Aufgabe 4: Urlaubsfahrt

Team-ID: 00587

Team-Name: Doge.NET

Bearbeiter dieser Aufgabe:
Johannes von Stoephasius

24. November 2019

Inhaltsverzeichnis

1 Lösungsidee

1.1 Definitionen

Weg Ein ”Weg” ist eine Menge von zu besuchenden Tankstellen .

optimaler Weg Der ”beste Weg” oder ”optimale Weg”, ist der Weg mit den wenigsten Stopps, und geringsten Preis um vom Ausganspunkt zu einer gegebenen Tankstelle zu kommen. Hierbei wird wie in der Aufgabenstellung eine minimal Stoppzahl priorisiert.

1.2 Kernidee

Der Algorithmus basiert auf der Idee durch die Lösung von Teilproblemen die insgesamt beste Lösung zu finden. Das heißt, dass zur Findung des besten Weges zu einem Punkt ein neuer Weg aus dem besten Weg zu den Punkten davor gebildet wird.

Zum Finden des besten Weges zu eienm Punkt wird der Weg von den Wegen zu den Punkten davor gewählt, der am wenigsten Stopps braucht und den geringsten Preis hat. Durch den iterativen Aufbau von den besten Wegen ist garantiert, dass am Ende der beste Weg zum Ziel gefunden wird.

1.3 Finden des besten Weges zum Ziel

Zum finden des besten Weges zum Ziel wird nach und nach eine Zuordnung aufgebaut, die für jede Tankstelle den besten Weg zu ihr enthält. Diese Zuordnung wird für eine beliebige Tankstelle ermittelt, indem zuerst für alle vorherigen Tankstellen eine Zuordnung ermittelt wird.

Sind die optimalen Wege zu allen Tankstellen vor einer gegebenen Tankstelle errechnet, kann die Ermittlung des optimalen Wegs zu dieser beginnen. Zuerst werden alle optimale Wege zu den vorherigen Tankstelle durchiteriert. Dabei wird zuerst überprüft von welchen dieser Tankstellen die aktuell betrachtete Tankstelle überhaupt erreicht werden kann, wenn nicht wird zur nächsten Tankstelle der Zuordnung übergegangen. Wenn doch, so wird ein neuer Weg gebildet, der aus dem vorherigen Weg und der betrachtete Tankstelle besteht.

Von allen dieser neuen Wege werden nun die kürzesten ausgewählt, also alle Wege mit einer Stoppzahl gleich der minimalen Stoppzahl. Danach wird der Preis aller übrigen Wege ermittelt (siehe Abschnitt 1.4), und von denen wird der billigste ausgewählt. Dieser Weg ist dann der optimale Weg vom Start zur aktuell betrachteten Tankstelle. Dieser wird dann in die Zuordnung eingetragen.

Dieses Verfahren wird für alle Tankstellen iterativ durchgeführt, bis der optimale Weg zum Streckenende Teil der Zuordnung ist. Dieser ist dann der optimale Weg vom Start zum Streckenende.

1.4 Ermittlung des Preises für einen Weg

Für die Ermittlung des Preises für einen Weg werden vorerst die Tankstellen dem Preis nach aufsteigend geordnet. Zuerst wird von der preiswertesten Tankstelle aus eine Strecke definiert, die von der Tankstelle aus über die maximalen Tanklänge reicht, oder wenn das Streckenende in dieser enthalten ist, bis zum Ziel geht.

Danach wird für die nächst-preiswerteste Tankstelle auch eine solche Strecke definiert, es sei denn, es gibt Überschneidungen mit einer bereits eingetragenen Strecke. In diesem Fall wird die bereits eingetragene Strecke durch maximales Volltanken des halb-leeren Tanks an dieser Tankstelle erweitert.

Dieses Verfahren wird so lange wiederholt, bis nur noch eine Strecke existiert, die den gesamten Weg abdeckt. Dieses Verfahren liefert das ideale Ergebnis, da preislich aufsteigend immer die beste Teillösung gefunden wird.

2 Umsetzung

2.1 Urlaubsfahrt

Die `public static class Urlaubsfahrt` enthält die Hauptmethode `public static Track FindBestTrack`.

`FindBestTrack` erhält als Parameter

- eine Liste von allen GasStations,
- ein Car,
- die Streckenlänge.

