39. Bundeswettbewerb Informatik

Runde 1

Aufgabe 5: Wichteln

07. November 2020

Inhaltsverzeichnis

Lösungsidee 2
Umsetzung 5
Beispiele 10
Quellcode 19

Lösungsidee

Nach der Anforderung der Aufgabe sollen in einer guten Verteilung möglichst viele 1. Wünsche erfüllt werden. D. h., alle Gegenstände, die als den 1. Wunsch eines bzw. mehrerer Schüler sind, müssen auch den Schülern, die das am besten (1. Wunsch) gefällt, verteilt werden.

Sollte ein Gegenstand von mehreren Schülern als den 1. Wunsch genannt, wird die Wahrscheinlichkeit, ob die anderen Wünsche eines Schülers auch später (als 2. oder 3. Wunsch) erfüllt werden können, für jeden diesen Schüler vermutet bzw. gerechnet. Man gibt den Gegenstand demjenigen, dessen Wahrscheinlichkeit am niedrigsten ist. Sind die Wahrscheinlichkeiten gleich, wird ein Schüler zufällig gewählt. Es ist sinnvoll, denn die anderen Schüler können noch andere Gegenstände bekommen, die ihren 2. bzw. 3. Wunsch erfüllt. Zumindest ist die Wahrscheinlichkeit, dass ihr 2. oder 3. Wunsch erfüllt werden können, höher als diesen gewählten Schüler. Somit werden am meisten 1. Wünsche erfüllt, während die Wahrscheinlichkeit, dass der 2. oder 3. Wunsch der anderen Schüler erfüllt werden können, am höchsten ist.

Diese oben beschriebene Wahrscheinlichkeit wird mit folgender Formel berechnet:

Lass

$$a_2 = \begin{cases} 1, wenn\ der\ 2.\ erwünschte\ Gegenstand\ nicht\ vorher\ gegeben\ wird \\ 0, wenn\ der\ 2.\ erwünschte\ Gegenstand\ den\ anderen\ Schülern\ als\ 1.\ Wunsch\ gegeben\ wird \end{cases}$$

Bzw.

$$a_2 = \begin{cases} 1, wenn\ der\ 2.\ Wunsch\ \text{erfüllbar ist} \\ 0, wenn\ der\ 2.\ Wunsch\ nicht\ erfüllbar\ ist \end{cases}$$

$$A_{2,2} = Anzahl\ der\ Schüler, die\ den\ 2.\ Wunsch\ des\ Schülers\ als\ ihren\ 2.\ Wunsch\ nennen$$

$$A_{2,3} = Anzahl\ der\ Schüler, die\ den\ 2.\ Wunsch\ des\ Schülers\ als\ ihren\ 3.\ Wunsch\ nennen$$

$$a_3 = \begin{cases} 1, wenn\ der\ 3.\ erwünschte\ Gegenstand\ nicht\ vorher\ gegeben\ wird \\ 0, wenn\ der\ 3.\ erwünschte\ Gegenstand\ schon\ als\ 1.\ oder\ 2.\ Wunsch\ gegeben\ wird \end{cases}$$

$$A_{3,3} = Anzahl\ der\ Schüler, die\ den\ 3.\ Wunsch\ des\ Schülers\ als\ ihren\ 3.\ Wunsch\ nennen$$

$$A_{3,2} = Anzahl\ der\ Schüler, die\ den\ 3.\ Wunsch\ des\ Schülers\ als\ ihren\ 2.\ Wunsch\ nennen$$

So erhält man:

$$P_1 = \begin{cases} \frac{10000}{A_{2,2}} - A_{2,3}\ ,\ wenn\ \frac{10000}{A_{2,2}} - A_{2,3} \geq 1 \\ 1,\ wenn\ \frac{10000}{A_{2,2}} - A_{2,3} < 1 \end{cases}$$

$$P_2 = \begin{cases} \frac{100}{A_{3,3}} - 10 \cdot A_{3,2},\ wenn\ \frac{100}{A_{3,3}} - 10 \cdot A_{3,2} \geq 1 \\ 1,\ wenn\ \frac{100}{A_{3,3}} - 10 \cdot A_{3,2} < 1 \end{cases}$$

$$P = P_1 \cdot a_2 + P_2 \cdot a_3$$

Die Idee hinter der Formel ist:

