

Aufgabe 1: Weniger krumme Touren

Teilnahme-ID: ?????

Bearbeiter/-in dieser Aufgabe:
Lennart Protte

February 16, 2023

Contents

| | | |
|----------|--------------------|----------|
| 1 | Lösungsidee | 2 |
| 2 | Umsetzung | 3 |
| 3 | Beispiele | 4 |
| 4 | Quellcode | 5 |

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 "‘ \LaTeX -Dokument'")!

Dann kannst du dieses Dokument mit deiner \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.

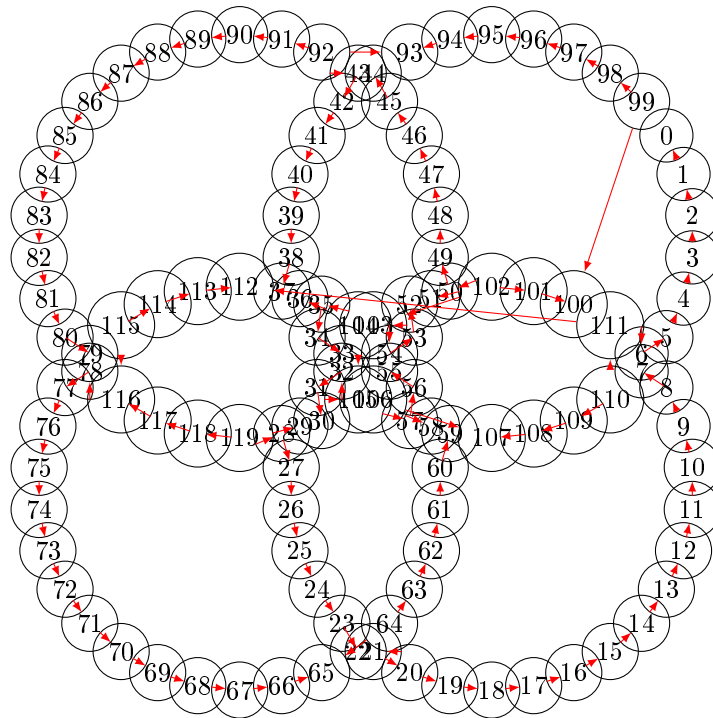
2 Umsetzung

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

```

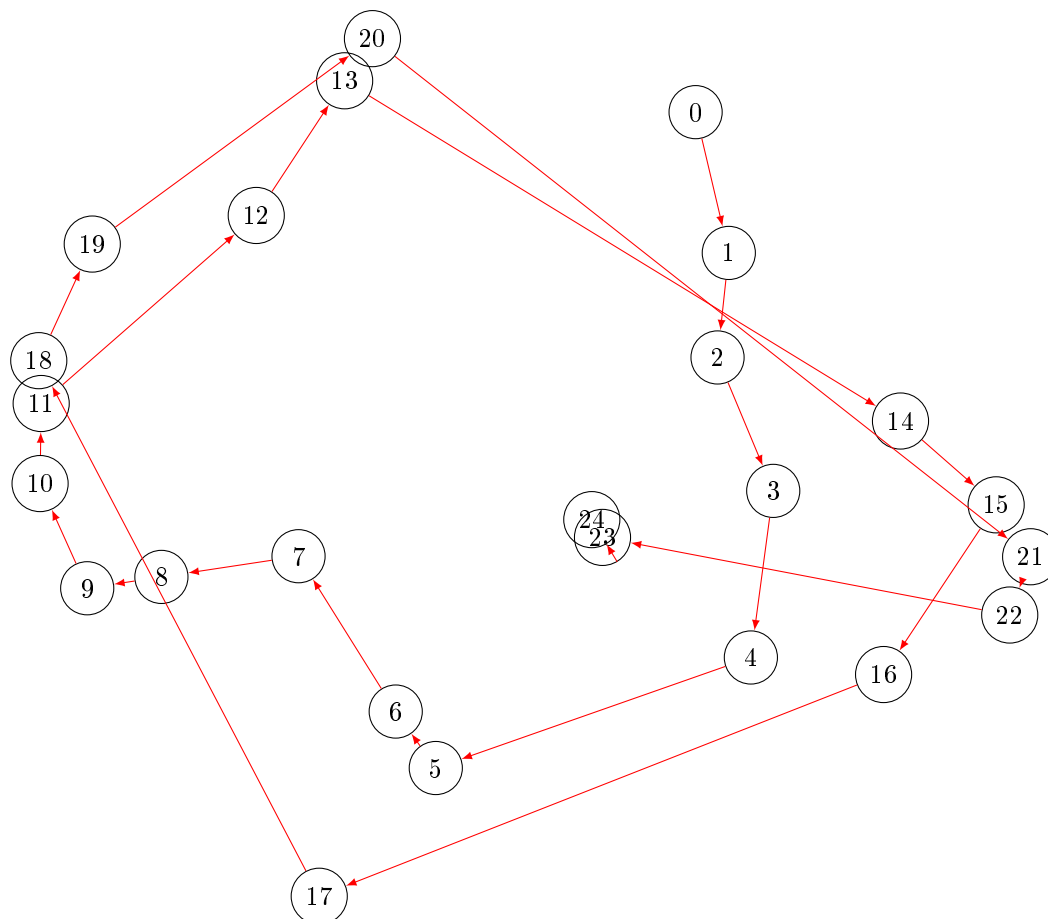
1: function CROSSANGLE(fromnode, overnode, tonode)
2:   p ← MAKEPAIR(overnode.first − fromnode.first, overnode.second − fromnode.second)
3:   q ← MAKEPAIR(tonode.first − overnode.first, tonode.second − overnode.second)
4:   angle ← acos( $\frac{p.firstq.first + p.secondq.second}{\sqrt{p.first^2 + p.second^2}\sqrt{q.first^2 + q.second^2}}$ ) * 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