2.2 GasStation

Das `public readonly struct GasStation` repräsentiert eine Tankstelle.

Es enthält

- die Position, also Distanz vom Ursprung in Kilometer,
- den Preis.

2.3 Car

Das `public readonly struct Car` repräsentiert die Fahrdaten des Autos und ist nur für die Datenspeicherung zuständig.

Es enthält

- den Verbrauch,

- die Tankkapazität,
- die Strecke, die man mit vollem Tank zurücklegen kann,
- wie viel Benzin am Anfang im Tank ist,
- die Strecke, die man mit dem Startbenzin zurücklegen kann.

2.4 Track

Das `public readonly struct Track` repräsentiert einen Wegabschnitt bis zu einer Tankstelle.
Es enthält

- eine Liste an Tankstellen, an denen gehalten werden kann,
- die Funktion `public readonly DrivingPlan? GetCheapestPathTo`, die für die Liste an Tankstellen die günstigste Tank-Verteilung berechnet, ohne auf die Stopp-Anzahl zu achten.

2.5 DrivingPlan

Das `public readonly struct DrivingPlan` stellt wie `Track` einen Fahrplan dar, jedoch speichert es anders als `Track` auch, wieviel bei jeder Tankstelle getankt werden soll.
Es enthält:

- eine Liste von Tupeln von einer Tankstelle und der Distanz, für die bei ihr getankt wird,
- die Funktion `public readonly decimal PriceFor`, die den Preis für den Weg für ein gegebenes `Car` zurückgibt.

3 Beispiele

Im Folgenden wird das Programm immer mit den zu Anfang aufgelisteten Argumenten aufgerufen. Für mehr Info zu den Parametern des Programms führen sie `Urlaubsfahrt.CLI --help` aus.

```
Urlaubsfahrt.CLI -f "examples/fahrt1.txt"

Stops:
  Gas station(0m 0EUR/l)
  Gas station(100m 1,45EUR/l)
  Gas station(400m 1,4EUR/l)

Driving Plan:
  Drive for 275m on the starting fuel
  Tank 48,00l at Gas station(400m 1,4EUR/l). (Cost: 67,20EUR)
  Tank 10,00l at Gas station(100m 1,45EUR/l). (Cost: 14,50EUR)

  Stops: 2
  Price: 81,70EUR
```

```
   Urlaubsfahrt.CLI -f "examples/fahrt2.txt"

Stops:
  Gas station(0m 0EUR/l)
  Gas station(1118m 1,15EUR/l)
  Gas station(1922m 1,17EUR/l)
  Gas station(2762m 1,19EUR/l)
  Gas station(3833m 1,18EUR/l)
  Gas station(4874m 1,19EUR/l)
  Gas station(5796m 1,17EUR/l)
  Gas station(6912m 1,31EUR/l)
  Gas station(7929m 1,15EUR/l)
```

```
  Gas station(8987m 1,21EUR/l)

Driving Plan:
  Drive for 1137,5m on the starting fuel
  Tank 269,32l at Gas station(1118m 1,15EUR/l). (Cost: 309,72EUR)
  Tank 274,00l at Gas station(7929m 1,15EUR/l). (Cost: 315,10EUR)
  Tank 192,96l at Gas station(1922m 1,17EUR/l). (Cost: 225,76EUR)
  Tank 274,00l at Gas station(5796m 1,17EUR/l). (Cost: 320,58EUR)
  Tank 274,00l at Gas station(3833m 1,18EUR/l). (Cost: 323,32EUR)
  Tank 184,64l at Gas station(2762m 1,19EUR/l). (Cost: 219,72EUR)
  Tank 197,12l at Gas station(4874m 1,19EUR/l). (Cost: 234,57EUR)
  Tank 223,04l at Gas station(8987m 1,21EUR/l). (Cost: 269,88EUR)
  Tank 237,92l at Gas station(6912m 1,31EUR/l). (Cost: 311,68EUR)

  Stops: 9
  Price: 2530,33EUR
```

```
    Urlaubsfahrt.CLI -f "examples/fahrt3.txt"

Stops:
  Gas station(0m 0EUR/l)
  Gas station(465m 1,24EUR/l)

Driving Plan:
  Drive for 533,333m on the starting fuel
  Tank 80,00l at Gas station(465m 1,24EUR/l). (Cost: 99,20EUR)

  Stops: 1
  Price: 99,20EUR
```

```
    Urlaubsfahrt.CLI -f "examples/fahrt4.txt"

Stops:
  Gas station(0m 0EUR/l)
  Gas station(264m 1,2EUR/l)
  Gas station(607m 1,3EUR/l)

Driving Plan:
  Drive for 370m on the starting fuel
  Tank 88,2l at Gas station(264m 1,2EUR/l). (Cost: 105,84EUR)
  Tank 100,8l at Gas station(607m 1,3EUR/l). (Cost: 131,04EUR)

  Stops: 2
  Price: 236,88EUR
```

```
Urlaubsfahrt.CLI -f "../examples/fahrt5.txt"

Stops:
  Gas station(0m 0EUR/l)
  Gas station(107m 1,16EUR/l)
  Gas station(1198m 1,15EUR/l)
  Gas station(2344m 1,17EUR/l)
  Gas station(3454m 1,16EUR/l)
  Gas station(4569m 1,17EUR/l)
  Gas station(5531m 1,31EUR/l)
  Gas station(6692m 1,34EUR/l)
  Gas station(7798m 1,16EUR/l)
  Gas station(8944m 1,31EUR/l)
```

```
Driving Plan:
  Drive for 200m on the starting fuel
  Tank 244,00l at Gas station(1198m 1,15EUR/l). (Cost: 280,60EUR)
  Tank 209,58l at Gas station(107m 1,16EUR/l). (Cost: 243,11EUR)
  Tank 244,00l at Gas station(3454m 1,16EUR/l). (Cost: 283,04EUR)
  Tank 244,00l at Gas station(7798m 1,16EUR/l). (Cost: 283,04EUR)
  Tank 229,76l at Gas station(2344m 1,17EUR/l). (Cost: 268,82EUR)
  Tank 234,15l at Gas station(4569m 1,17EUR/l). (Cost: 273,96EUR)
  Tank 202,02l at Gas station(5531m 1,31EUR/l). (Cost: 264,65EUR)
  Tank 218,42l at Gas station(8944m 1,31EUR/l). (Cost: 286,13EUR)
  Tank 232,07l at Gas station(6692m 1,34EUR/l). (Cost: 310,97EUR)

  Stops: 9
  Price: 2494,32EUR
```