1. Die Wahrscheinlichkeit P_1, dass der 2. Wunsch des Schülers erfüllt werden kann, hat einen Anfangswert von *10000*. Sollte die anderen Schüler den gleichen 2. Wunsch wie diesen Schüler haben, wird diese *10000* durch die Anzahl dieser Schüler mit gleichem Wunsch ($A_{2,2}$) dividiert, denn es kann auch sein, dass dieser gewünschte Gegenstand an den anderen Schülern als deren 2. Wunsch verteilt wird, welche die P_1 des jetzigen Schülers reduziert. Hier ergibt sich die Division Sinn, weil die Wahrscheinlichkeit, dass einer dieser Schüler den Gegenstand bekommt, an dieser Stelle gleich ist.
   Außerdem werden die Schüler, deren 3. Wunsch gleich ist wie den 2. Wunsch des jetzigen Schülers, in der Berechnung von P_1 betrachtet ($A_{2,3}$). Hier wird diese Anzahl $A_{2,3}$ von der P_1 nach der Division subtrahiert, weil es zwar noch möglich ist, dieser Gegenstand als einen 3. Wunsch zu geben, jedoch erfüllt man lieber einen 2. Wunsch als den 3. Wunsch.

   Falls der erwünschte Gegenstand schon als den 1. Wunsch eines anderen Schülers genannt wird, beträgt die Koeffizient a_2 0, damit beträgt die Wahrscheinlichkeit, dass der 2. Wunsch des jetzigen Schülers erfüllt werden kann, auch 0. Ansonsten bleibt die Wahrscheinlichkeit P_1 immer positiv, weil sich diese Wahrscheinlichkeit nicht ausschließen lässt.

2. Die Wahrscheinlichkeit P2, dass der 3. Wunsch des jetzigen Schülers erfüllt werden kann, lässt sich mit ähnlichem Verfahren berechnen. Allerdings wird die Anzahl der Schüler $A_{3,2}$, die den gleichen 2. Wunsch haben wie den 3. Wunsch des jetzigen Schülers, mit 10 multipliziert und dann von P_2 subtrahiert. Der Grund für die Multiplikation ist, dass man lieber einen 2. Wunsch mit dem Gegenstand erfüllt als einen 3. Wunsch.
   Auch wird die Koeffizient a_3 0, wenn der gewünschte Gegenstand bereits als den 1. oder 2. Wunsch gegeben wird.

3. Die gesamte Wahrscheinlichkeit für diesen Schüler ist dann die Summe von $P_1 \cdot a_2$ und $P_2 \cdot a_3$.

Für den 2. Wunsch geht man ähnlich vor. Man gibt alle Gegenstände in der 2. Reihe (also als 2. Wunsch), die noch nicht als 1. Wunsch geschenkt werden und von den Schülern, deren 1. Wunsch noch nicht erfüllt werden, erwünscht sind, an diesen Schülern. Dadurch kann die maximale Anzahl der erfüllten 2. Wünsche erreicht werden.

Sollte nochmal ein Gegenstand von mehreren Schülern als 2. Wunsch genannt werden, wird die obengenannten Wahrscheinlichkeit mit ähnlichem Verfahren berechnet und der Gegenstand dem Schüler gegeben, der die niedrigste Wahrscheinlichkeit hat. Der einzige Unterschied liegt daran, dass P_1 nicht mehr betrachtet wird, denn jetzt wird entschieden, wessen 2. Wunsch erfüllt werden soll und die Wahrscheinlichkeit für die Erfüllbarkeit eines zweiten Wunsches sinnlos wird.

