

# Aufgabe 1: Weniger krumme Touren

Teilnahme-ID: ?????

Bearbeiter/-in dieser Aufgabe:  
Lennart Protte

February 16, 2023

## Contents

<b>1</b>	<b>Lösungsidee</b>	<b>2</b>
<b>2</b>	<b>Umsetzung</b>	<b>3</b>
<b>3</b>	<b>Beispiele</b>	<b>4</b>
<b>4</b>	<b>Quellcode</b>	<b>14</b>

**Anleitung:** Trage oben in den Zeilen 8 bis 10 die Aufgabennummer, die Teilnahme-ID und die/den Bearbeiterin/Bearbeiter dieser Aufgabe mit Vor- und Nachnamen ein. Vergiss nicht, auch den Aufgabennamen anzupassen (statt "‘ $\text{\LaTeX}$ -Dokument'")!

Dann kannst du dieses Dokument mit deiner  $\text{\LaTeX}$ -Umgebung übersetzen.

Die Texte, die hier bereits stehen, geben ein paar Hinweise zur Einsendung. Du solltest sie aber in deiner Einsendung wieder entfernen!

## 1 Lösungsidee

Die Idee der Lösung sollte hieraus vollkommen ersichtlich werden, ohne dass auf die eigentliche Implementierung Bezug genommen wird.

## 2 Umsetzung

Hier wird kurz erläutert, wie die Lösungsidee im Programm tatsächlich umgesetzt wurde. Hier können auch Implementierungsdetails erwähnt werden.

