**Aufgabe 2: Rechtsrun in Rechthausen**

Definition Linksabbiegen

Nach der Definition, die in dem Programm benutzt wird, ist ein Abbiegevorgang Linksabbiegen, wenn:

- Die Anzahl der Straßen, die von dieser Straße wegführen, größer als zwei ist

- Die Straße nicht die erste Straße gegen den Uhrzeigersinn von der Straße ist, von der die Kreuzung betreten wird

- Die Straße links von dem verlängerten Vektor liegt, der die Fahrtrichtung auf der Straße zuvor beschreibt

Basierend auf dieser Definition sind um zu bestimmen, ob ein Abbiegevorgang Linksabbiegen ist, die aktuelle Kreuzung, die Kreuzung, von der man kommt, und die Zielkreuzung erforderlich.

Diese Definition liefert die in der Aufgabe geforderten Ergebnisse.

Weitere Beispiele:

Von HF sind G und I erlaubt, E und C dagegen nicht.

Von FG ist I erlaubt, C dagegen nicht.

Kürzesten Weg finden

Die beste Methode, den Weg zu finden, ist, alle erlaubten Wege von S zu prüfen, bis alle Wege zu T führen oder durch K.O.- Kriterien von der Suche nach dem Optimum ausgeschlossen werden

K.O.- Kriterien

- Die Folge von zwei Kreuzungen trat in irgendeinem Weg früher auf

Beispiel: ABCD wird von ACDE ausgeschlossen

- Die Anzahl der Wege und die Gesamtlänge des Weges überschreiten die besten bisher gefundenen Lösungen

Dadurch reduziert sich, sobald eine Lösung gefunden wurde, die Zahl der noch übrigen Kandidaten auf wenige, was recht schnell zum Ende der Suche führt.

Sind alle Kreuzungskombinationen möglich?

Hier wird für jede Kombination von S und T eine Suche wie bei der Suche nach dem kürzesten Weg durchgeführt, doch sobald eine Lösung gefunden wird, wird die Suche abgebrochen. Gibt es für eine Kombination kein Ergebnis, wird die Suche abgebrochen und festgestellt, dass es eine nicht durch Nur-Rechtsabbiegen erreichbare Kreuzungskombination gibt.

Wie stark verlängert das Verbot die Weglänge maximal?

Auch hier wird eine modifizierte Version der Suche nach dem kürzesten Weg benutzt, einmal wird Linksabbiegen verboten, einmal erlaubt. Die K.O.- Kriterien sind auch hier anwendbar. Anschließend wird der Faktor berechnet, um den die Version ohne Linksabbiegen länger ist.

Diese Berechnung wird für alle Kombinationen von S und T durchgeführt und die Kombination gesucht, für die dieser Faktor am Größten ist.

Umsetzung

Das Programm ist eine Windows Forms Anwendung, die Bedienung ist selbsterklärend.

WICHTIG: Das Programm benutzt zur Identifikation der Kreuzungen bei Teilaufgabe 3 die Ids aus der Datei.

BUG: Nach erstem Auswählen der Datei ist das Programm in der Taskleiste

Gelegentlich schließt sich das Programm nicht korrekt.

public class Kreuzung

{

public string Id;

public Point Position;

public List<string> Anschlüsse = new List<string>();

}

public class Pfad

{

public List<string> IDs = new List<string>();

public State Status = State.None;

public double Length = 0;

public Pfad Clon()

{

Pfad path = new Pfad();

path.Status = Status;

path.IDs = IDs;

return path;

}

public enum State

{

None,

Ok,

Invalid

}

}

public static List<Kreuzung> Kreuzungen;

private void Form1\_Load(object sender, EventArgs e)

{

if (fileDialog.ShowDialog() == DialogResult.OK)

{

Kreuzungen = LoadFile(fileDialog.FileName);

}

}

public static List<Kreuzung> LoadFile(string path)

{

List<Kreuzung> result = new List<Kreuzung>();

using (var reader = new StreamReader(path))

{

//Nicht benötigte Zeilen einlesen

reader.ReadLine();

reader.ReadLine();

reader.ReadLine();

while (true)

{

//Kreuzungen einlesen

var line = reader.ReadLine();

if (line.StartsWith("#"))

break;

var k = line.Split(' ');

Kreuzung kreuzung = new Kreuzung()

{

Id = k[0],

Position = new System.Windows.Point(Convert.ToInt32(k[1]), Convert.ToInt32(k[2]))

};

result.Add(kreuzung);

}

var street = Convert.ToInt32(reader.ReadLine());

for (int i = 0; i < street; i++)

{

//Anschlüsse einlesen

var line = reader.ReadLine().Split(' ');

result.First(x => x.Id == line[0])

.Anschlüsse.Add(line[1]);

result.First(x => x.Id == line[1])

.Anschlüsse.Add(line[0]);

}

}

return result;

}

static double GetAngle(Kreuzung start, Kreuzung position, Kreuzung goal)

