39. Bundeswettbewerb Informatik, 1. Runde

Jan Ehehalt, Jonathan Hager

Inhaltsverzeichnis

1 Wörter aufräumen

1.1 Lösungsidee

Es werden zwei Listen angelegt, die jeweils alle Lücken und alle vollständigen Wörter getrennt speichern. Das Programm geht die Liste der Lücken durch und fügt für jede Lücke, der sich nur ein Wort eindeutig zuordnen lässt, das entsprechende Wort ein. Dieser Durchlauf wird so oft wiederholt, bis entweder alle Lücken gefüllt sind, oder in einem Durchlauf kein Wort mehr eingefügt werden konnte. Diese Lücken müssen anhand der Länge dem entsprechenden Wort zugeordnet werden. Dieses Verfahren muss funktionieren, da in jede Lücke nur ein Wort passen kann.

1.2 Umsetzung

2 Dreieckspuzzle

3 Tobis Turnier

4 Streichholzrätsel

Im Folgenden sieht man ein Codebeispiel für Code, der beispielhaft geschrieben wurde.

```
// Kommentare sind ne geile Sache
import com.badlogic.ShitRenderer;

public static void main(String[] args){
    Controller.control(theWorld);
    Math.atan(1/0);
}
```

Im Codebeispiel sieht man, wie der JavaScript Compiler intern arbeitet. Besondere Achtung sollte hierbei dem Math-Befehl gegeben werden, denn Math wurde nicht importiert und deshalb crasht es bereits deshalb, nicht wegen der ZeroDivision, da JavaScript hier einfach das Ergebnis würfeln wurde. # Ehre

5 Wichteln

5.1 Lösungsidee

Im Wesentlichen ist das Problem der Zuordnung mit dem *Stable Marriage Problem* vergleichbar. Es gibt zwei verschiedene Gruppen, die so gut wie möglich verteilt werden müssen. Deshalb lässt sich die Aufgabe mit einer leicht angepassten Variante des *Gale-Shapley* Algorithmus lösen. Allgemein fragen alle Schüler ohne Geschenk bei den

Geschenken an, ob diese noch keinen Partner haben oder den neuen Partner dem aktuellen vorziehen. Dabei wird ein Erstwunsch allem vorgezogen, dann folgen Zweitwunsch, Drittwunsch und zuletzt eine Zuteilung ohne Wunsch. Jeder Schüler fragt nacheinander seine Wünsche an. Sollte er danach noch kein Geschenk haben, versucht er ein übriges Geschenk zu bekommen.

5.2 Umsetzung

Student
hasGift: boolean index: int presentId: int wishes: int[] asked: boolean[]
Student(int[], int, int) requestPresent(Present[], Student[], int, int): void

Abbildung 1: Die Klasse *Student*

Jeder Schüler wird durch ein Objekt der Klasse *Student* repräsentiert. Dabei muss jeder Schüler speichern, ob er bereits ein Geschenk hat. Zudem weiß jeder Schüler, welche Geschenk er als Erst-, Zweit- und Drittwunsch will. Für den Algorithmus ist es notwendig, dass jeder Schüler zusätzlich seine Position im Array und alle Geschenke, die er schon versucht hat zu bekommen, kennt. Der Schüler braucht *requestPresent()* als einzige Methode. Übergeben bekommt er zum einen den *Present* und *Student* Array, zum anderen den Index des entsprechenden Geschenks, sowie den Grad des Wunsches. Hierbei hat ein Erstwunsch den Grad 0, ein Zweitwunsch 1, ein Drittwunsch 2 und eine reine Zuteilung 3. Diesen Grad braucht das Geschenk, um den neuen Schüler mit dem möglicherweise aktuellen Schüler zu vergleichen und zu ermitteln, welchen Schüler es bevorzugt.

Present
studentId: int wish: int
Present() changeStudent(students: Student[], wish: int, studentId: int): boolean

Abbildung 2: Die Klasse *Present*

Die Geschenke werden von der Klasse *Present* repräsentiert. Ein Geschenk muss hierfür wissen, an welchen Schüler es vergeben ist. Dazu wird der Index des Schülers ge-

speichert. Um zwei Schüler vergleichen zu können, muss zudem bekannt sein, welchen Grad der Wunsch des aktuellen Schülers hat. Dieser wird als *int* gespeichert. Damit der Algorithmus funktioniert, muss das Geschenk seinen Schüler wechseln können. Hierfür wird der *Student* Array übergeben, sowie der Grad des Wunsches vom neuen Schüler und der Index des neuen Schülers. Die Methode gibt durch einen *boolean* zurück, ob der übergebene Schüler übernommen oder abgelehnt wurde. Die Methode arbeitet wie folgt:

```
1  boolean changeStudent(Student[] students, int wish, int studentId) {
2     if(this.wish > wish){
3        if(this.studentId >= 0) {
4           students[this.studentId].hasGift = false;
5        }
6
7        this.studentId = studentId;
8        this.wish = wish;
9        students[studentId].hasGift = true;
10
11       return true;
12    }
13    else {
14       return false;
15    }
16 }
```

