Aufgabe 3: Nasser Hund
Analytische Geometrie Lösung

Team-ID: 00001

Team-Name: Was?

Bearbeiter dieser Aufgabe:
Mathieu de Borman

28. November 2025

Inhaltsverzeichnis

1 Lösungsidee

Seien $P_1, P_2, ..., P_k$ die Wegstrecken und $S_1, S_2, ..., S_s$ die Seen, sodass der See S_i aus n_i Seitenstrecken S_{ij} mit $1 \leq i \leq s$ und $1 \leq j \leq n_i$ bestehen. Dann ist die längstmögliche Leinenlänge gleich dem Minimum aller Abstände zwischen aller P_l und aller S_{ij}.
Sei $P_l = AB, S_{ij} = CD$ und $d(P, S)$ der Abstand zwischen Punkt P und Strecke S. Dann ist der Abstand zwischen P_l und S_{ij} gleich

$$\min(d(A, S_{ij}), d(B, S_{ij}), d(C, P_l), d(D, P_l)).$$

Sei P ein Punkt und $S = AB$ eine Strecke. Sei

$$t = \frac{(\vec{P} - \vec{A}) \cdot (\vec{B} - \vec{A})}{| \vec{B} - \vec{A} |^2} = (\vec{P} - \vec{A}) \cdot \frac{\vec{B} - \vec{A}}{| \vec{B} - \vec{A} |} \cdot \frac{1}{| \vec{B} - \vec{A} |},$$

also dem Verhältnis der Länge der orthogonalen Projektion von $\overrightarrow{AP}$ in Richtung $\overrightarrow{AB}$ (Skalarprodukt von $\overrightarrow{AP}$ mit Einheitsvektor von $\overrightarrow{AB}$) zur Länge von $\overrightarrow{AB}$. Dann ist der Abstand von P und S für $t \geq 0$ gleich $| \vec{P} - \vec{A} |$, für $t \geq 1$ gleich $| \vec{P} - \vec{B} |$ und für $0 < t < 1$ gleich $| \vec{P} - (\vec{A} + t \cdot (\vec{B} - \vec{A})) |$.
Zusätzlich muss überprüft werden, ob die Pfade in einem oder mehreren Seen liegen. Hierfür muss sichergestellt werden, dass sich einerseits keine der Wege sich auch mit der Seeseiten kreuzt, und zusätzlich, ob eine oder beide Eckpunkte eines Weges innerhalb einer der Seen liegen.
Ersteres erfolgt anhand des Vergleichs der Orientierung (dem Vorzeichen des Vektorprodukt

im Zweidimensionalem) der beiden Endpunkten der einen Strecke und jeweils einem Endpunkt der anderen Strecke[1], wobei jedoch der kollineare Fall ignoriert werden kann, da bei Parallelität entweder die Strecken sich nicht schneiden oder ein Abstand und damit eine maximale Leinenlänge von 0 bestimmt wird, und letzteres anhand des sog. Ray-Casting-Algorithmus[2], wobei für den Strahl eine Parallele der x-Achse in positiver X-Richtung verwendet wird. Hierfür wird für jeden Endpunkt der Wegstrecken überprüft, ob er innerhalb eines Sees liegt, indem für alle Seen gezählt wird, wie oft sich der Strahl ausgehend von dem Endpunkt mit den Kanten des Sees schneidet. Ist diese Zahl ungerade, so liegt der Punkt innerhalb des Sees, andernfalls nicht. Damit ein Strahl ausgehend vom Endpunkt $P = (x_0, y_0)$ die Strecke zwischen $A = (x_a, y_a)$ und $B = (x_b, y_b)$ schneidet, muss einerseits

$$y_a > y_0 \oplus y_b > y_0$$

(d.h. y_0 muss zwischen y_a und y_b liegen. Es wird zweimal $>$ verwendet, um Doppelzählungen zu vermeiden, wenn der Strahl durch einen Eckpunkt des Sees geht, wodurch aus der Fall $y_a = y_b = y_0$ sowie wenn der Strahl nur durch den Eckpunkt eines konvexen Polygons geht (dieser Fall kann ebenso ignoriert werden, da statt eines Fehlers einfach eine maximale Leinenlänge von 0 ausgegeben wird), abgedeckt wird), und andererseits auch $x_0 < x$ (der Fall $x_0 = x$ wird bereits durch die Orientierung abgefangen) mit $\frac{y_b - y_a}{x_b - x_a} \cdot (x - x_a) + y_a = y_0$ (Punkt-Steigungsform), also

$$((x_0 - x_a) \cdot (y_b - y_a) - (x_b - x_a) \cdot (y_0 - y_a)) \cdot (y_b - y_a) < 0$$