4 Quellcode

```
using System;
using System.Collections.Generic;
using System.Linq;

public static class Urlaubsfahrt
{
    /// <summary>
    /// Findet den kürzesten und preiswertesten Weg durch die Tankstellen. Priorität hat eine gerine Stoppzahl, nicht der Preis
    /// </summary>
    /// <param name="allStations">Alle Tankstellen, sie können ungeordnet sein.</param>
    public static Track FindBestTrack(
        IEnumerable<GasStation> allStations,
        Car car,
        decimal destinationPosition)
    {
        // Der erste Weg der existiert ist ein Weg ohne Stopps
        List<Track> optimalSubTracks = new List<Track> { Track.Empty };

        // Kopiere die übergebenen Stationen in eine eigene, sortierte Liste
        var actualStations = allStations
                .OrderBy(x => x.Position)
                .ToList();

        // Füge das Ziel als eine Tankstelle mit einem Preis von null an
        actualStations.Add(new GasStation(destinationPosition, 0));

        // Für jede Tankstelle wird der ideale Weg ermittelt
        foreach (GasStation station in actualStations)
        {
            optimalSubTracks.RemoveAll(x =>
                // Zuerst werden alle Wege entfernt, dessen letzte Station weiter von der aktuellen Station weg ist, als das Auto Reichweite hat
                station.Position - x.LastStop.Position > car.TankDistance);

            if (optimalSubTracks.Count == 0)
            {
                // Wenn man zu eiem Punkt nicht kommen kann, kann man auch nicht zu den Punkten dahinter oder zum Ziel kommen
```

```
                throw new InvalidOperationException("No solutions found");
            }

            optimalSubTracks.AddRange(optimalSubTracks
                // An jeden alten Weg wird die aktuelle Station angehangen
                .Select(x => x.With(station))
                // Es wird für jeden Weg der Preis gebildet
                .Select(x => (Track: x, Price: x.GetCheapestPriceTo(station, car)))
                // Es werden die nicht möglichen entfernt
                .Where(x => x.Price.HasValue)
                // Es werden die Wege gewählt, die am wenigsten Stopps brauchen
                .AllMinsBy(x => x.Track.Stops.Count)
                // Es werden die günstigsten gwählt
                .AllMinsBy(x => x.Price!.Value)
                .Select(x => x.Track));
        }

        return optimalSubTracks
            .Select(x => (Track: x, Price: x.GetCheapestPriceTo(destinationPosition, car)))
            .Where(x => x.Price.HasValue)
            .AllMinsBy(x => x.Track.Stops.Count)
            .AllMinsBy(x => x.Price!.Value)
            .Select(x => x.Track)
            // Wenn es mehrere gleich-günstige Wege gibt wird der letzte gewählt
            .Last();
    }
}
```

```
using System;
using System.Diagnostics.CodeAnalysis;

public readonly struct GasStation : IEquatable<GasStation>
{
    /// <summary>
    /// Der Startpunkt mit Position und Preis gleich 0.
    /// </summary>
    public static readonly GasStation Home = new GasStation(0, 0);

    /// <summary>
    /// Der Preis in Euro pro Liter.
    /// </summary>
    public readonly decimal Price;

    /// <summary>
    /// Die Position der Tankstelle
    /// </summary>
    public readonly decimal Position;

    public GasStation(decimal position, decimal price)
    {
        Price = price;
        Position = position;
    }

    public override readonly bool Equals(object? obj) => obj is GasStation station &&
        Equals(station);

    public readonly bool Equals([AllowNull] GasStation other) => Price == other.Price &&
```

```
        Position == other.Position;

    public override readonly int GetHashCode() => HashCode.Combine(Price, Position);

    public static bool operator ==(GasStation left, GasStation right) =>
        left.Equals(right);

    public static bool operator !=(GasStation left, GasStation right) => !(left == right);

    public override readonly string ToString() => $"Gas station({Position}m {Price}EUR/l)";
}
```

```
using System;
using System.Collections.Generic;
using System.Collections.Immutable;
using System.Diagnostics.CodeAnalysis;
using System.Linq;

public readonly struct Track : IEquatable<Track>
{
    public static readonly Track Empty = new
        Track(ImmutableList<GasStation>.Empty.Add(GasStation.Home));

    public readonly ImmutableList<GasStation> Stops;

    public readonly GasStation LastStop => Stops[Stops.Count - 1];

    private Track(ImmutableList<GasStation> stops) => Stops = stops;

    public readonly Track With(GasStation newEnd) => new Track(Stops.Add(newEnd));

    public readonly decimal? GetCheapestPriceTo(GasStation destination, Car car) =>
        GetCheapestPriceTo(destination.Position, car);

    public readonly decimal? GetCheapestPriceTo(decimal destination, Car car) =>
        GetCheapestPathTo(destination, car)?.PriceFor(car);

    public readonly DrivingPlan? GetCheapestPathTo(decimal destination, Car car)
    {
        DrivingPlan drivingPlan = DrivingPlan.Empty;

        //Wenn wir mit dem vorhandenen Tank bis zum Ziel fahren können, müssen wir nicht
            tanken
        if (destination < car.StartingFuelDistance)
        {
            drivingPlan.Add(GasStation.Home, destination);
            return drivingPlan;
        }

        //Das gesamte Startbenzin nutzen
        drivingPlan.Add(GasStation.Home, car.StartingFuelDistance);

        HashSet<Range> coveredRanges = new HashSet<Range>
        {
            new Range(0, car.StartingFuelDistance),
            new Range(destination, destination + car.TankDistance)
        };