{

if (start.Id == goal.Id)

return 180;

var facing = new Vector(position.Position.X - start.Position.X, position.Position.Y - start.Position.Y);

var direction = new Vector(goal.Position.X - position.Position.X, goal.Position.Y - position.Position.Y);

var angle = Vector.AngleBetween(facing, direction);

return angle;

}

private static List<Pfad> Pfade(

Kreuzung start, Kreuzung goal, bool leftTurnAllowed = false, bool bestSolution = true)

{

if (goal.Anschlüsse.Count == 0)

return new List<Pfad>();

//Initialisieren

var paths = new List<Pfad>() {new Pfad()};

paths.First().IDs.Add(start.Id);

while (paths.Any(x => x.Status == Pfad.State.None))

{

#region Check

var recLength = double.MaxValue;

var recCount = int.MaxValue;

if (paths.Any(x => x.Status == Pfad.State.Ok))

{

recLength = paths.Where(x => x.Status == Pfad.State.Ok).OrderBy(o => o.Length).First().Length;

recCount = paths.Where(x => x.Status == Pfad.State.Ok).OrderBy(o => o.IDs.Count).First().IDs.Count;

if(!bestSolution)

return new List<Pfad>(){ paths.First(x => x.Status == Pfad.State.Ok)};

}

foreach (var path in paths.Where(x => x.Status == Pfad.State.None))

{

//Regel 1

if (path.IDs.Count > 3)

{

foreach (var p in paths.Where(x => x.Status != Pfad.State.Invalid))

{

for (var i = 0; i < p.IDs.Count - 2; i++)

{

if (path.IDs[path.IDs.Count - 2] == p.IDs[i] &&

path.IDs[path.IDs.Count - 1] == p.IDs[i + 1] &&

i < path.IDs.Count - 2)

{

path.Status = Pfad.State.Invalid;

break;

}

}

if (path.Status == Pfad.State.Invalid)

break;

}

}

//Regel 2

if (path.Length > recLength && path.IDs.Count > recCount)

path.Status = Pfad.State.Invalid;

//Wurde das Ziel erreicht?

if (path.IDs.Last() == goal.Id)

path.Status = Pfad.State.Ok;

}

#endregion

List<Pfad> newpaths = new List<Pfad>();

foreach (var p in paths.Where(x => x.Status == Pfad.State.Ok).ToList())

{

newpaths.Add(p);

}

foreach (var path in paths.Where(x => x.Status == Pfad.State.None).ToList())

{

var kreuzung = Kreuzungen.First(x => x.Id == path.IDs.Last());

List<string> validGoals = new List<string>();

if (kreuzung.Anschlüsse.Count < 2 || path.IDs.Count == 1 || leftTurnAllowed)

validGoals = kreuzung.Anschlüsse;

else

{

var dict = new Dictionary<string, double>();

foreach (var k in kreuzung.Anschlüsse)

{

dict.Add(k,

GetAngle(Kreuzungen.First(x => x.Id == path.IDs[path.IDs.Count - 2]),

Kreuzungen.First(x => x.Id == path.IDs.Last()),

Kreuzungen.First(x => x.Id == k)));

}

//Suche nach erlaubten Abbiegevorgängen

if (dict.Values.Any(x => x < 1))

validGoals = dict.Where(x => x.Value < 1).ToDictionary(key => key.Key, value => value.Value).Keys.ToList();

else

validGoals.Add(dict.OrderBy(o => o.Value).First().Key);

}

if (validGoals.Count == 0)

{

path.Status = Pfad.State.Invalid;

}

else

{

foreach (var validGoal in validGoals.ToList())

{

var newPfad = new Pfad()

{

IDs = path.IDs.ToList(),

Status = path.Status,

Length = path.Length

};

newPfad.IDs.Add(validGoal);

//Update Length

var a = Kreuzungen.First(x => x.Id == validGoal);

var b = Kreuzungen.First(x => x.Id == newPfad.IDs[newPfad.IDs.Count - 2]);

newPfad.Length += Math.Sqrt(

Math.Pow(a.Position.X - b.Position.X, 2)

+ Math.Pow(a.Position.Y - b.Position.Y, 2));

newpaths.Add(newPfad);

}

}

}

paths = newpaths;

}

//Kürstesten Weg nach beiden Methoden herausfinden

var result = new List<Pfad>();

if (paths.Any(x => x.Status == Pfad.State.Ok))

{

//Rückgabeliste füllen

result.Add(paths.Where(x => x.Status == Pfad.State.Ok).OrderBy(o => o.IDs.Count).First());

result.Add(paths.Where(x => x.Status == Pfad.State.Ok).OrderBy(o => o.Length).First());

}

return result;

}

private void btnTA3\_Click(object sender, EventArgs e)

{

if (Kreuzungen.All(x => x.Id != txtFrom.Text) || Kreuzungen.All(x => x.Id != txtTo.Text))

MessageBox.Show("Bitte gültige Kreuzungsnamen eingeben.");

else

{

var res = Pfade(Kreuzungen.First(x => x.Id == txtFrom.Text),

Kreuzungen.First(x => x.Id == txtTo.Text));