---

```

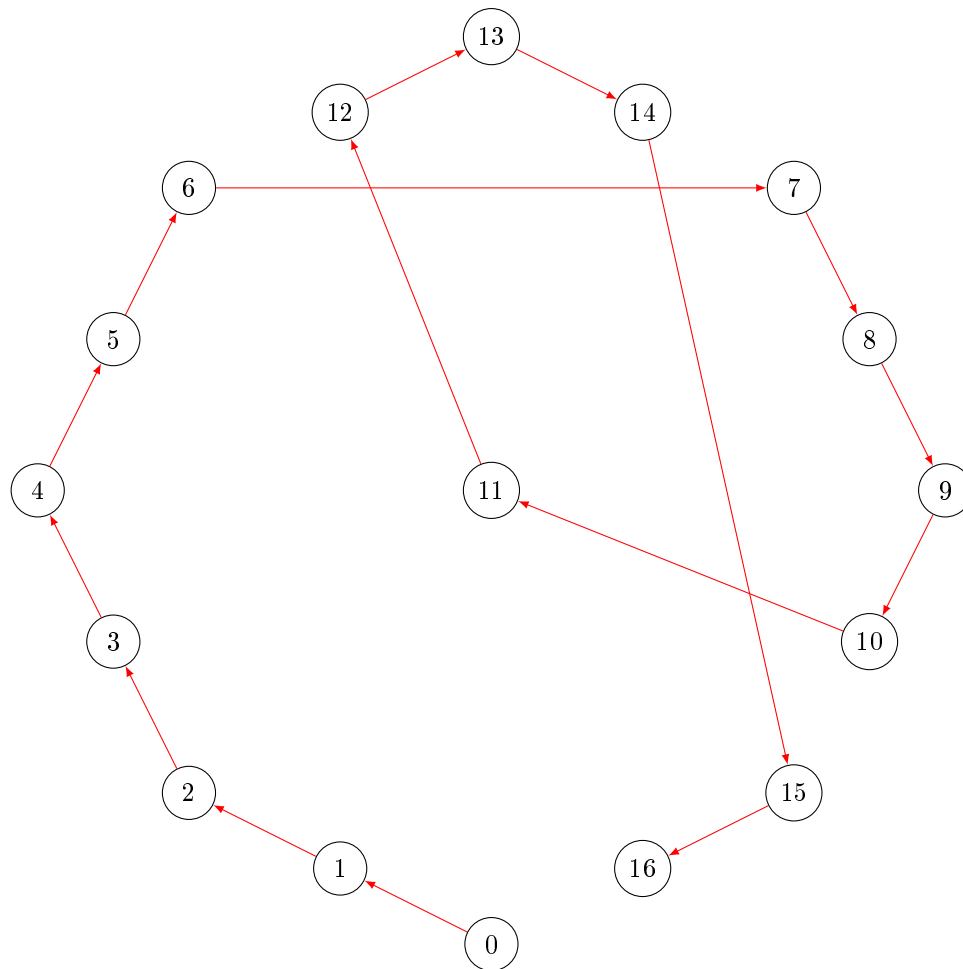
1: function CROSSANGLE(fromnode, overnode, tonode)
2:   p ← MAKEPAIR(overnode.first − fromnode.first, overnode.second − fromnode.second)
3:   q ← MAKEPAIR(tonode.first − overnode.first, tonode.second − overnode.second)
4:   angle ← acos( $\frac{p.firstq.first + p.secondq.second}{\sqrt{p.first^2 + p.second^2}\sqrt{q.first^2 + q.second^2}}$ ) * 180/π
5:   if angle > 180 then
6:     angle ← 180 − angle
7:   end if
8:   return angle
9: end function
10:
11: function SOLVE(route, coordinates)
12:   if SIZE(route) = SIZE(coordinates) then
13:     return true
14:   end if
15:   if NOT ISEMPTY(route) then
16:     p ← BACK(route)
17:     SORT(coordinates, lambda(lhs, rhs) ←  $\sqrt{(p.first - lhs.first)^2 + (p.second - lhs.second)^2} <$ 
 $\sqrt{(p.first - rhs.first)^2 + (p.second - rhs.second)^2}$ )
18:   end if
19:   for i ← 1 to SIZE(coordinates) do
20:     if coordinates[i] ∈ route then
21:       continue
22:     end if
23:     angle ← −1
24:     if SIZE(route) ≥ 2 then
25:       end if
26:     if ISEMPTY(route) OR ((NOT coordinates[i] ∈ route) AND (SIZE(route) < 2 OR angle ≥ 90
OR angle = 0)) then
27:       PUSHBACK(route, coordinates[i])
28:       if SOLVE(route, coordinates) then
29:         return true
30:       else
31:         POPBACK(route)
32:       end if
33:     end if
34:   end for
35:   return false
36: end function

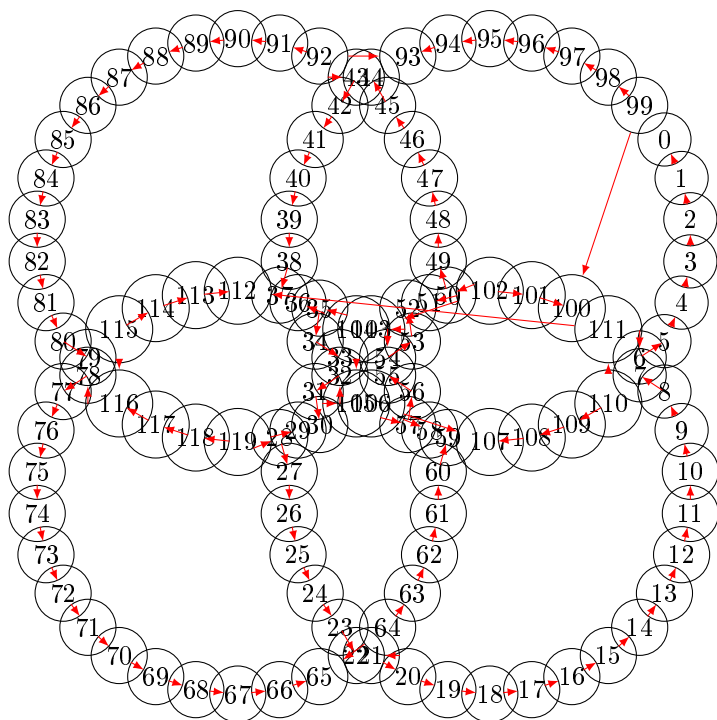
```