(die erneute Multiplikation mit $(y_b - y_a)$ ist erforderlich, da während der Umformung zum Entfernen der Division (ineffizienter und ggf. Rundungsfehler) bereits einmal damit multipliziert werden muss, der Term jedoch negativ sein kann. Durch erneute Multiplikation wird dies wieder aufgehoben) gelten.

2 Umsetzung

Ursprünglich in C implementiert, danach Port auf JavaScript für bessere Visualisierung. Die Implementierung entspricht im Wesentlichen der Lösungsidee. Die Orientierung wird mithilfe der Funktion `o` berechnet, der Abstand zwischen mehreren Punkten mithilfe von `d`. Der Ray-Casting-Algorithmus ist so implementiert, dass für alle Seen für alle Endpunkte der Wege ein mit 0 initialisiertes Bit für jedes der Kanten des Sees durch ein XOR geflippt wird, sofern der Strahl diese Kante schneidet. Dadurch kann dann, nachdem eine Schleife alle Kanten durchlaufen hat, ein Fehler ausgegeben werden, wenn das Bit 1 ist, da es dann eine ungerade Anzahl geflippt wurde.

3 Laufzeitanalyse

Sei k die Anzahl der Wege und n_s die Gesamtanzahl aller Eckpunkte der Seen. Dann ist die Laufzeit $\mathcal{O}(k \cdot n_s)$, da sowohl `o` als auch `d` in $\mathcal{O}(1)$ laufen und die Eingabe, die Schnittüberprüfung und der Ray-Casting-Algorithmus allesamt in $\mathcal{O}(k \cdot n_s)$ laufen (siehe for-Schleifen). Da jeder Weg mit jedem See überprüft werden muss, sollte dies auch die optimalste Laufzeit sein.

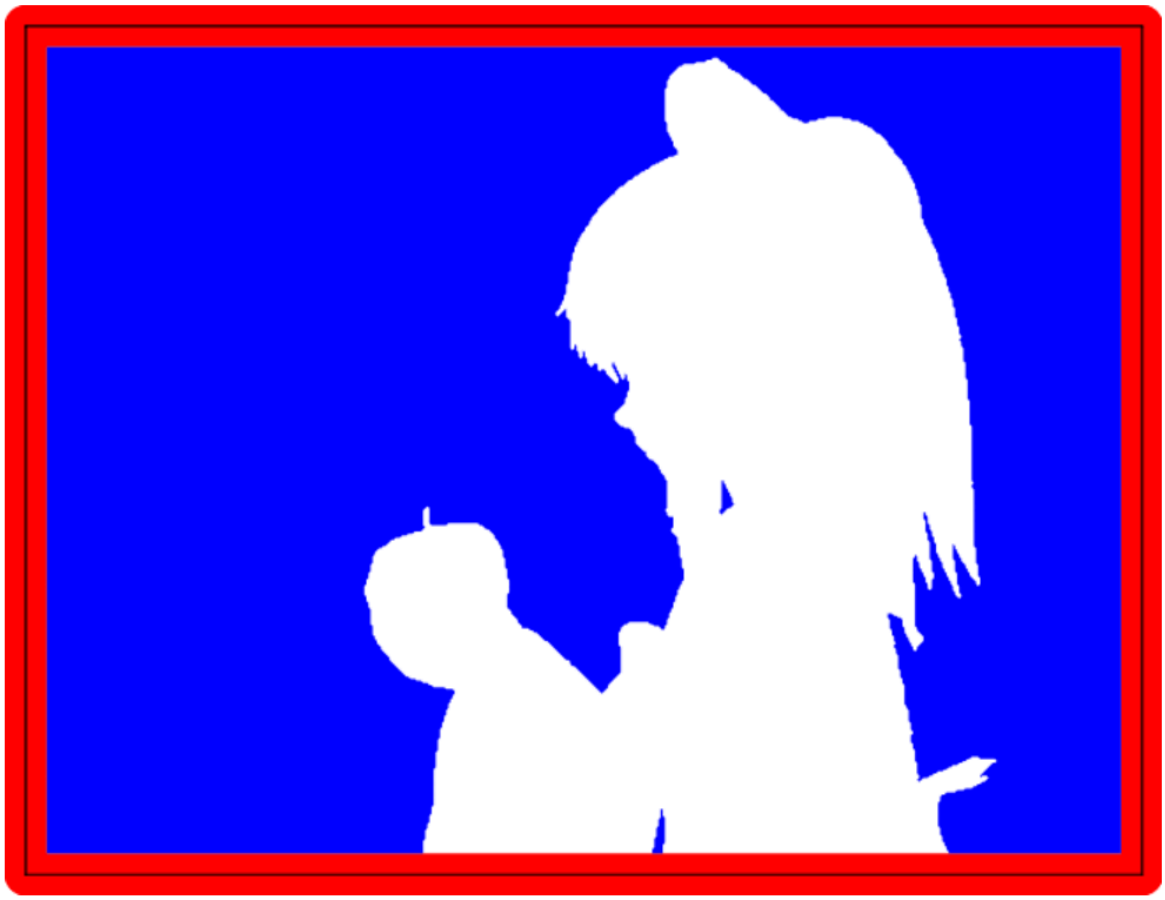
[1] https://www.geeksforgeeks.org/dsa/check-if-two-given-line-segments-intersect/
[2] https://en.wikipedia.org/wiki/Point_in_polygon#Ray_casting_algorithm