```

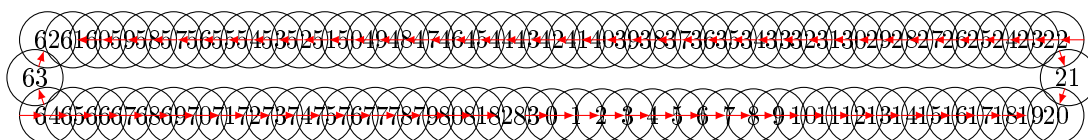


Figur: wenigerkrumm3

3 Beispiele



Figur: wenigerkrumm4



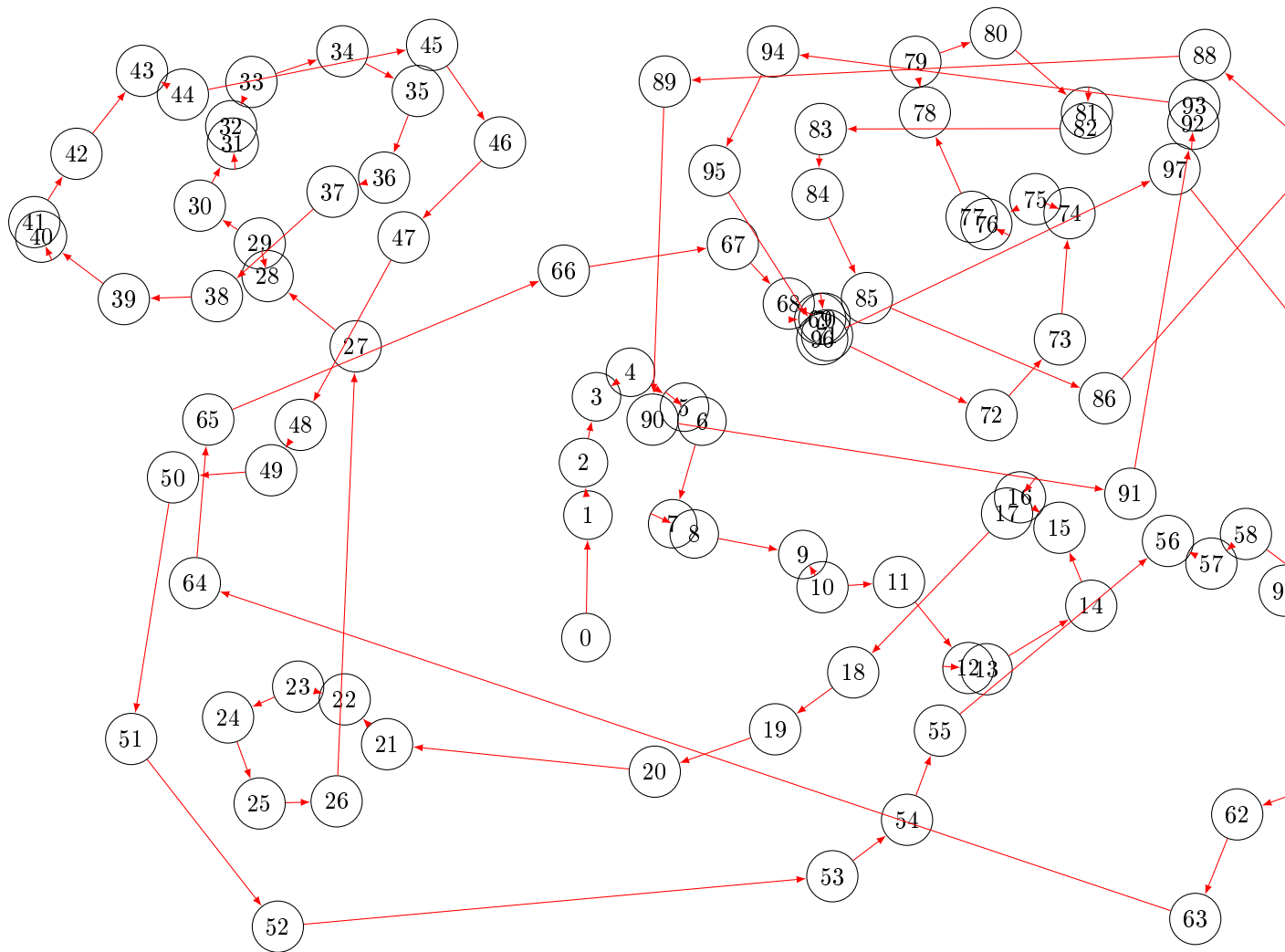
Figur: wenigerkrumm1

4 Quellcode

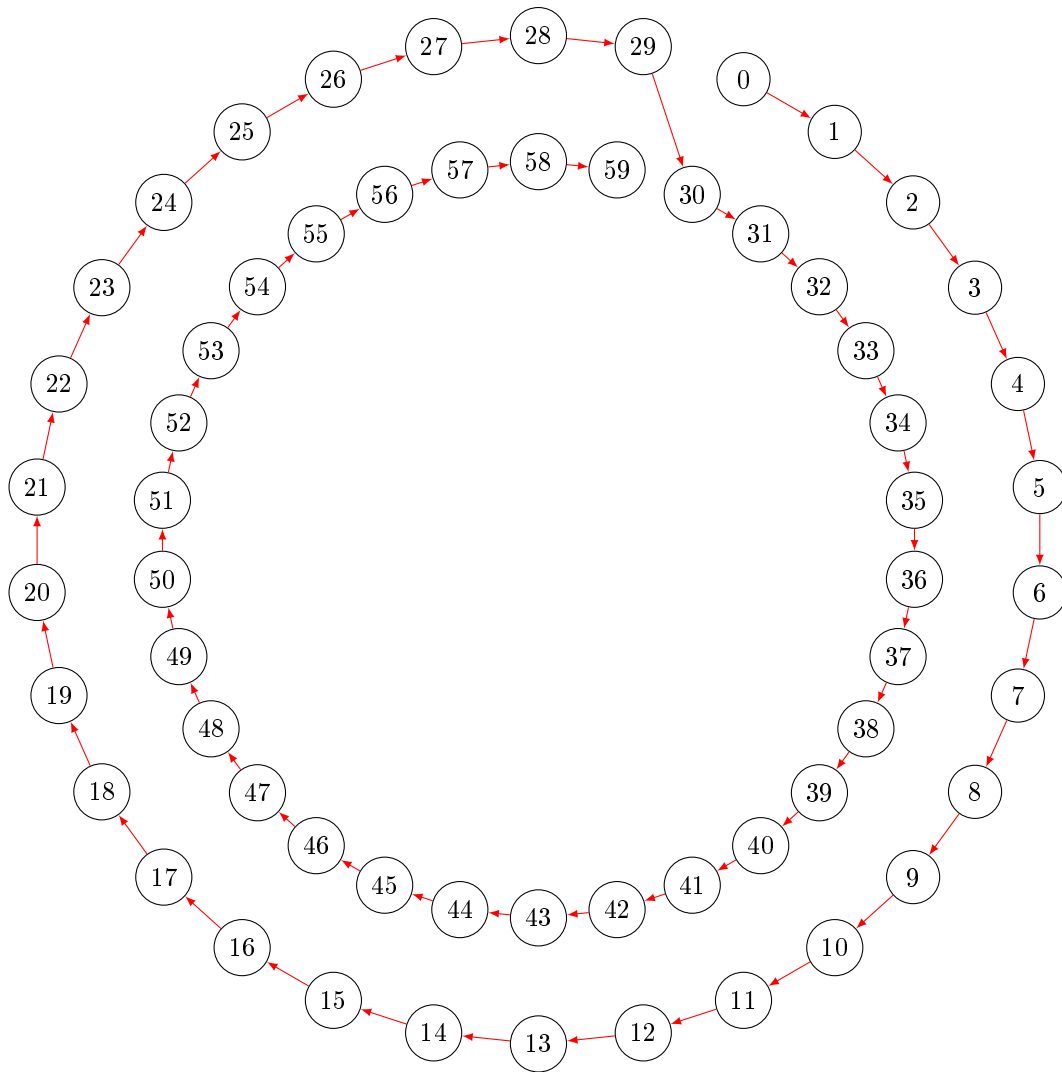
```

1  /**
2  * Liest die Eingabedateien ein
3  * und versucht für jede Datei eine Lösung entsprechend der Aufgabenstellung zu finden
4  * Die Lösung wird anschließend in die entsprechende Ausgabedatei geschrieben