---

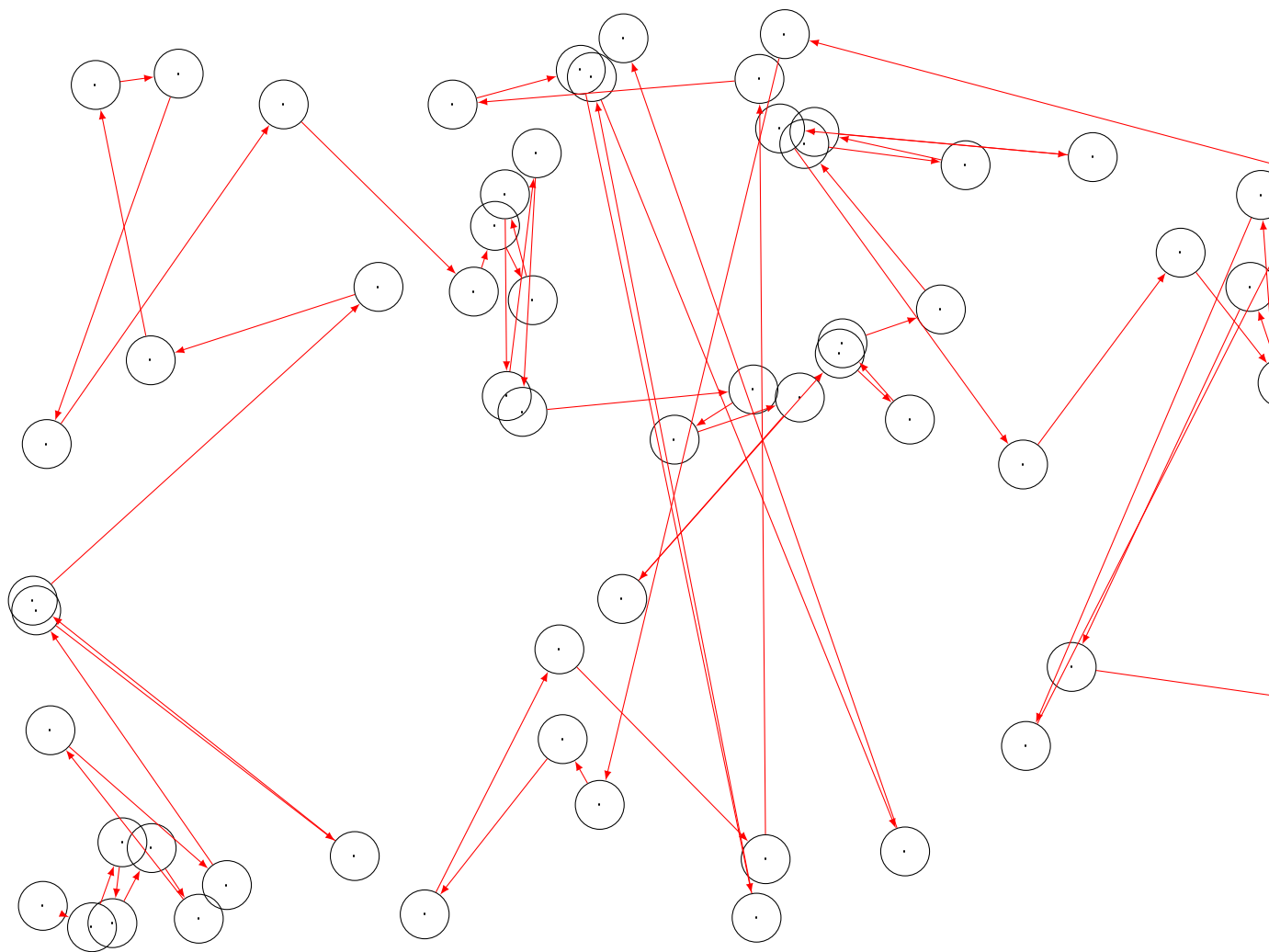
### **3 Beispiele**

Genügend Beispiele einbinden! Die Beispiele von der BwInf-Webseite sollten hier diskutiert werden, aber auch eigene Beispiele sind sehr gut – besonders wenn sie Spezialfälle abdecken. Aber bitte nicht 30 Seiten Programmausgabe hier einfügen!