4 Werkzeuge

In dieser Arbeit wurden keine KIs, ILP-Solver oder LLMs verwendet.
Zur Erstellung weiterer Beispiele wurde GeoGebra benutzt.

5 Video

Diese Großartige Aufgabe bietet großen kreativen Spielraum. Polygone haben nämlich die tolle eigenschaft, jegliche Formen annehmen zu können. So auch das Bild dieser Tollen Dame.



Kennern wird nun vermutlich auffallen, dass diese Person eigentlich aus einem Video stammt. Also haben wir uns darum gekümmert dieses Video nachzustellen. Sollten sie zurzeit keine Internetverbindung haben, so müssen sie nicht darauf verzichten. Es besteht nämlich die möglichkeit das Video durch ausführen von badapple.txt durch die Lösung anzuschauen.

6 Beispiele

Alle Lösungen wurden mithilfe des gleichen Programms gelöst. Dabei wurde für alle Aufgaben eine zusätzliche Grafik erstellt, die die Korrektheit unseres Programms stützen sollte. Dabei stehen die schwarzen Linien für die Wege bzw. Kanten, die in der Aufgabenstellung vorgegeben sind. Die blaue Fläche steht für die Seen, die in der Aufgabenstellung ebenfalls vorgegeben sind. Die roten Flächen, die die schwarzen Linien umrahmen stehen für die Flächen, die der Hund unter der gegebenen Leinenlänge frei zur Verfügung hat. Für alle schwarzen Linien wird daher ein roter Kreis erstellt, dessen Radius die maximale Distanz der Leine ist. Beim Betrachten der Grafiken erkennt man, dass sich die roten Flächen nie mit der blauen Fläche überschneiden. Aus diesem Grund sind die Lösungen quantitativ gesehen korrekt.

6.1 Standardbeispiele

Zunächst einmal finden Sie in unserer Tabelle alle Angaben sowie Lösungen für die gegebenen Beispielaufgaben. Zusätzlich wurde noch eine Grafik für alle Aufgaben erstellt, die unten zu finden ist.

