

Aufgabe 2: Nummernmerker

Team-ID: 00587

Team-Name: Doge.NET

Bearbeiter dieser Aufgabe:
Nikolas Kilian

19. November 2019

Inhaltsverzeichnis

1	Lösungsidee	1
2	Umsetzung	2
3	Beispiele	2

1 Lösungsidee

Um die optimale Aufteilung zu ermitteln, verwenden wir eine Variation des Knapsack-Algorithmus. Dieser funktioniert wie folgt:

```
1 TeileNummerAuf(nullstellen) {  
2     if ( <Bereits für gleiche Parameter aufgerufen> ) {  
3         return <Bereits errechnetes Ergebnis>;  
4     }  
5  
6     if ( <Zu wenig Stellen zum aufteilen> ) {  
7         return <Fehler>;  
8     }  
9  
10    for (int i in 2..4) {  
11        subAufteilung = TeileNummerAuf(nullstellen.Skip(i));  
12  
13        if ( <subAufteilung Fehler produziert hat> ) continue;  
14  
15        Möglichkeiten.Add([i].Concat(subAufteilung));  
16    }  
17  
18    return Möglichkeiten.Max(aufteilung => BewerteAufteilung(nullstellen, aufteilung));  
19 }  
20  
21 BewerteAufteilung(nullstellen, aufteilung) {  
22     return <Anzahl an führenden Nullstellen in der Aufteilung>;  
23 }
```

Hierbei werden bereits errechnete Ergebnisse global gespeichert, sodass bei mehreren Rechnungen nacheinander die Ergebnisse der vorigen Durchläufe eventuell bei folgenden Rechnungen wiederverwendet werden können.

2 Umsetzung

Für die Umsetzung haben uns für eine Implementierung in C# 8.0 mit .NET Core 3.1 entschieden. Der Sourcecode ähnelt stark dem Pseudocode (siehe Lösungsidee); die Zentrale Methode die den Algorithmus ausführt hat die Signatur `NummerMerkingSolution MerkNummern(ArraySegment<bool> zeros, int minSequenceLength = 2, int maxSequenceLength = 4)`. Hierbei sind `min-/maxSequenceLength` die Minimal-/Maximallängen der einzelnen aufgeteilten Segmente.

Für das Speichern alter Ergebnisse wird ein struct `MarkedNummer` verwendet, welches die Eingaben für die Methode zwischenspeichert, und ein struct `NummerMerkingSolution`, welches die Ergebnisse zwischenspeichert. Diese werden in einem `System.Collections.Generic.Dictionary`2` aufeinander gemappt, sodass immer einer `MarkedNummer` eine `NummerMerkingSolution` zugeordnet ist.

Um Rechenzeit zu sparen, wird anders als im Pseudocode kein modifiziertes Array zurückgegeben, sondern eine Instanz des structs `System.ArraySegment`1`. Alle Instanzen dieses structs zeigen beim Ausführen auf das gleiche Array, womit unnötige Array-Allocations verhindert werden, was Kosten des Garbage collectors spart.

3 Beispiele

```

1 Starting splitting of number 005480000005179734 with segments of length 2..4
2   Digits: 18
3 Results:
4   Leading zeros hit: 2
5   Final distribution: 0054 8000 0005 1797
6
7 Starting splitting of number 03495929533790154412660 with segments of length 2..4
8   Digits: 23
9 Results:
10  Leading zeros hit: 1
11  Final distribution: 0349 5929 5337 9015 441 26
12
13 Starting splitting of number 5319974879022725607620179 with segments of length 2..4
14   Digits: 25
15 Results:
16  Leading zeros hit: 0
17  Final distribution: 5319 9748 7902 2725 6076 201
18
19 Starting splitting of number 9088761051699482789038331267 with segments of length 2..4
20   Digits: 28
21 Results:
22  Leading zeros hit: 0
23  Final distribution: 9088 7610 5169 9482 7890 3833 12
24
25 Starting splitting of number 011000000011000100111111101011 with segments of length 2..4
26   Digits: 30
27 Results:
28  Leading zeros hit: 3
29  Final distribution: 0110 0000 001 1000 1001 1111 110 10
30 \end{lstlisting}
31
32 \section{Quellcode}
33
34 \begin{lstcs}
35 private static readonly Dictionary<MarkedNummer, NummerMerkingSolution> MarkedNumbers =
36     new Dictionary<MarkedNummer, NummerMerkingSolution>();
37
38 public static NummerMerkingSolution MerkNummern(ArraySegment<bool> zeros, int
    minSequenceLength, int maxSequenceLength) =>

```

```
39     MerkNummern(new MerkedNumber(zeros, minSequenceLength, maxSequenceLength));
40
41 private static NummerMerkingSolution MerkNummern(MerkedNumber merkedNumber)
42 {
43     if (MerkedNumbers.TryGetValue(merkedNumber, out var optimalDistribution)) return
44         optimalDistribution;
45
46     if (merkedNumber.Zeros.Count < merkedNumber.MinSequenceLength)
47     {
48         return MerkedNumbers[merkedNumber] = NummerMerkingSolution.Failure();
49     }
50
51     int nextGenerationSize =
52         Math.Min(
53             merkedNumber.Zeros.Count,
54             merkedNumber.MaxSequenceLength + 1)
55             - merkedNumber.MinSequenceLength;
56
57     if (nextGenerationSize <= 0)
58     {
59         return MerkedNumbers[merkedNumber] = NummerMerkingSolution.Empty();
60     }
61
62     var nextGeneration = new NummerMerkingSolution[nextGenerationSize];
63
64     for (int i = 0; i < nextGenerationSize; i++)
65     {
66         int length = i + merkedNumber.MinSequenceLength;
67
68         var subSolution =
69             MerkNummern(
70                 new ArraySegment<bool>(
71                     merkedNumber.Zeros.Array,
72                     merkedNumber.Zeros.Offset + length,
73                     merkedNumber.Zeros.Count - length),
74                     merkedNumber.MinSequenceLength,
75                     merkedNumber.MaxSequenceLength);
76
77         nextGeneration[i] = !subSolution.IsSuccessful
78             ? NummerMerkingSolution.Failure()
79             : NummerMerkingSolution.Success(
80                 subSolution.Distribution.PrecedeOne(length),
81                 subSolution.LeadingZerosHit
82                 + (((IList<bool>)merkedNumber.Zeros)[0] ? 1 : 0));
83     }
84
85     var elements = nextGeneration.WhereF(x => x.IsSuccessful);
86
87     if (elements.Length == 0)
88     {
89         return MerkedNumbers[merkedNumber] = NummerMerkingSolution.Failure();
90     }
91
92     if (elements.Length == 1)
93     {
94         return MerkedNumbers[merkedNumber] = elements[0];
95     }
96 }
```

```
96     NummerMerkingSolution bestSolution = elements.AggregateF((x, y) => x.LeadngZerosHit <  
    y.LeadngZerosHit ? x : y);  
97     return MerkedNummers[merkedNummer] = bestSolution;  
98 }
```