Die Formel lautet jetzt also:

$$P = P_2 \cdot a_3$$

Für den 3. Wunsch wird nichts mehr berechnet. Sollte es noch offener 3. Wunsch geben (also Wünsche von den Schülern, deren 1. und 2. Wunsch nicht erfüllt sind und der gewünschte Gegenstand noch nicht an anderen Schüler gegeben ist), wird dieser direkt erfüllt, weil es keinen Unterschied mehr macht, auch wenn möglicherweise mehrere gleiche Wünsche existieren. Nur einer davon können erfüllt, und andere nicht. Dabei spielt die Reihfolge aber keine Rolle, denn man braucht nur die maximale Anzahl von erfüllten 3. Wünschen (selbstverständlich unter der Bedingung, dass es zuerst maximale Anzahl von erfüllten 1. und 2. Wünsche gibt).

Am Ende werden die restlichen Gegenstände, die nicht verteilt sind, an den Schülern, die noch keinen Gegenstand erhalten haben, gegeben.

Zur Veranschaulichung wird das oben beschriebene Verfahren mithilfe eines Beispiels dargestellt. Das befindet sich in dem Abschnitt Beispiel.

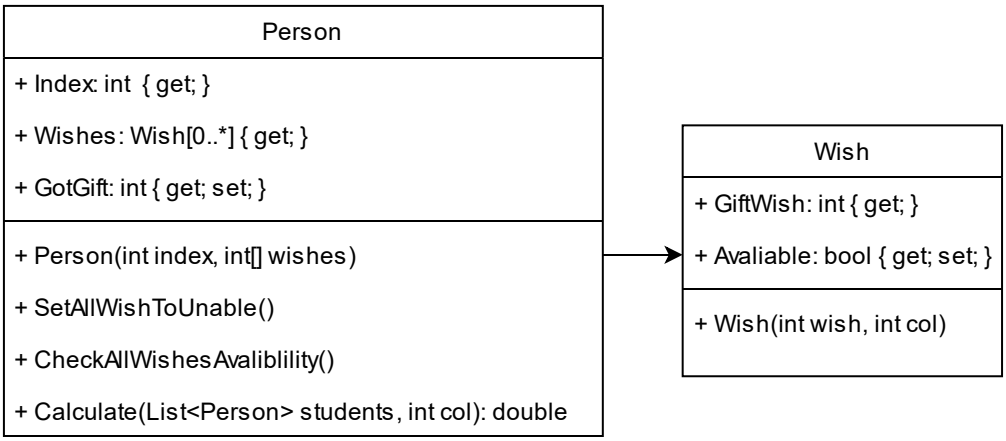
Umsetzung

Die Wichteln-Datei wird zunächst mithilfe von *StreamReader* eingelesen. Der Nutzer soll dabei den Namen der Test-Datei eingeben.

```
Console.WriteLine("Bitte Name der Test-Datei eingeben...");
string testFilePath = Console.ReadLine();
      ……
using (StreamReader sr = File.OpenText(testFilePath))
```

Die Schüler mit ihren jeweiligen Wünschen werden in einer Liste *students* gespeichert, deren Datentyp *Person* ist. Die Klasse *Person* speichert die Nummer des Schülers, dessen drei Wünschen und den Gegenstand, der ihm gegeben werden soll. Darüber hinaus gibt es eine Klasse *Wish*, die ein Wunsch eines Schülers repräsentiert und dabei speichert, welcher Gegenstand erwünscht ist und ob dieser Wunsch erfüllt werden kann.

```
List<Person> students = new List<Person>(studentCount);
// Das Einlesen der Datei wird hier nicht vollständig gezeigt
// Vgl. Quellcode
```



Nachdem alle Wünsche der Schüler eingelesen werden, wird noch eine Zuordnungstabelle *giftsTo* angelegt. Diese Zuordnungstabelle soll speichern, welcher Schüler den Gegenstand bekommt. Hierbei ist der Schlüssel *die Nummer des Gegenstands*, der Datenwert dann *die Nummer des Schülers*, der diesen Gegenstand bekommt.

```
Dictionary<int, int> giftsTo = new Dictionary<int, int>();
```

Danach wird für den 1., 2. und 3. Wunsch geprüft, welche Wünsche erfüllt werden sollen und welche nicht.