Dateiname	Ausgabe
hund01.txt	Runtime: 0.035s Min dis: 4.036036763977875
hund02.txt	Runtime: 0.012s Min dis: 4.20987849267374
hund03.txt	Runtime: 0.008s Min dis: 2.23606797749979
hund04.txt	Runtime: 0.009s Min dis: 1.6712580435934667
hund05.txt	Runtime: 0.355s Min dis: 50.320350312988666
hund06.txt	Runtime: 0.249s Min dis: 199.5725023626895

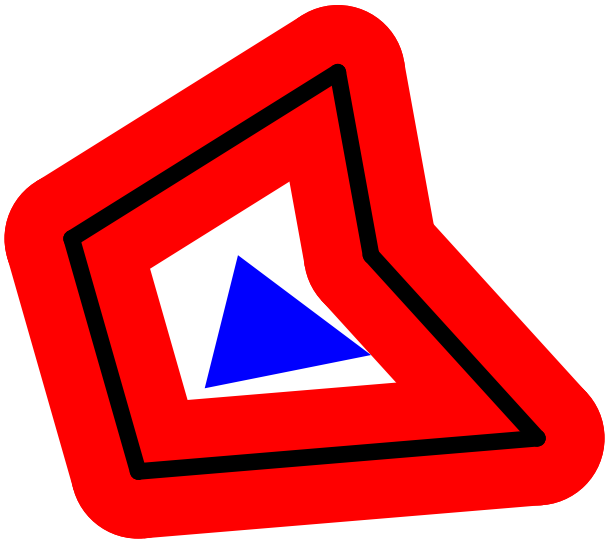