```

```
45          foreach (GasStation station in Stops.Where(x => x.Price > 0).OrderBy(x =>
                x.Price)) //Die Tankstellen dem Preis nach überprüfen
46          {
47              Range newRange = new Range(station.Position, station.Position +
                    car.TankDistance);
48
49              decimal distance = car.TankDistance;
50
51              bool startHit = false, endHit = false;
52
53              foreach (var coveredRange in coveredRanges.ToList())
54              {
55                  //Überprüfen ob der Start- oder Endpunkt bereits überdeckt sind t collide
                        with the given range.
56                  bool containsStart = coveredRange.Contains(newRange.Start);
57                  bool containsEnd = coveredRange.Contains(newRange.End);
58
59                  //Wenn der gesamte Bereich überdeckt ist muss nicht getankt werden
60                  if (containsStart && containsEnd)
61                  {
62                      newRange = Range.NaR;
63                      break;
64                  }
65
66                  if (!containsStart && !containsEnd) continue;
67
68                  coveredRanges.Remove(coveredRange);
69
70                  if (containsStart)
71                  {
72                      if (startHit) throw new InvalidOperationException();
73                      startHit = true;
74
75                      distance -= coveredRange.End - newRange.Start;
76                      newRange = new Range(coveredRange.Start, newRange.End);
77                  }
78                  else if (containsEnd)
79                  {
80                      if (endHit) throw new InvalidOperationException();
81                      endHit = true;
82
83                      distance -= newRange.End - coveredRange.Start;
84                      newRange = new Range(newRange.Start, coveredRange.End);
85                  }
86
87                  if (newRange.Length == 0 || distance == 0)
88                  {
89                      newRange = Range.NaR;
90                      break;
91                  }
92              }
93
94              if (newRange.IsNaR) continue;
95
96              coveredRanges.Add(newRange);
97              drivingPlan.Add(station, distance);
98
99              //Wenn nur eine Range existiert, die den gesamten Bereich abdeckt, dann wird an
```

```
                den übrigen Stationen nicht getankt
            if (newRange.Start == 0 && newRange.End >= destination)
            {
                return drivingPlan;
            }
        }

        return null; //Wenn der Weg nicht vollständig Abgedeckt ist, ist dieser Track
            nicht möglich
    }

    public override readonly bool Equals(object? obj) => obj is Track track &&
        Equals(track);

    public readonly bool Equals([AllowNull] Track other) =>
        EqualityComparer<ImmutableList<GasStation>>.Default.Equals(Stops, other.Stops);

    public override readonly int GetHashCode() => HashCode.Combine(Stops);

    public static bool operator ==(Track left, Track right) => left.Equals(right);

    public static bool operator !=(Track left, Track right) => !(left == right);

    public override readonly string ToString() => string.Join(Environment.NewLine,
        Stops.Select(x => " " + x));
}
```

