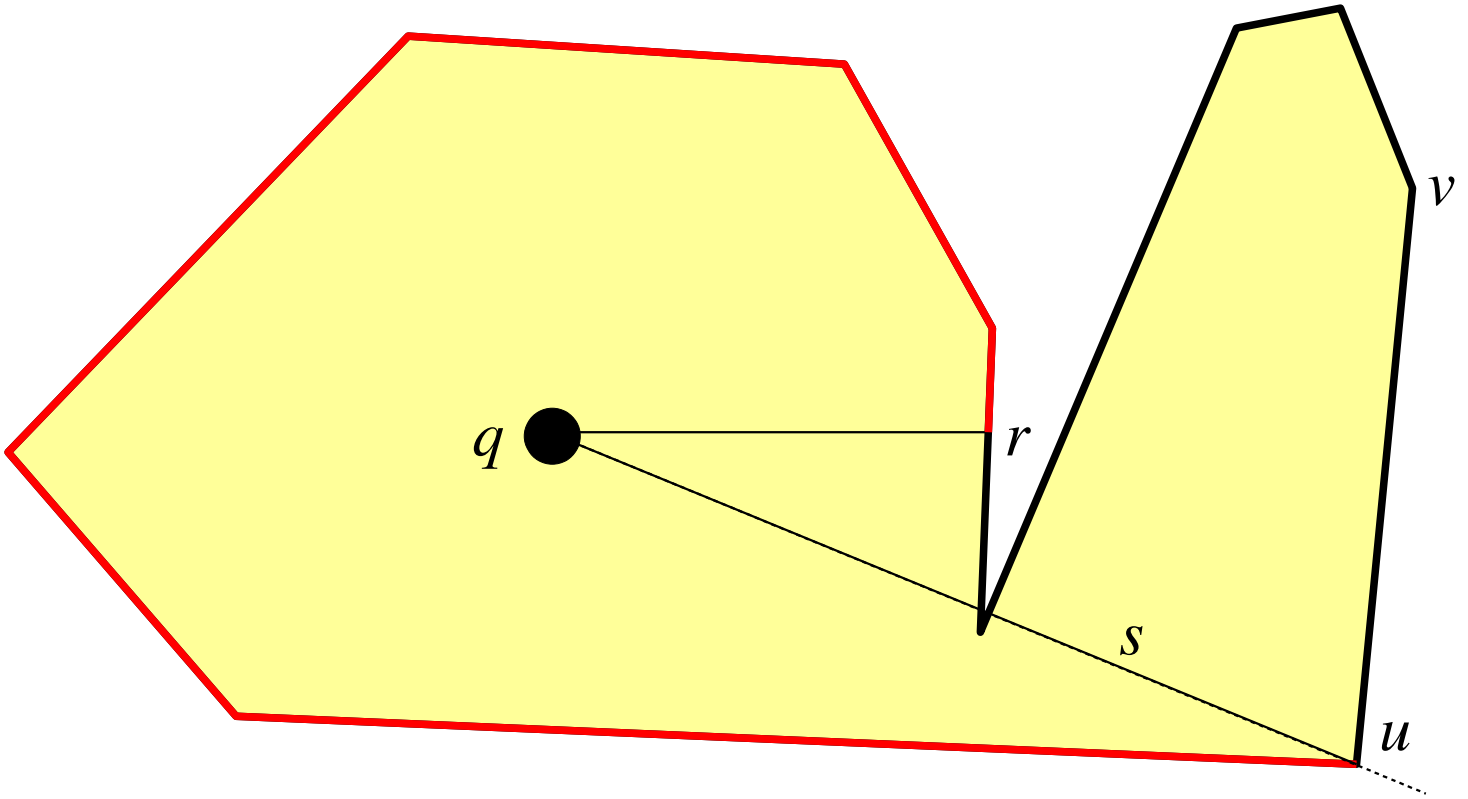


Wir nehmen alles vom Stapel herunter, was von der Kante zwischen u und v verdeckt wird.