Außerdem gibt die Methode *GetAllGivenGiftInCol* an, welche Gegenstände in dieser Reihe (also als 1., 2. oder 3. Wunsch der Schüler) gewünscht sind. Duplikate Werte werden nur einmal zurückgegeben.

```
// Prueft fuer jede Reihe (also der 1. 2. und 3. Wunsch)
processCol(0, ref students, ref giftsTo, GetAllGivenGiftInCol(0, stu-
dents));
processCol(1, ref students, ref giftsTo, giftsTo.Keys.ToArray());
processCol(2, ref students, ref giftsTo, giftsTo.Keys.ToArray());
```

```
// int col: die Reihe zu bearbeiten,
// also ob man den 1. , 2. oder 3. Wunsch pruefen sollte
// giftsTo: Zuordnungstabelle, vgl. Dokumentation
// givenGifts: Die Gegenstaende, die bereits an den
// anderen Schuelern gegeben wurden
private static void processCol(int col, ref List<Person> stu-
dents, ref Dictionary<int, int> giftsTo, int[] givenGifts)
```

In der Methode *ProcessCol* werden alle Wünsche, die nur einmal in der Reihe (als 1., 2. oder 3. Wunsch) vorkommen, direkt erfüllt, sofern die erfüllbar sind. Ein Wunsch ist dann nicht mehr erfüllbar, wenn der erwünschte Gegenstand schon an den anderen Schülern gegeben wurde.

```
# region Teste die Wuensche, die nur einmal in dieser Reihe erscheinen
for (int i = 0; i < studentCount; i++)
{
  int[] allPersonWithSameWish = GetAllPersonIndexWithSameGiftWish(stu-
dents[i].Wishes[col], students, col);
  // Sofern ein Wunsch nur einmal in der Reihe erscheint
  // wird der gewuenschte Gegenstand diesem Schueler gegeben.
  if (allPersonWithSameWish.Length == 1 && allPersonWith-
SameWish[0] == i)
  {
    giftsTo.Add(students[i].Wishes[col].GiftWish, students[i].Index);
    students[i].GotGift = students[i].Wishes[col].GiftWish;
    students[i].SetAllWishToUnable();
  }
}
```

Dabei gibt die Methode *GetAllPersonIndexWithSameGiftWish(Wish giftWish, List<Person> students, int col)* die Nummer der Schüler zurück, deren col-1´st Wunsch gleich ist wie den gegebenen gewünschten Gegenstand “giftWish“.

Die Methode *SetAllWishToUnable* setzt alle Wünsche dieses Schülers zu „nicht erfüllbar“. Also werden die Attribute *Avaliable* der allen drei Wünschen des Schülers zu *false* gesetzt.

Nachdem alle dieser Wünsche erfüllt sind, wird für jeden Schüler geprüft, welche der restlichen Wünsche des Schülers noch erfüllbar sind. Falls ein Wunsch nicht mehr erfüllbar ist, wird sein Attribute *Avaliable* zu *false* gesetzt (vgl. Klassendiagramm).

```
for (int idx = 0; idx < studentCount; idx++)
{
  students[idx].CheckAllWishesAvaliblility(givenGifts.Un-
ion(giftsTo.Keys).ToArray(), col + 1);
}
# endregion
```