MessageBox.Show($"Schnellster Weg nach Straßenzahl: {String.Join(" ", res[0].IDs)}, nach Länge: {String.Join(" ", res[1].IDs)} ({res[1].Length})");

}

}

private void btnTA4\_Click(object sender, EventArgs e)

{

foreach (Kreuzung t in Kreuzungen)

{

foreach (Kreuzung t1 in Kreuzungen)

{

if (t.Id == t1.Id)

break;

if (Pfade(t, t1, false, false).Count == 0)

{

MessageBox.Show($"Nicht möglich bei {t.Id}-{t1.Id}");

return;

}

}

}

MessageBox.Show("Immer möglich.");

}

private void btnA5\_Click(object sender, EventArgs e)

{

var count = new Dictionary<string, float>();

var length = new Dictionary<string, float>();

using (var writer = new StreamWriter("output.txt"))

{

writer.WriteLine("#Format: Start-Ziel: (Schnellster Pfad nach Straßenzahl)-Faktor(Teilaufgabe 5);(Schnellster Pfad nach Länge)-Faktor");

for (var i = 0; i < Kreuzungen.Count; i++)

{

for (var ii = 0; ii < Kreuzungen.Count; ii++)

{

if (i == ii) continue;

var left = Pfade(Kreuzungen[i], Kreuzungen[ii], true);

var noLeft = Pfade(Kreuzungen[i], Kreuzungen[ii]);

writer.Write($"{Kreuzungen[i].Id}-{Kreuzungen[ii].Id}: ");

if (left.Count == 0)

writer.WriteLine("Nicht möglich.");

else if (noLeft.Count == 0)

{

writer.Write("Nur mit Linksabbiegen möglich. ");

writer.Write(String.Join(" ", left[0].IDs));

writer.Write(";");

writer.Write(String.Join(" ", left[1].IDs));

writer.WriteLine();

}

else

{

writer.Write(String.Join(" ", noLeft[0].IDs));

writer.Write(

$"-{Convert.ToSingle(noLeft[0].IDs.Count) / Convert.ToSingle(left[0].IDs.Count)}");

count.Add($"{noLeft[0].IDs.First()}-{noLeft[0].IDs.Last()}", Convert.ToSingle(noLeft[0].IDs.Count) / Convert.ToSingle(left[0].IDs.Count));

writer.Write(";");

writer.Write(String.Join(" ", noLeft[1].IDs));

writer.Write(

$"-{Convert.ToSingle(noLeft[1].Length) / Convert.ToSingle(left[1].Length)}");

length.Add($"{noLeft[1].IDs.First()}-{noLeft[1].IDs.Last()}", Convert.ToSingle(noLeft[1].Length) / Convert.ToSingle(left[1].Length));

writer.WriteLine();

}

Debug.WriteLine($"{Kreuzungen[i].Id}-{Kreuzungen[ii].Id}");

}

}

}

MessageBox.Show($"Maximum bei {count.OrderBy(o => o.Value).Last().Key} ({count.OrderBy(o => o.Value).Last().Value}) für Straßenzahl. " +

$"Maximum bei {length.OrderBy(o => o.Value).Last().Key} ({length.OrderBy(o => o.Value).Last().Value}) für Länge.");

Process.Start("output.txt");

}

private void btnLoad\_Click(object sender, EventArgs e)

{

if (fileDialog.ShowDialog() == DialogResult.OK)

{

Kreuzungen = LoadFile(fileDialog.FileName);

}

}

Beispiele

Das Programm wurde mit dem offiziellen Beispiel sowie mit einigen Beispielen von BwInf-Teilnehmern geprüft. Die Eingabedateien finden sich in dem Order „Aufgabe 2 – Beispiele“.

Die Ergebnisse lauten:

|  |  |  |  |
| --- | --- | --- | --- |
| Dateiname | „A – B“: Kürzester Weg | Jede Kreuzung erreichbar | Linksfaktor |
| rechtshausen1.txt | ADEFHEB(beide) | Ja | HD (2,33; 3) |
| U-Turn.txt | AB(beide) | Ja | EC (2,33; 3) |
| beispiel1.txt | AB(beide) | Nein (H-A) | PL (5,33; 9,15) |
| beispiel2.txt | AIB (beide) | Nein (S-A) | fo (13); RL (22,77) |

Laufzeit

Die Laufzeit steigt bei zunehmender Kreuzungszahl trotz Filterung nach nicht brauchbaren Pfaden extrem schnell an, besonders bei Teilaufgabe 5. Das liegt daran, dass bei höherer Straßenzahl die Anzahl der Pfade sich bei jeder Stufe, selbst wenn es nur zwei mögliche Abzweigungen gibt, verdoppelt. Dazu müssen bei Teilaufgabe 5 (2\*n²)-optimale Pfade berechnet werden, womit die Anzahl der Berechnungen exponentiell steigt. Diese Eigenschaft lässt sich, da sie in der Natur der Aufgabe liegt, nicht optimieren, da der Faktor nicht vorhergesagt werden kann und jeweils in beide Richtungen gerechnet werden muss (AB – BA).