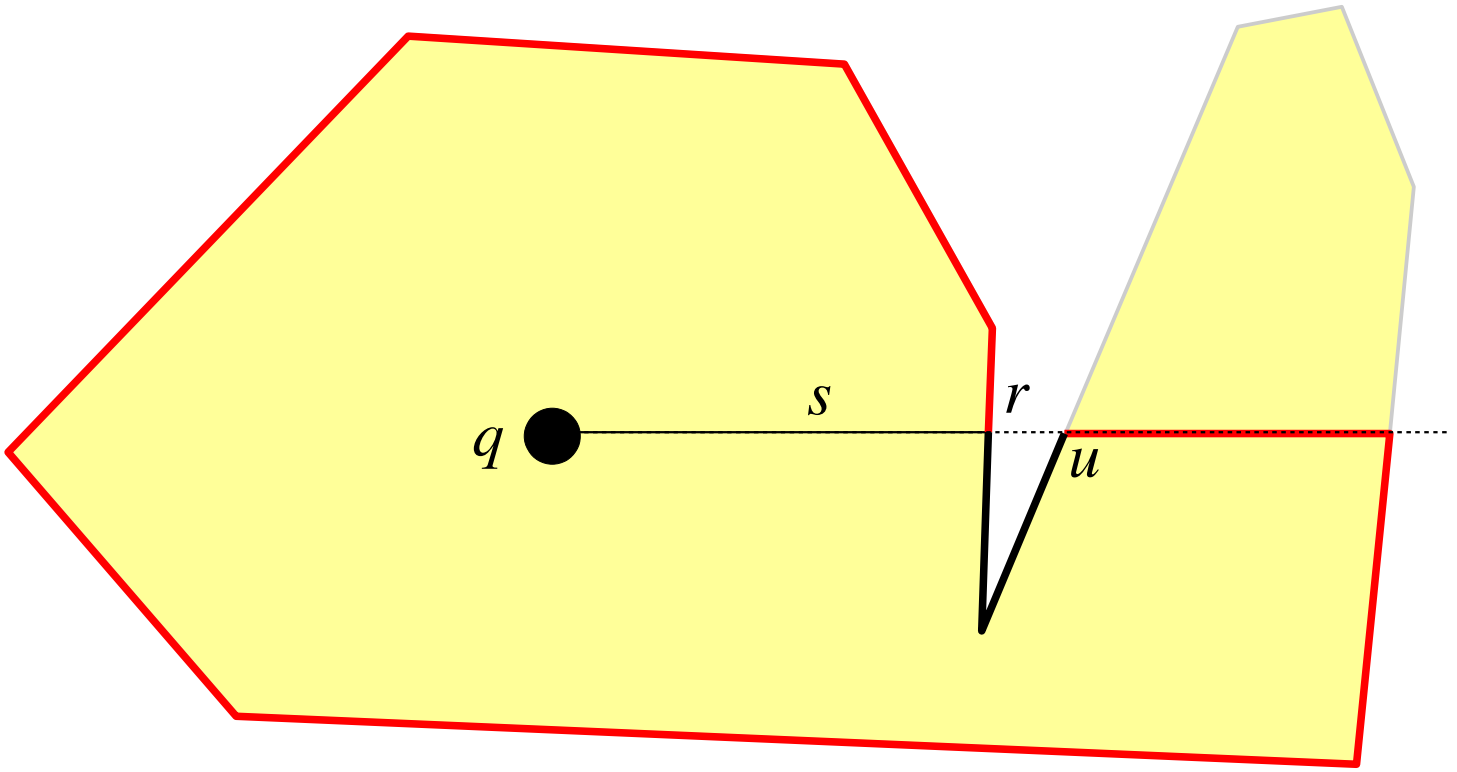
Wir verfolgen den Rand bis er wieder auftaucht.

u und der Stapel werden aktualisiert.

Im 1.1. und im 2.1. Unterfall gibt es noch eine kleine Schwierigkeit:



Aber auch hier warten wir einfach, bis der Rand wieder auftaucht.
Dabei müssen wir einen Winkelzähler mitführen.



Zusammenfassung:

Es gibt $O(n)$ Ereignispunkte.

Die amortisierten Kosten pro Ereignispunkt sind in $O(1)$.

Laufzeit: $O(n)$