Danach werden die Wünsche, die mehrmals in dieser Reihe (*col*) vorkommen, geprüft, und dabei wird entschieden, wessen Wunsch erfüllt werden soll und wessen nicht. Hier wird die Wahrscheinlichkeit berechnet, ob ein anderer Wunsch des Schülers später erfüllt werden kann. Die Formel wurde in der Lösungsidee beschrieben. Der Wunsch von dem Schüler mit der niedrigsten Wahrscheinlichkeit wird dann erfüllt.

```
#region Testet die Wuensche, die mehrmals in dieser Reihe erscheinen
for (int i = 0; i < studentCount; i++)
{
  if (students[i].Wishes[col].Avaliable)
  {
    // Hier wird der Schüler mit der niedrigsten Wahrscheinlichkeit
    // gesucht und seine Nummer
// in die Variable personIndexWithLowestScore gespeichert
// für vollständigen Code vgl. Quellcode
    giftsTo.Add(students[i].Wishes[col].GiftWish, personIndexWith-
LowestScore);
    students[personIndexWithLowestScore].GotGift = stu-
dents[i].Wishes[col].GiftWish;
    students[personIndexWithLowestScore].SetAllWishToUnable();

    for (int idx = 0; idx < studentCount; idx++)
    {
      students[idx].CheckAllWishesAvaliblility(gift-
sTo.Keys.ToArray(), col);
    }
  }
}
#endregion
```

Dabei wird jedes Mal nach einem Erfüllen eines Wunsches für jeden anderen Schüler mithilfe von der Methode *CheckAllWishesAvaliblity* geprüft, welche der restlichen Wünsche noch erfüllbar sind.

Die Methode zur Wahrscheinlichkeitsrechnung heißt *Calculate* (vgl. Klassendiagramm). Die Formel befindet sich in der Lösungsidee und die Implementierung wie folgt:

```
public double Calculate(List<Person> students, int col)
{
  double score1 = 0;
  double score2 = 0;

  // Berechnung von P1
  if (col == 0 && this.Wishes[1].Avaliable)
  {
    // Die Divisionrechnung
    score1 = 10000d / (double)GetAllPersonIndexWithSameGift-
Wish(this.Wishes[1], students, 1).Length;
int appearTimesIn2 = GetAllPersonIndexWithSameGift-
Wish(this.Wishes[1], students, 2).Length;
    // Die Substraktion
    // P1 muss positiv bleiben
    score1 = score1 - appearTimesIn2 > 1 ? score1 - appearTimesIn2 : 1;
  }

  // Berechnung von P2
  if (col < 2 && this.Wishes[2].Avaliable)
  {
    score2 = 100d / (double)GetAllPersonIndexWithSameGift-
Wish(this.Wishes[2], students, 2).Length;
int appearTimesIn1 = GetAllPersonIndexWithSameGift-
Wish(this.Wishes[2], students, 1).Length;
    // P2 muss positiv bleiben
    score2 = score2 - 10 * appearTimesIn1 > 1 ? score2 - 10 * ap-
pearTimesIn1 : 1;
  }

  // Die gesamte Wahrscheinlichkeit
  return score1 + score2;
}
```

Nachdem alle erfüllbaren Wünsche erfüllt wurden, werden die restlichen Gegenstände, die noch nicht gegeben wurden, an den Schüler verteilt, die eben keinen Gegenstand bekommen haben. Hier spielt die Reihfolge keine Rolle, denn die maximale Anzahl von erfüllten 1., 2. und 3. Wünschen wurde mit dem vorherigen Verfahren bereits erreicht und die Verteilung anderer Gegenstände nichts mehr beeinflusst.

Hier wird also zunächst eine zweite Zuordnungstabelle *personWithGift* initialisiert. Der Schlüssel dieser Tabelle ist die Nummer des Schülers und der Wert ist die Nummer des Gegenstands, den dieser Schüler bekommt.

```
Dictionary<int, int> personWithGift = new Dictionary<int, int>();
```

