

Junioraufgabe 1

33.Bundeswettbewerb Informatik 2014/'15

Der Script-Tim

Inhaltsverzeichnis

1	Zwei Strategien finden	2
1.1	Strategie A	2
1.2	Strategie B	2
2	Entwurf eines Programmes	3
2.1	Allgemeines Problem	3
2.2	Grober Programmablauf	3
2.3	Wahl der Programmiersprache	3
2.4	Programmablauf	3
2.4.1	Laden der Fahrzeugdaten	3
2.4.2	Verarbeitung der Daten	4
2.4.3	Strategie A	5
2.4.4	Strategie B	6
2.4.5	Ausgabe der Parkspuren	7
3	Anwendung auf gegebene Beispiele	7

1 Zwei Strategien finden

Meine zwei Strategien zur Zuweisung der Autos in die Parkbahnen sind:

1.1 Strategie A

Der Fährbegleiter füllt die Parkbahnen abwechselnd von links nach rechts; Schema:

Spur 1	Spur 2	Spur 3
Auto 1	Auto 2	Auto 3
Auto 4	Auto 5	Auto 6

Folglich gilt: $aktuelleSpur = (autonummer \bmod 3) + 1$, wenn keine Bahn übersprungen wird. Die aktuelle Bahn wird übersprungen, wenn sie das aktuell zuzuweisende Auto nicht aufnehmen kann. Kann keine Bahn das Auto aufnehmen, fährt die Fähre ab.

1.2 Strategie B

Das aktuelle Auto wird immer auf die am kürzesten beparkte Spur gewiesen (\rightarrow die Bahn mit dem meisten freien Platz). Sind alle Spuren gleichermaßen beparkt (z.B. am Anfang; alle 0m), oder sind zwei oder mehr Bahnen gleichermaßen beparkt und gleichzeitig die kürzesten, wird die linkeste zugewiesen. Passt das zuzuweisende Auto nicht auf die kürzeste Spur, passt es logischer Weise auch auf keine andere und die Fähre legt ab.

Die Strategien weisen in bestimmten Situationen unterschiedlich viele Autos zu.

2 Entwurf eines Programmes

2.1 Allgemeines Problem

Das zugrunde liegende Problem hierbei besteht in der Ausführung der erdachten Verteilungsstrategien sowie der Verwaltung der Fahrzeuge und Parkspuren; letzterem Problem soll hier mit OOP zu Leibe gerückt werden.

2.2 Grober Programmablauf

Grob ist das Programm in vier Schritte aufgeteilt:

1. Einlesen der Fahrzeugdaten
2. Zuweisen der Fahrzeuge auf die Parkspuren nach Strategie A
3. Zuweisen der Fahrzeuge auf die Parkspuren nach Strategie B
4. Ausgeben der Parkspuren gegeneinander aufgestellt

Interessant sind hierbei natürlich 2 und 3. Sowohl 2 als auch 3 starten die Bearbeitung - analog zur Realität - am Anfang der Fahrzeugschlange und "fertigen" die Autos eins nach dem anderen ab; natürlich kann die Abfertigung an jeder Stelle abgebrochen werden (wenn die Fähre voll ist). Die Parkspuren werden als dynamisches Array oder Container repräsentiert, denen das jeweilige Auto dann zugewiesen wird. Die genaue Zuweisung ist dann Sache der jeweiligen Strategie. Danach werden die Parkspuren in einer Tabelle gegeneinander aufgestellt.

2.3 Wahl der Programmiersprache

Als Programmiersprache habe ich die C++-ähnliche Scriptsprache PHP gewählt.

Info

Zur Ausführung eines PHP-Scriptes wird ein Webserver benötigt; für den Fall, dass Sie gerade keinen zur Hand haben, habe ich das Script auf meinen geladen:

<http://www.tim-hollmann.de/BwInf/Junioraufgabe1/index.php>. Dass das Script nicht nachträglich bearbeitet wurde, beweist der in den HTML-Quelltext eingebettete MD5-Hashcode, den das Script bei jeder Ausführung von sich selbst erzeugt; bei einer kleinsten Veränderung des Scriptes würde sich der Hash grundlegend verändern; der Hash des finalen Scriptes lautet **f705874975b2734e8c8709e63b7307fb**.