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Linq;
public readonly struct DrivingPlan : IEquatable<DrivingPlan>
{
    public readonly List<(GasStation Station, decimal Distance)> Stops;

    public static DrivingPlan Empty => new DrivingPlan(new List<(GasStation Station,
        decimal Distance)>());

    public DrivingPlan(List<(GasStation Station, decimal Distance)> stops) => Stops =
        stops;

    public readonly decimal PriceFor(Car car) => Stops.Sum(x => x.Distance *
        car.GetPriceForDistanceAt(x.Station));

    public readonly void Add(GasStation station, decimal distance) => Stops.Add((station,
        distance));

    public readonly void Sort() => Stops.Sort((x, y) =>
        x.Station.Position.CompareTo(y.Station.Position));

    public override bool Equals(object? obj) => obj is DrivingPlan plan && Equals(plan);

    public bool Equals([AllowNull] DrivingPlan other) => Stops.SequenceEqual(other.Stops);

    public override int GetHashCode() => HashCode.Combine(Stops);

    public static bool operator ==(DrivingPlan left, DrivingPlan right) =>
        left.Equals(right);
```

```
26
27      public static bool operator !=(DrivingPlan left, DrivingPlan right) => !(left ==
            right);
28
29      public override readonly string ToString() => $"Track ({Stops.Count}) {{
            {string.Join(", ", Stops)} }}";
30
31      public readonly string ToString(Car car) =>
32          " Drive for " + Stops[0].Distance + "m on the starting fuel"
33          + Environment.NewLine
34          + string.Join(Environment.NewLine, Stops
35              .Skip(1) // Skip the home "station"
36              .Select(x => $" Tank {x.Distance * car.FuelUsage}l at {x.Station}. (Cost:
                    {car.GetPriceForDistanceAt(x.Station) * x.Distance}EUR)"));
37  }
```