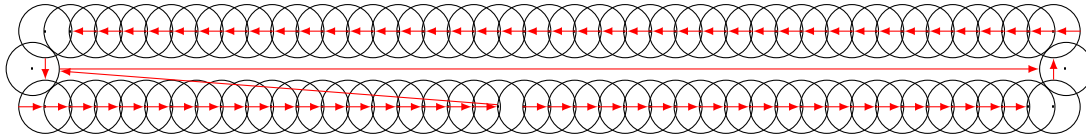


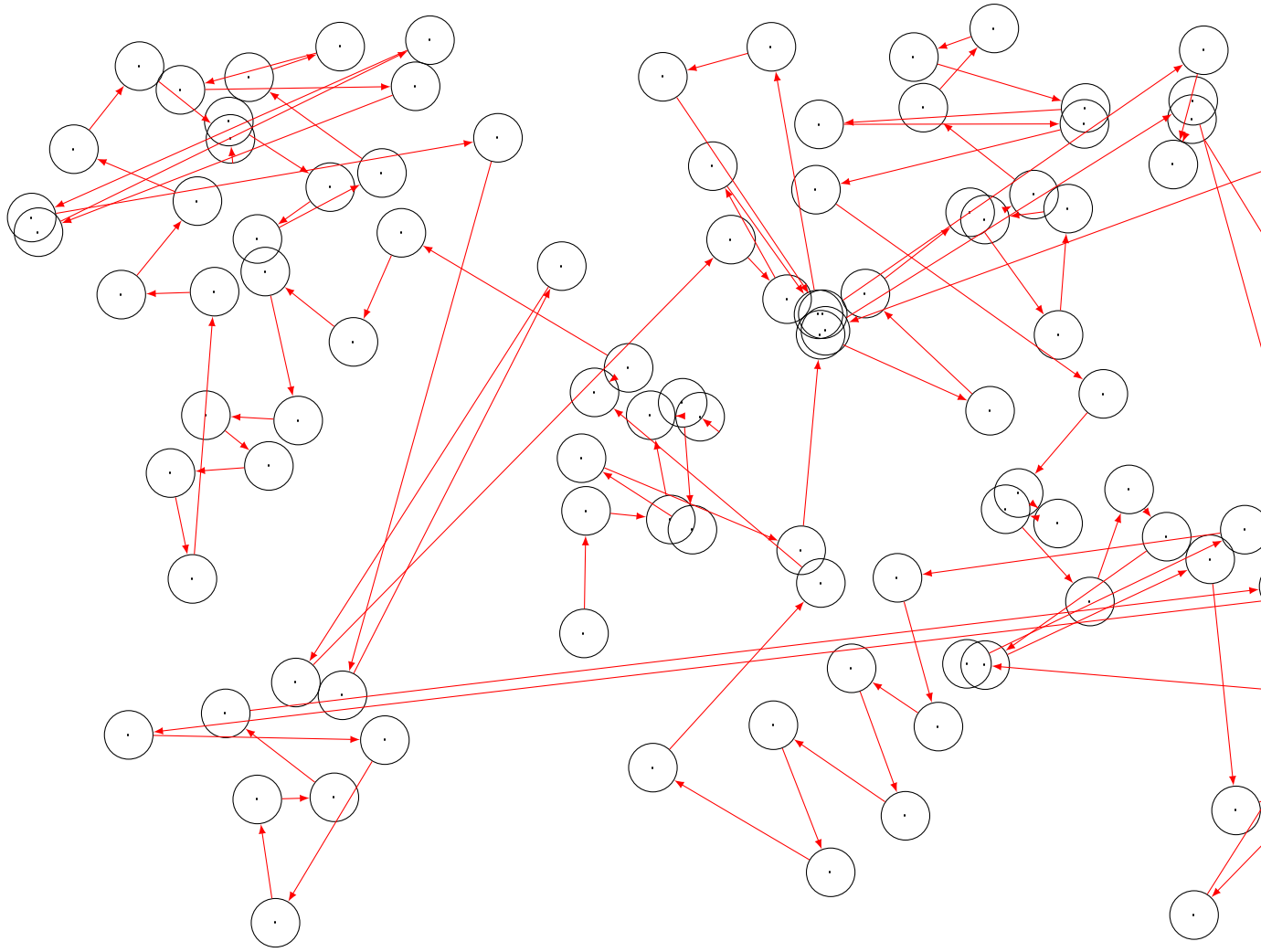


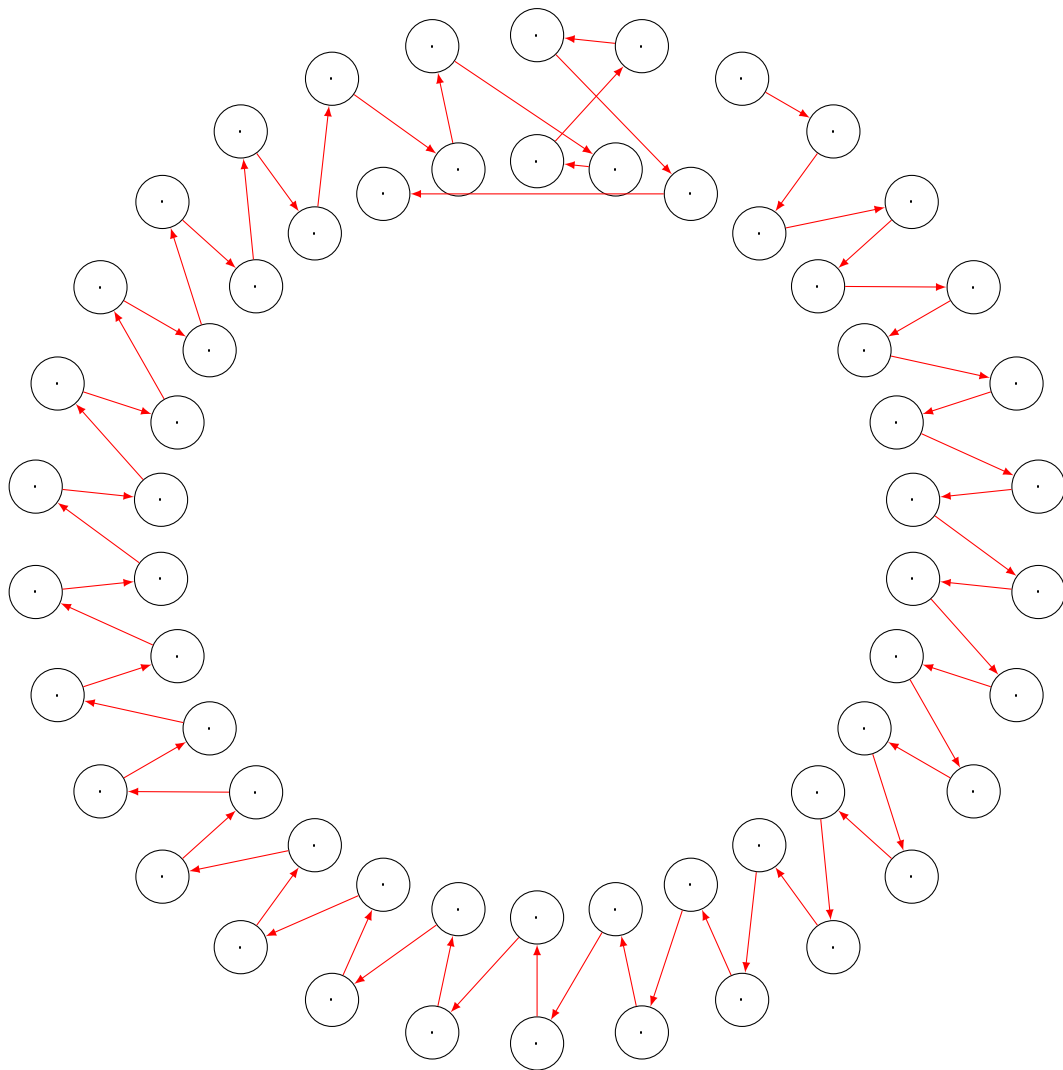


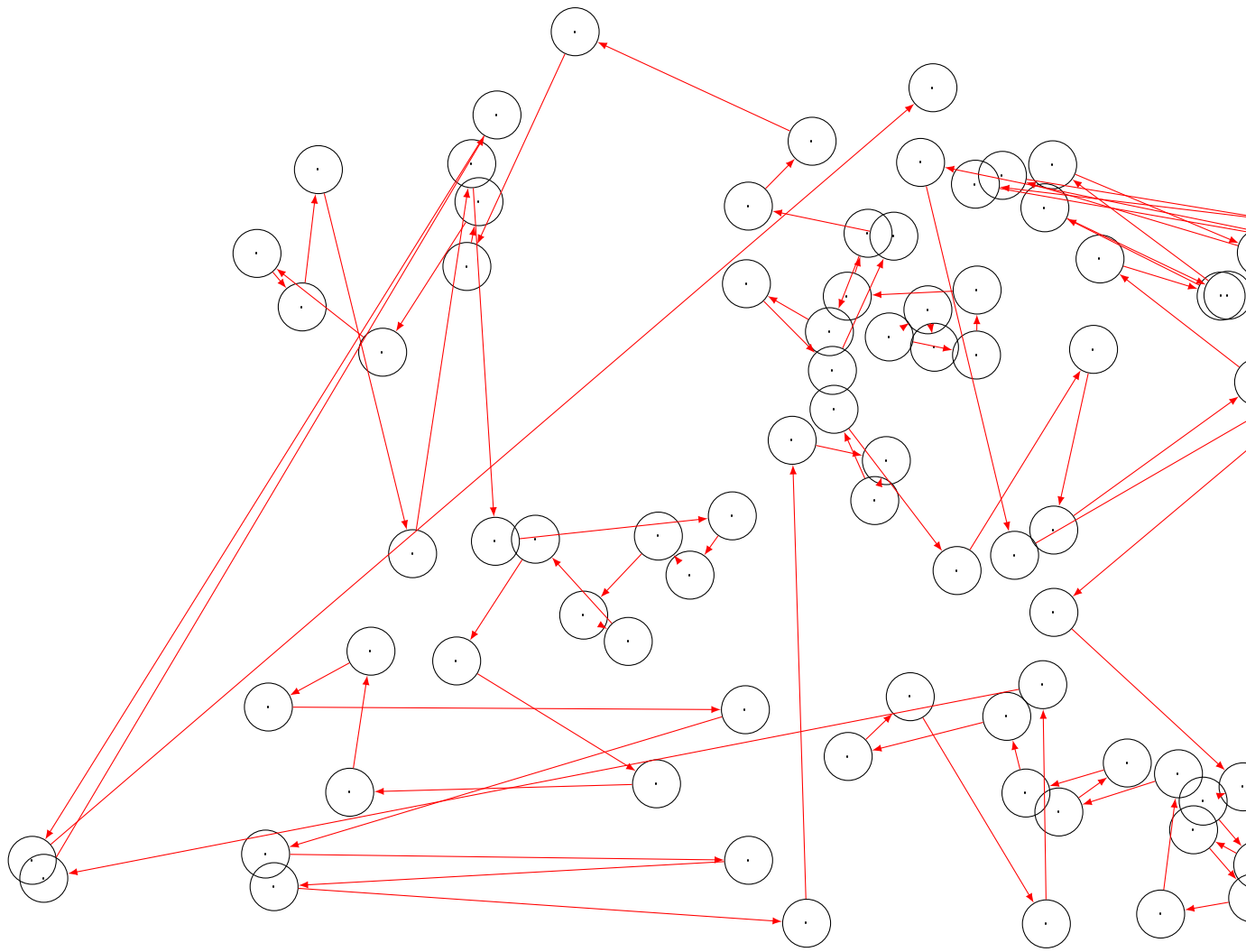


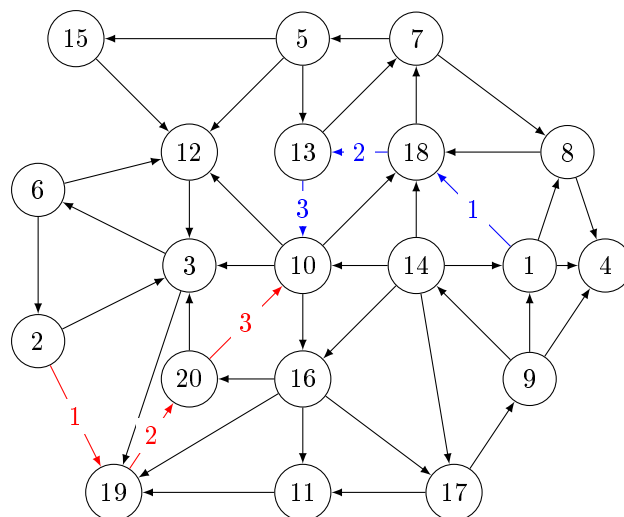












Figur 4: huepfburg0.txt Parcours

Schritt	Spieler 1 Knotenmenge	Spieler 2 Knotenmenge
0.	{ 1 }	{ 2 }
1.	{ 4, 8, 18 }	{ 3, 19 }
2.	{ 4, 7, 13, 18 }	{ 6, 19, 20 }
3.	{ 5, 7, 10, 13 }	{ 2, 3, 10, 12, 20 }

Tabelle 3: Schrittfolge für Figur 4.

Bei Figur 4 handelt es sich um den Beispielparcours aus der Aufgabenstellung beziehungsweise um den Parcours aus huepfburg0.txt aus den Eingabedateien. Es handelt sich daher um einen lösbaren Parcours, in dem sich beide Spieler im dritten Schritt auf Feld zehn treffen. Wie Tabelle 3 zu entnehmen ist, ergibt sich im dritten Schritt der Knoten zehn aus der Schnittmenge der zwei Breitensuchen. In der Methode `int[][] sameTargetRoute(...)