Abbildung 3: Die Methode *changeStudent()*

Zuerst wird verglichen, ob der Grad des gespeicherten Wunsches größer ist, als der des neuen Schülers. Ein kleinerer Grad ist immer einem größeren vorzuziehen. Ist der Grad des neuen Wunsches also nicht echt kleiner als der bisherige, gibt die Methode direkt *false* zurück, da der neue Schüler abgelehnt wird. Andernfalls soll der neue Schüler jedoch den alten ersetzen. Hierfür wird dem alten Schüler erst mitgeteilt, dass er kein Geschenk mehr hat. Davor muss geprüft werden, ob es überhaupt einen alten Schüler gibt. Gibt es keinen, hat *studentId* den Standardwert -1, weshalb dieser Fall abgefangen werden muss. Anschließend werden die Werte des neuen Schülers übernommen, ihm wird mitgeteilt, dass er nun ein Geschenk hat und die Methode gibt *true* zurück. Der Algorithmus selbst ist in zwei Phasen unterteilt:

1. Jeder Schüler versucht seinen Erstwunsch zu bekommen.

2. Jeder Schüler fragt pro Durchlauf ein Geschenk, dass er noch nicht versucht hat, an. Hierbei werden nacheinander die Wünsche favorisiert. Sind bereits alle probiert wurden, werden einfach alle restlichen Geschenke versucht.

Phase 1 findet nur beim ersten Durchlauf durch alle Schüler statt. Danach wird in Phase 2 übergegangen und solang über alle Schüler iteriert, bis jeder ein Geschenk bekommen hat.

Endet die Schleife, wurde die bestmögliche Verteilung erreicht. Der gesamte Quellcode des Algorithmus ist in Abbildung 4 zu finden.

5.3 Beispiele

Anhand eines selbsterstellten, kleinen Beispiels wird hier die Funktion des Algorithmus dargestellt. Anschließend werden alle Ergebnisse der vorgegebenen Beispielaufgaben aufgeführt.

Schüler	Erstwunsch	Zweitwunsch	Drittwunsch
A	1	2	3
B	2	3	4
C	2	1	4
D	4	1	3

Tabelle 1: Beispielhafte Verteilung der Wünsche

Zuerst wird das Beispiel aus Tabelle 1 vom Programm eingelesen. Dabei wird für jeden Schüler, sowie für jedes Geschenk ein Objekt erstellt. Danach beginnt der Algorithmus direkt mit der Verteilung der Geschenke. Im ersten Durchlauf versucht jeder Schüler seinen Erstwunsch zu bekommen. Die Schleife beginnt mit Schüler A. Da sein Erstwunsch frei ist, bekommt er Geschenk 1 zugeteilt. Dasselbe passiert bei Schüler B, dieser bekommt folglich Geschenk 2 zugeteilt. Der Erstwunsch von Schüler C ist bereits vergeben und da C es mit einem Erstwunsch haben will, es aber bereits an einen Erstwunsch vergeben ist, wird Schüler C abgelehnt. Schüler D bekommt Geschenk 4 als Erstwunsch zugeteilt. Jetzt wird in Phase 2 gewechselt. Schüler A und B werden übersprungen, da beide bereits ein Geschenk haben. Schüler C hat seinen Erstwunsch bereits angefragt und probiert deshalb jetzt seinen Zweitwunsch. Dieser ist ebenfalls bereits vergeben und da Schüler C mit einem Zweitwunsch anfragt, wird er abgelehnt. D wird ebenfalls übersprungen. Im nächsten Durchlauf wird ebenfalls jeder außer C übersprungen. Dieser fragt nun seinen Drittwunsch an, welcher bereits an einen höheren Wunsch vergebenen ist. Somit beginnt erneut ein neuer Durchlauf der Schleife, bei dem C nun irgendein Geschenk, das er noch nicht probiert hat, versucht. Es bleibt nur Geschenk 3 übrig, welches er zugeteilt bekommt. Damit wurde jedem Schüler ein Geschenk zugeteilt. Die ausgegebene Verteilung lässt sich Tabelle 2 entnehmen.

Schüler	Geschenk
A	1
B	2
C	3
D	4

Tabelle 2: Das Ergebnis des Programms

```java
// Phase 1
for(int i=0; i<students.length; i++) {
    students[i].requestPresent(presents, students, students[i].wishes[0], 0);
}

// Phase 2
boolean finished = false;
do {
    // Die Schleife ist prinzipiell immer fertig, außer es wird noch ein
        Schüler gefunden, der kein Geschenk hat
    finished = true;
    for(int i = 0; i < students.length; i++) {
        if(!students[i].hasGift) {
            finished = false;

            // Es wird versucht, einen der Wünsche zu erfüllen
            if(!students[i].asked[students[i].wishes[0]]) {
               students[i].requestPresent(presents, students,
                   students[i].wishes[0], 0);
            } else if(!students[i].asked[students[i].wishes[1]]) {
               students[i].requestPresent(presents, students,
                   students[i].wishes[1], 1);
            } else if(!students[i].asked[students[i].wishes[2]]) {
               students[i].requestPresent(presents, students,
                   students[i].wishes[2], 2);
            } else {
                // Da bereits alle Wünsche probiert wurden, wird versucht
                    irgendein Geschenk zu bekommen
               for(int j = 0; j < presents.length; j++) {
                   if(!students[i].asked[j]) {
                       students[i].requestPresent(presents, students, j, 3);
                   }
                }
            }
        }
    }
} while(!finished);
```

Abbildung 4: Die Implementierung des Algorithmus