5  * Sollte es keine Lösung geben, wird dies ebenfalls in die Ausgabedatei geschrieben
6  * @return 0, wenn es zu keinem RuntimeError oder keiner RuntimeException gekommen ist
7  */
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);

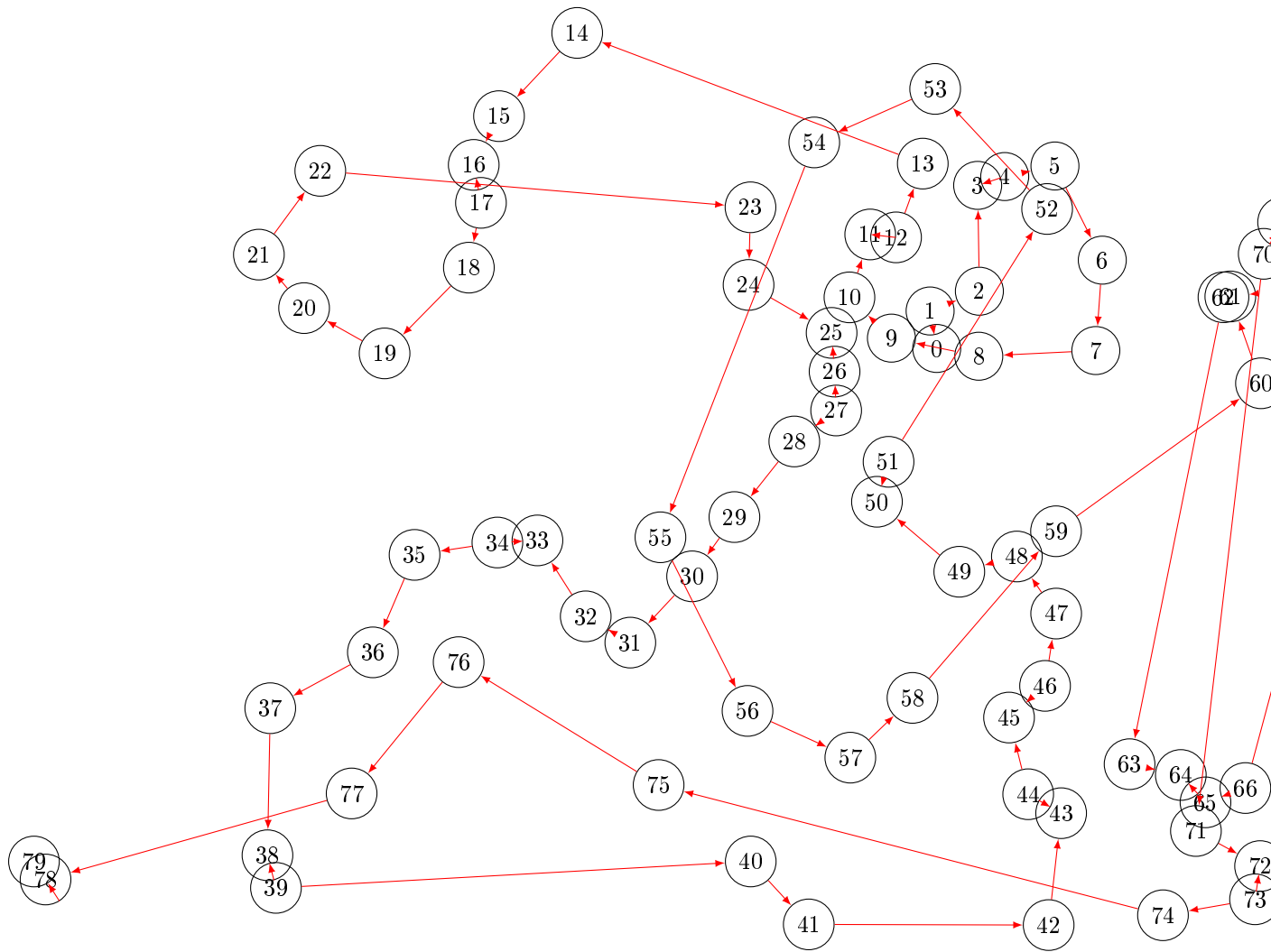
```



Figur: wenigerkrumm7



Figur: wenigerkrumm2



Figur: wenigerkrumm6


```

21      //Öffnet die Ausgabedatei
22      ofstream fout(output_file);
23
24      //Liest die Eingabedatei ein
25      vector<pair<double, double> > coordinates;
26      double x, y;
27      while (fin >> x >> y) {
28          coordinates.emplace_back(x, y);
29      }
30
31      //Berechnet die Lösung
32      vector<pair<double, double> > result;
33      if (solve(result, coordinates)) {
34          fout << "Es_konnte_eine_Flugstrecke_durch_alle_Außenposten_ermittelt_werden"
35      << endl;
36          for (int i = 0; i < result.size(); i++) {
37              if (i != 0 && i != result.size() - 1) {
38                  fout << cross_angle(result[i - 1], result[i], result[i + 1]) << "°";
39              }
40              fout << "[" << result[i].first << ", " << result[i].second << "]" << "->";
41          }
42          } else {
43          fout << "Es_konnte_keine_Flugstrecke_durch_alle_Außenposten_ermittelt_werde"
44      << endl;
45          }
46      }
47      return 0;

```

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