Abbildung 1: hund01.txt

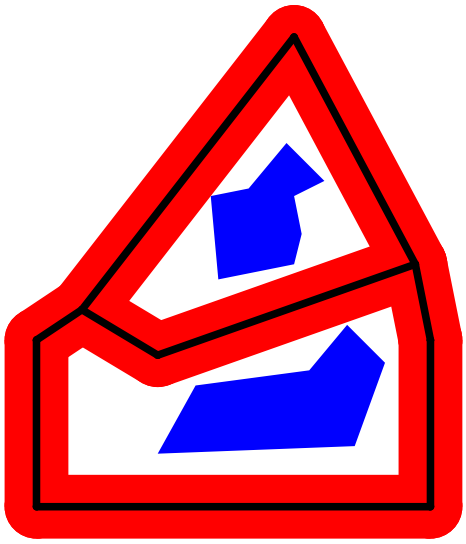


Abbildung 2: hund02.txt

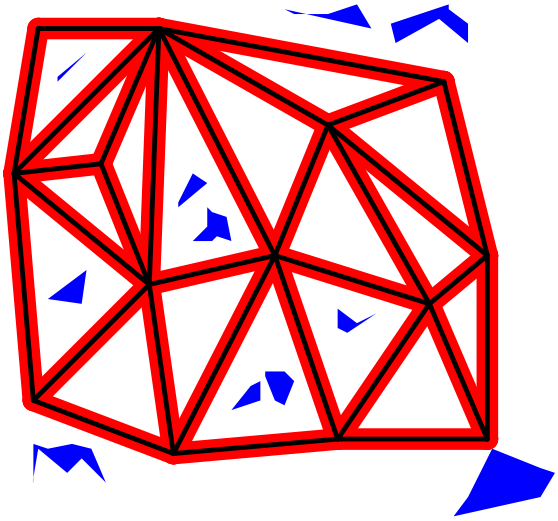


Abbildung 3: hund03.txt

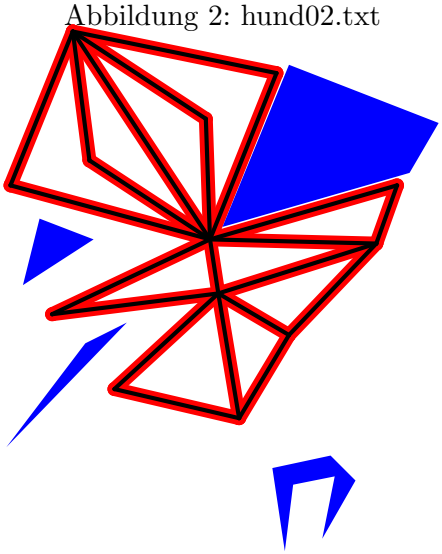


Abbildung 4: hund04.txt

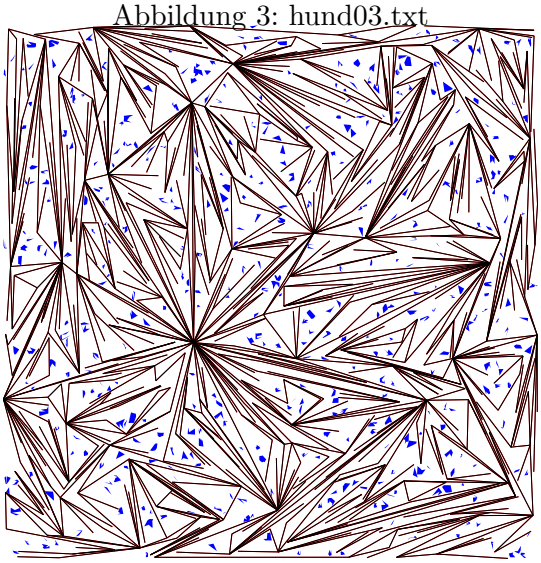


Abbildung 5: hund05.txt

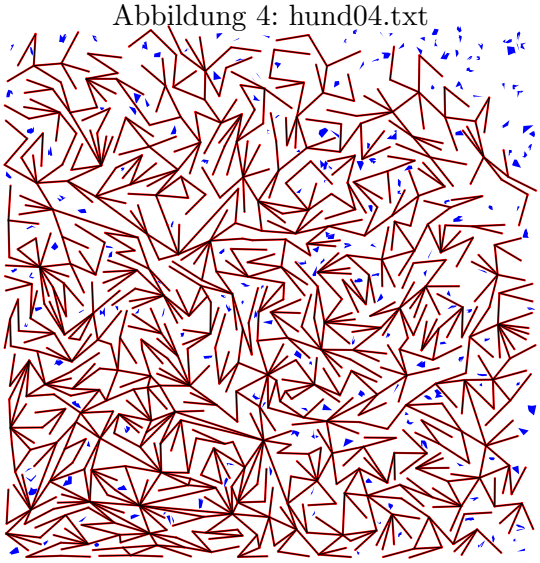
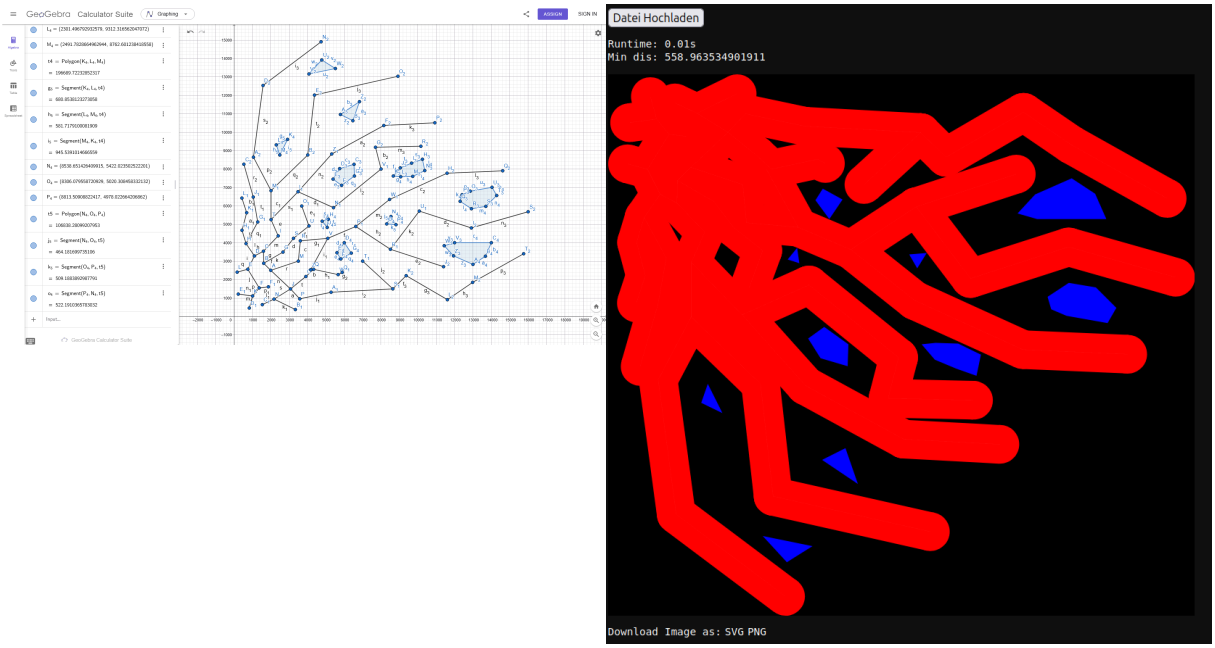