` liefert die Bedingung in der `do-while` Schleife in Schritt drei `false`. Daher wird im Anschluss der Zielknoten durch die Methode `int firstSameTargetOfTimelines(...)` gesetzt und die Wege mit `int[] findSingleRouteInTimeline(...)` gesetzt und anschließend wie folgt ausgegeben ausgegeben.

```

1  Ergebnis für huepfburg0.txt
2  Der Parcours hat folgende Lösung:
3  Zielfeld: 10
4  Anzahl an Schritten: 4
5  Sasha's Weg:
6  1 -> 18 -> 13 -> 10
7  Mika's Weg:
8  2 -> 19 -> 20 -> 10
9

```

Programmausgabe Figur 4.

## 4 Quellcode

```

2  /**
3  * Liest die Eingabedateien ein
4  * und versucht für jede Datei eine Lösung entsprechend der Aufgabenstellung zu finden
5  * Die Lösung wird anschließend in die entsprechende Ausgabedatei geschrieben
6  * Sollte es keine Lösung geben, wird dies ebenfalls in die Ausgabedatei geschrieben
7  * @return 0, wenn es zu keinem RuntimeError oder keiner RuntimeException gekommen ist
8  */
9  int main() {
10     string input_dir = "../LennartProtte/Aufgabe1-Implementierung/Eingabedateien";
11     string output_dir = "../LennartProtte/Aufgabe1-Implementierung/Ausgabedateien";
12
13     //Durchläuft alle Dateien im Eingabeordner
14     for (const std::filesystem::directory_entry &entry: filesystem::directory_iterator(
15         input_dir)) {
16
17         //Liest den Dateinamen aus
18         string input_file = entry.path();
19         string output_file = output_dir + "/" + entry.path().filename().string();
20
21         //Öffnet die Eingabedatei
22         ifstream fin(input_file);
23
24         //Öffnet die Ausgabedatei
25         ofstream fout(output_file);
26
27         //Liest die Eingabedatei ein
28         vector<pair<double, double> > coordinates;
29         double x, y;
30         while (fin >> x >> y) {
31             coordinates.emplace_back(x, y);
32         }
33
34         //Berechnet die Lösung
35         vector<pair<double, double> > result;
36         if (solve(result, coordinates)) {
37             fout << "Es konnte eine Flugstrecke durch alle Außenposten ermittelt werden"
38             << endl;
39             for (int i = 0; i < result.size(); i++) {
40                 if (i != 0 && i != result.size() - 1) {
41                     fout << cross_angle(result[i - 1], result[i], result[i + 1]) << "°";
42                 }
43                 fout << "[" << result[i].first << ", " << result[i].second << "]" << "->";
44             }
45             } else {
46                 fout << "Es konnte keine Flugstrecke durch alle Außenposten ermittelt werde"
47                 << endl;
48             }
49         }
50         return 0;
51     }
52 }

