Lab 08 – Parallel Programming

Aufgabe 1 – Alle Routen finden

Fügen Sie der `Links`-Klasse eine neue Methode hinzu, die für alle bekannten Städte jeweils den kürzesten Weg zu jeder anderen bekannten Stadt mit allen bekannten Verkehrsmitteln berechnet.

Die Signatur der Methode ist wie folgt:

```
public List<List<Link>> FindAllShortestRoutes()
```

Bei 10 bekannten Städten und 3 verschiedenen Verkehrsmitteln ergeben sich damit also $10 \times 10 \times 3 = 300$ zu berechnende Routen. Die Reihenfolge der Resultate spielt keine Rolle.

Hinweis: Für die Berechnung der Routen können Sie auf die Methode `FindShortestRouteBetween` zurückgreifen.

Aufgabe 2 – Parallelisierung

Die Methode aus der Aufgabe 1 ist sehr rechenintensiv und kann bei sequenzieller Ausführung mit vielen Städten sehr lange dauern. Sie sollen daher die Berechnung parallelisieren. Ergänzen Sie die `Links`-Klasse um eine weitere Methode mit der folgenden Signatur:

```
public List<List<Link>> FindAllShortestRoutesParallel()
```

Diese Methode soll dieselben Resultate produzieren wie jene aus Aufgabe 1, jedoch unter Ausnutzung alles Cores. Verwenden Sie ein geeignetes Mittel, um den Algorithmus zu parallelisieren.

Dies können die `Parallel.For` Konzepte (Achtung: welche Loops können parallelisiert werden?), oder mit dem PLINQ Konzept `Enumerable.Range().AsParallel()`

Aufgabe 3 – Test

Auf dem Netzwerkshare finden Sie neue Testdaten (`citiesTestDataLab8.txt` und `linksTestDataLab8.txt`) und Unittests. Integrieren Sie diese in das Projekt und testen Sie Ihre Implementierung.

Hinweise:

1. *Denken Sie daran bei den Testdaten-Files in VisualStudio die «Copy to Output Directory» Option zu setzen.*
2. *Die Ausführung der Unittests kann einige Minuten in Anspruch nehmen.*

Testfragen

- Werden Threads, die mit der `Task`-Klasse gestartet werden, als Foreground- oder Background-Threads ausgeführt?
- Was bedeutet es, wenn
    a) der Haupt-Thread vor dem nebenläufigen Hintergrund-Thread endet
    b) der nebenläufige Hintergrund-Thread vor dem Haupt-Thread endet
- PLINQ parallelisiert die LINQ Ausführung und führt damit zu besserer Performance. Warum werden nicht standardmässig alle LINQ-Statements parallel ausgeführt?
- Was versteht man unter dem Begriff „Work Stealing“?
- In welchen konkreten Beispielen macht es Sinn, die LINQ-Methode `WithDegreeOfParalellism` einzusetzen? Weshalb kann es wünschenswert sein, die Parallelität zu limitieren?
- Was versteht man unter Thread Pooling? Weshalb wird Thread Pooling eingesetzt?
- Wann eignet sich Parallelisierung mit PLINQ? Welche Voraussetzungen müssen erfüllt sein?