2.4 Programmablauf

2.4.1 Laden der Fahrzeugdaten

Zunächst werden die Fahrzeugdaten geladen; hierbei kommen mehrere mögliche Datenquellen zum Einsatz:

- Laden der Daten aus einer auf dem Server befindlichen Textdatei (Ich habe alle Beispiele aus der Angabe entsprechend beigefügt)

```
43 || $daten = file_get_contents($_GET["file"]);
44 || $source_description = "Benutzerdefinierte Datenquelle '<strong>".$_GET["file"]."
||     </strong>' geladen.";
```

- Direktes Übergeben benutzerdefinierter Daten über das HTML-Formular; der Benutzer verwendet hier die exakte Syntax aus der Angabe

```
51 || $daten = $_GET["custom"];
52 || $source_description = "Übergebene Datenquelle '".$_GET["custom"]."'.";
```

- Zufällige Daten; der Benutzer fordert den Server auf, zufällige Fahrzeugdaten zu erzeugen

```
55 || $runden = (rand()%15) +5; //5-19 Autos in der Warteschlange
56 || $daten = "";
57 ||
58 || for ($x = 0; $x <= $runden; $x++){
59 ||     $daten .= (empty($daten) ? "" : ";"); //Trenn-Semikolon (nicht, wenn erstes
||         Auto)
60 ||     $daten .= ((rand()%10)+2)); //Vorkommazahl; [2-11]
61 ||     $daten .= "."; //"Komma"; hier gleich in amerikanischer Schreibweise
62 ||     $daten .= rand()%9 +1; //1.Nachkommastelle; [0-9]
63 ||     $daten .= rand()%9 +1; //2.Nachkommastelle; [0-9]
64 ||     //Fahrzeuge sind 2.00 bis 11.99 Meter lang
65 || }
66 || $source_description = "Zufällige Werte wurden generiert.";
```

Fordert der Benutzer keine Datenquelle explizit an, wird automatisch das erste Beispiel aus der Angabe verwendet.

2.4.2 Verarbeitung der Daten

Die geladenen Daten werden überprüft und evtl. korrigiert (Leerzeichen entfernt, amerikanische in europäische Dezimalpunktierung umgewandelt).

```
65 || //Dezimalpunktierung von (,) zu (.) ändern -> Zahlen können sowohl
66 || //in amerikanischer als auch in europäischer Dezimalschreibweise übergeben werden
67 || $daten = preg_replace("|,|", ".", $daten);
68 ||
69 || //evtl. Leerzeichen entfernen
70 || $daten = preg_replace("| |", "", $daten);
```

Bis zu diesem Punkt war *\$daten* für PHP nur ein String, jetzt wird daraus ein *double*-Array indem es an den Semikolons (oder Semikolen?) getrennt wird:

```
73 || $daten = explode(";", $daten);
```

Anschließend wird sicherheitshalber überprüft, ob die übergebenen Daten alle numerisch sind:

```

76 foreach($daten as $satz){
77     if (!(is_numeric($satz))){
78         Die("Fehler: Datensatz ".$satz." ist nicht numerisch!");
79     }
80 }

```

Um die Fahrzeuge besser verwalten zu können, definiere ich die Klasse *fahrzeug*:

```

23 class fahrzeug{
24     public $nummer;
25     public $laenge;
26
27     function __construct($nummer, $laenge){
28         $this->nummer = $nummer;
29         $this->laenge = $laenge;
30     }
31 };

```

Die Fahrzeugschlange wird von einem dynamischen Array *\$fahrzeuge* des Typs *fahrzeug* repräsentiert;

```

78 //Fahrzeugschläge definieren...
79 $fahrzeuge = array();
80 //... und füllen
81 for ($i = 0; $i <= sizeof($daten)-1; $i++){
82     $fahrzeuge[] = new fahrzeug($i, $daten[$i]);
83 }

```

Für die Parkspuren definiere ich ebenfalls eine eigene Klasse:

```

18 class spur{
19     public $fahrzeuge = array(); //Fahrzeuge auf der Spur
20     public $belegt = 0; //Belegter Platz
21 };

```

Ein Mehrdimensionales Array verwaltet die Parkspuren; [0][0..2] für Strategie A, [1][0..2] für Strategie B:

```

84 $spur = array(array(new spur, new spur, new spur), array(new spur, new spur, new spur));

```

2.4.3 Strategie A

Strategie A setzt die Zuweisung der Spuren über mehrere Fahrzeuge hinweg fort; folglich muss die Information der zuletzt zugewiesenen Parkspur irgendwie zur nächsten Zuweisung weitergegeben werden, ich verwende die Variable *\$position*. Alle Fahrzeuge werden von vorne nach hinten abgefertigt;

```

90 $i = 0;
91 $ende = false;
92 $position = 0;
93
94 while ($ende == false && $i < sizeof($fahrzeuge)){
95     //Ist überhaupt eine Spur vorhanden, auf die das Auto passen würde?