```

Methode graphFromLines

```

1  /**
2  * Versucht rekursiv mit backtracking eine möglichst kurze Route durch den Graphen zu
3  * finden,
4  * welche die Kriterien der Aufgabenstellung erfüllt.
5  * @param route die aktuelle Route
6  * @param coordinates eine Menge aller eingelesenen Koordinaten
7  * @return true, wenn alle Knoten in der Lösungsmenge (route) enthalten sind, sonst false
8  */
9  bool solve(vector<pair<double, double> > &route, vector<pair<double, double> > &
10             coordinates) {
11     //Wenn alle Knoten in der Lösungsmenge sind
12     if (route.size() == coordinates.size()) {
13         return true;
14     }
15     //Sortiere nach dem nächsten Knoten
16     if (!route.empty()) {
17         const auto &p = route.back();
18         sort(coordinates.begin(), coordinates.end(),
19              [p](const auto &lhs, const auto &rhs) {
20                  return sqrt(pow((p.first - lhs.first), 2.0) + (pow((p.second - lhs.
21                  second), 2.0)))
22                  < sqrt(pow((p.first - rhs.first), 2.0) + (pow((p.second - rhs.second),
23                  2.0)));
24              });
25     }
26     //Für jeden Knoten
27     for (int i = 0; i < coordinates.size(); i++) {
28         //Wenn dieser Knoten bereits in der Lösungsmenge existiert, überspringe diesen
29         if (std::find(route.begin(), route.end(), coordinates[i]) != route.end()) {
30             continue;
31         }
32         double angle = -1;
33         if (route.size() >= 2) {
34             angle = cross_angle(route[route.size() - 2], route.back(), coordinates[i]);
35         }
36         if (route.empty() ||
37             (std::find(route.begin(), route.end(), coordinates[i]) == route.end() &&
38              (route.size() < 2 || angle >= 90 || angle == 0)))
39         {
40             //Füge den Knoten hinzu
41             route.push_back(coordinates[i]);
42             //Wenn es eine Lösung mit der aktuellen Route gibt
43             if (solve(route, coordinates)) {
44                 return true;
45             } else {
46                 route.pop_back();
47             }
48         }
49     }
50     //Wenn es mit der aktuellen Route keine Lösung geben kann
51     return false;
52 }

```

Methode graphFromLines

```
2  /**
3  * Berechnet den Winkel zwischen den Vektoren von from_node nach over_node und over_node
4  * nach to_node
5  * @param over_node der zweite Knoten
6  * @param to_node der dritte Knoten (Zielknoten)
7  * @param from_node der erste Knoten
8  * @return false, wenn der Winkel der Kanten kleiner als 90° beträgt, sonst true
9  */
10 double cross_angle(const pair<double, double> &from_node,
11                   const pair<double, double> &over_node,
12                   const pair<double, double> &to_node) {
13     pair<double, double> p, q;
14     p = make_pair(over_node.first - from_node.first,
15                  over_node.second - from_node.second);
16     q = make_pair(to_node.first - over_node.first,
17                  to_node.second - over_node.second);
18     double angle = acos(
19         (p.first * q.first + p.second * q.second) / (
20             sqrt(pow(p.first, 2.0) + (pow(p.second, 2.0))) *
21             sqrt(pow(q.first, 2.0) + (pow(q.second, 2.0)))
22         )
23     ) * 180 / M_PI; //Umrechnung von Radian nach Grad
24     if(angle > 180) {
25         angle = 180 - angle;
26     }
27     return angle;
28 }
```

Methode graphFromLines