Abbildung 6: hund06.txt

6.2 Zusatzbeispiele

Um die Leistungsfähigkeit unseres Programms darzustellen, haben wir uns für weitere Beispiele entschieden, die die Leistungsfähigkeit sowie die Korrektheit des Programms

darstellen sollen. Alle Zusatzbeispiele sind ebenfalls im Ordner zu finden. Einige Aufgaben wurden dabei mithilfe von GeoGebra erstellt und zur Aufgabe exportiert. Alle Aufgaben wurden dabei von der Originalposition an der x-Achse gespiegelt.



6.3 Renderfähigkeit und Kunst

Wir haben uns dazu entschieden, uns noch näher an das Limit an zu nähern und deshalb weitere Beispiele generiert, die die Leistungsfähigkeit des Programms darstellt.

Abbildung 7: Bad Apple

Abbildung 8: Hat Kid

Abbildung 9: Fortnite Season 1 Weltkarte

Abbildung 10: Rick Astley

Abbildung 11: Rick Astley 2

script.js JavaScript

```
161 function o(a, b, c) {
162  let t = (b[1] - a[1]) * (c[0] - b[0]) - (b[0] - a[0]) * (c[1] - b[1])
163  return t == 0 ? 0 : (t > 0 ? 1 : 2)
164 }
```

```
function d(a, b, c) {
 let dabx = b[0] - a[0]
 let daby = b[1] - a[1]
 let dacx = c[0] - a[0]
 let dacy = c[1] - a[1]
 let t = (dabx * dacx + daby * dacy) / (dabx * dabx + daby * daby)
 t = t > 1 ? 1 : (t < 0 ? 0 : t)
 let dx = dacx - t * dabx
 let dy = dacy - t * daby
 return (dx * dx + dy * dy) ** 0.5
}
function minDis(k, p, s, nArr, lArr) {
 let min = -1
 for (let i = 0; i < s; i++) {
  let n = nArr[i]
  let l = lArr[i]
  for (let j = 0; j < k; j++) {
   for (let m = 0; m < n; m++) if (o(p[j][0], p[j][1], l[m]) != o(p[j][0], p[j][1], l[(m + 1) % n]) && o(l[m], l[(m + 1) % n], p[j][0]) != o(l[m], l[(m + 1) % n], p[j][1])) return -2
   for (let q = 0; q < 2; q++) {
    let c = 0;
    for (let m = 0; m < n; m++) c ^= l[m][1] > p[j][q][1] ^ l[(m + 1) % n][1] > p[j][q][1] && ((p[j][q][0] - l[m][0]) * (l[(m + 1) % n][1] - l[m][1]) - (p[j][q][1] - l[m][1]) * (l[(m + 1) % n][0] - l[m][0])) * (l[(m + 1) % n][1] - l[m][1]) < 0
    if (c) return -1
   }
  }
  for (let j = 0; j < k; j++)  for (let m = 0; m < n; m++) {
   let t
   min = ((t = d(p[j][0], p[j][1], l[m])) < min || min < 0 ? t : min)
   min = ((t = d(p[j][0], p[j][1], l[(m + 1) % n])) < min || min < 0 ? t : min)
   min = ((t = d(l[m], l[(m + 1) % n], p[j][0])) < min || min < 0 ? t : min)
   min = ((t = d(l[m], l[(m + 1) % n], p[j][1])) < min || min < 0 ? t : min)
  }
 }
 return min
}
function main(k, p, s, nArr, lArr) {
 let dis = minDis(k, p, s, nArr, lArr)
 draw(p, lArr, dis)
 return dis
}
```