```

```

96     if (!passt($spur[0][0], $fahrzeuge[$i]) && !passt($spur[0][1], $fahrzeuge[$i]) &&
97         !passt($spur[0][2], $fahrzeuge[$i])){ $ende = true; }
98     //Wenn die aktuelle Spur das Auto aufnehmen kann, wird das Auto zugewiesen
99     if (passt($spur[0][$position], $fahrzeuge[$i])){
100         $spur[0][$position]->fahrzeuge[] = $fahrzeuge[$i];
101         $spur[0][$position]->belegt += (($spur[0][$position]->belegt == 0) ? 0:
102             abstand) + $fahrzeuge[$i]->laenge;
103         $i++;
104     }
105     //Zur nächsten Spur weitergehen
106     $position = ($position + 1)%3;
107 }

```

Die Funktion *passt* überprüft, ob das in Parameter 2 übergebene Auto noch auf die in Parameter 1 übergebene Spur passen würde. Zunächst wird überprüft, ob überhaupt eine Spur das Auto aufnehmen kann (Z.96); ansonsten wird die Schleife beendet und die Fähre "fährt ab". Wenn das aktuelle Auto *\$fahrzeuge[\$i]* auf die aktuelle Spur *\$spur[0][\$position]* passt (Z.98), wird es der Parkspur zugewiesen (Z.99+100) und zum nächsten Auto gewechselt (Z.101). Das Wechseln zur nächsten Spur ist unabhängig vom Zuweisen des Fahrzeuges und bewirkt so, dass, wenn das Auto nicht auf die Spur passt, trotzdem zur nächsten Spur gewechselt wird.

2.4.4 Strategie B

Auch Strategie B geht vom Anfang der Warteschlange aus vor. Für Strategie B werden die Parkspuren zunächst nach belegtem Platz sortiert: ein temporäres Array *\$temp* wird mit den Platzwerten gefüllt und anschließend mit der PHP-eigenen Funktion *sort()* sortiert:

```

115 $temp = array($spur[1][0]->belegt, $spur[1][1]->belegt, $spur[1][0]->belegt);
116 sort($temp);

```

Info

sort() sortiert die Einträge eines Arrays; die kleinsten Einträge kommen mit dem Index nach unten, sodass der kleinste Wert *\$temp[0]* sein sollte. Sind zwei Werte gleich groß, werden sie in der Reihenfolge zurückgegeben, wie sie ursprünglich übergeben waren. Da die Spuren von links nach rechts übergeben werden (siehe oben Z.115), sind diese im Zweifelsfall auch die kürzesten → entsprechend der Strategie.

Man hat nun den größten freien Parkplatz ermittelt - jetzt wird die dazugehörige Parkspur gesucht:

```

119 $spurKurz = NULL; //Spur mit den wenigsten "verparkten" Metern
120
121 switch ($temp[0]){
122     case $spur[1][0]->belegt:
123         $spurKurz = 0;
124         break;
125     case $spur[1][1]->belegt:
126         $spurKurz = 1;

```

```

127 |         break;
128 |     case $spur[1][2]->belegt:
129 |         $spurKurz = 2;
130 |         break;
131 |     default:
132 |         Die("Fehler");
133 | }

```

Der Index der Spur mit dem meisten freien Parkplatz ist also nun in $\$spurKurz$. Passt das aktuelle Auto $\$i$ auf die Spur, wird es der Spur zugewiesen; wenn nicht, fährt die Föhre ab (logisch: wenn das Auto nicht auf die kürzest beparkte Parkspur passt, passt es auch auf keine andere):

```

135 | if (passt($spur[1][$spurKurz], $fahrzeuge[$i])){
136 |     $spur[1][$spurKurz]->fahrzeuge[] = $fahrzeuge[$i];
137 |     $spur[1][$spurKurz]->belegt += (($spur[1][$spurKurz]->belegt == 0) ? 0: abstand)
138 |         + $fahrzeuge[$i]->laenge;
139 | }else{
140 |     $ende = true;
141 | }
142 | $i++;

```

2.4.5 Ausgabe der Parkspuren

Nachdem also beide Strategien ausgeführt wurden, „verpackt“ das Script die Spuren beider Strategien in eine HTML-Tabelle und sendet diese zum Client. Wie genau dies passiert, ist hier unerheblich, da es sich um bloßes Einsetzen von Werten in eine HTML-Tabelle handelt.

3 Anwendung auf gegebene Beispiele

(Leider wollte L^AT_EX hier nicht so, wie ich wollte.)

In bestimmten Situationen nimmt Strategie A mehr Autos auf (vgl.Abb.5), in anderen B (vgl.Abb.6).

Junioraufgabe #1 :: Der Script-Tim

Datenquelle

Beispiele aus der Angabe

Beispiel #3 [fahrzeuge_03.txt]

Anfordern

Benutzerdefinierte Dateneingabe

6,96; 5,06; 3,77; 4,95

Absenden

Zufällige Werte

Anfordern

Aktuell verwendete Datenquelle:

Benutzerdefinierte Datenquelle 'fahrzeuge_03.txt' geladen.

Warteschlange: 11 Auto(s)	Strategie A			Strategie B		
Fahrzeuge verlassen (% von wartenden)	9 (82%)			8 (73%)		
Freier Platz insg.[m]	2.78			7.19		
	Spur 1 [2 Autos]	Spur 2 [3 Autos]	Spur 3 [4 Autos]	Spur 1 [3 Autos]	Spur 2 [3 Autos]	Spur 3 [2 Autos]
	[0](5.23m)	[1](4.41m)	[2](3.33m)	[0](5.23m)	[1](4.41m)	[2](3.33m)
	[3](13.13m)	[4](9.12m)	[5](4.38m)	[5](4.38m)	[4](9.12m)	[3](13.13m)
		[7](5.37m)	[6](6.34m)	[6](6.34m)	[7](5.37m)	
			[8](4.11m)			

Abbildung 1: Beispielhafte Darstellung im Browser

Warteschlange: 13 Auto(s)	Strategie A			Strategie B		
Fahrzeuge verlassen (% von wartenden)	11 (85%)			11 (85%)		
Freier Platz insg.[m]	8.22			8.22		
	Spur 1 [3 Autos]	Spur 2 [4 Autos]	Spur 3 [4 Autos]	Spur 1 [3 Autos]	Spur 2 [4 Autos]	Spur 3 [4 Autos]
	[0](6.96m)	[1](5.06m)	[2](3.77m)	[0](6.96m)	[1](5.06m)	[2](3.77m)
	[3](3.95m)	[4](3.91m)	[5](3.54m)	[5](3.54m)	[4](3.91m)	[3](3.95m)
	[6](4.26m)	[7](4.03m)	[8](5.43m)	[8](5.43m)	[7](4.03m)	[6](4.26m)
		[9](4.04m)	[10](4.43m)		[10](4.43m)	[9](4.04m)

Abbildung 2: Anwendung auf Beispiel 1

Warteschlange: 8 Auto(s)	Strategie A			Strategie B		
Fahrzeuge verlassen (% von wartenden)	7 (88%)			7 (88%)		
Freier Platz insg.[m]	25.77			25.77		
	Spur 1 [3 Autos]	Spur 2 [2 Autos]	Spur 3 [2 Autos]	Spur 1 [3 Autos]	Spur 2 [2 Autos]	Spur 3 [2 Autos]
	[0](4.14m)	[1](3.63m)	[2](3.92m)	[0](4.14m)	[1](3.63m)	[2](3.92m)
	[3](7.95m)	[4](5.23m)	[5](3.30m)	[5](3.30m)	[3](7.95m)	[4](5.23m)
	[6](4.86m)			[6](4.86m)		

Abbildung 3: Anwendung auf Beispiel 2

Warteschlange: 11 Auto(s)	Strategie A			Strategie B		
Fahrzeuge verlassen (% von wartenden)	9 (82%)			8 (73%)		
Freier Platz insg.[m]	2.78			7.19		
	Spur 1 [2 Autos]	Spur 2 [3 Autos]	Spur 3 [4 Autos]	Spur 1 [3 Autos]	Spur 2 [3 Autos]	Spur 3 [2 Autos]
	[0](5.23m)	[1](4.41m)	[2](3.33m)	[0](5.23m)	[1](4.41m)	[2](3.33m)
	[3](13.13m)	[4](9.12m)	[5](4.38m)	[5](4.38m)	[4](9.12m)	[3](13.13m)
		[7](5.37m)	[6](6.34m)	[6](6.34m)	[7](5.37m)	
			[8](4.11m)			

Abbildung 4: Anwendung auf Beispiel 3

Warteschlange: 13 Auto(s)	Strategie A			Strategie B		
Fahrzeuge verladen (% von wartenden)	6 (46%)			7 (54%)		
Freier Platz insg.[m]	19.53			10.56		
	Spur 1 [2 Autos]	Spur 2 [2 Autos]	Spur 3 [2 Autos]	Spur 1 [2 Autos]	Spur 2 [3 Autos]	Spur 3 [2 Autos]
	[0](6.49m)	[1](3.76m)	[2](4.23m)	[0](6.49m)	[1](3.76m)	[2](4.23m)
	[3](5.66m)	[4](10.11m)	[5](9.32m)	[5](9.32m)	[3](5.66m)	[4](10.11m)
					[6](8.67m)	

Abbildung 5: Zufälliges Beispiel [6.49; 3.76; 4.23; 5.66; 10.11; 5.93; 8.67]