In einer Schleife wird getestet, ob ein Gegenstand bereits einem Schüler zugeordnet wurde bzw. ob ein Schlüssel mit der Nummer des Gegenstands in der Zuordnungstabelle *giftsTo* existiert. Ist es der Fall, werden die Nummer des Schülers, also der Datenwert, als Schlüssel der zweiten Zuordnungstabelle *personWithGift* und die Nummer für den ihm zugeteilten Gegenstand als Datenwert gespeichert. Ist es aber hingegen nicht der Fall, wird dann dieser Gegenstand an einem Schüler gegeben, der noch keinen bekommen hat.

```
#region Teste, welche Geschenke noch nicht gegeben sind und verteilen d
ie
for (int i = 0; i < studentCount; i++)
{
  int personIndex;
  if (giftsTo.TryGetValue(i, out personIndex)) personWithGift.Add(per-
sonIndex, i);
  else
  {
    for (int j = 0; j < students.Count; j++)
    {
      // Falls der Schueler nachher keinen gewuenschten Gegenstand be-
kommen hat
      // wird ein noch nicht gegebener Gegenstand ihm gegeben
      if (students[j].GotGift == -1)
      {
        students[j].GotGift = i;
        giftsTo.Add(i, students[j].Index);
        personWithGift.Add(students[j].Index, i);
        break;
      }
    }
  }
}
#endregion
```

Am Ende wird die Verteilung in der Console gezeigt. Weil der Index mit 0 und die Nummerierung der Schüler mit 1 anfängt, wird hier i+1 gezeigt.

```
for (int i = 0; i < studentCount; i++)
{
    Console.WriteLine($"{i + 1} {personWithGift[i] + 1}");
}
```

Beispiele

Erwartete Formen von Daten

Angabe für das Programm:

Name der Test-Datei sollte nach Aufforderung des Programms eingegeben wird.

Nach der Ausgabe des Ergebnisses wird das Programm den Nutzer auffordern, eine beliebige Taste zu drücken, um das Programm zu schließen.

Ausgabe:

n Zeilen von zwei Integer *S*, *G*, die durch ein Leerzeichen getrennt werden, wobei *n=Anzahl der Schüler*, *S=Nummer der Schüler*, *G=Nummer des Gegenstands*, den dieser Schüler bekommt.

```
1 G1
2 G2
...
n-1 Gn-1
n Gn
```

Beispiel 1, vorgegeben, wichteln1.txt

Ausgabe:

```
1 6
2 2
3 1
4 3
5 5
6 4
7 7
8 10
9 9
10 8
```

Erklärung

Zunächst werden alle 1. Wünsche, die nur einmal erscheinen, direkt erfüllt. In diesem Fall sind hier die grün markierten Zahlen.

```
10
 2 10  6
 2  7  3
 4  7  1
 3  4  9
 3  7  9
 4  3  2
 7  6  2
10  2  4
 9  8  1
 4  9  6
```

Weil die drei Wünsche der Schüler bereits erfüllt sind, werden die weiteren Wünsche dieser Schüler dann nicht mehr erfüllt. Also markieren wir ihre 2. und 3. Wünsche als nicht erfüllbar (hier zur visuellen Darstellung markieren wir sie mit ~~grün Durchstreichen~~).

Außerdem müssen alle Gegenstände, die als den 1. Wunsch genannt sind, auch als 1. Wunsch verteilt, sodass eine maximale Anzahl von erfüllten 1. Wünschen erreicht werden kann. D. h., die als 1. Wunsch genannten Gegenstände werden nicht als 2. Oder 3. Wunsch eines Schülers gegeben. Insofern werden solche 2. und 3. Wünsche, deren gewünschten Gegenstände als den 1. Wunsch genannt werden, auch als nicht erfüllbar markiert (zur visuellen Darstellung markieren wir sie mit ~~blau Durchstreichen~~). Sie sind in diesem Fall 2, 4, 3, 7, 10 und 9.

```
10
 2 10  6
 2  7  3
 4  7  1
 3  4  9
 3  7  9
 4  3  2
 7  6  2
10  2  4
 9  8  1
 4  9  6
```

Danach werden für die Wahrscheinlichkeit berechnet. Zuerst betrachten wir die Wünsche nach dem Gegenstand *2*. Diese Wünsche sind von Schüler *1* und *2*.

Für Schüler *1* beträgt die Wahrscheinlichkeit P_{S1}:

$$P_{S1} = 0 + \left(\frac{100}{2} - 0\right) \cdot 1 = 50$$

Und für Schüler *2* die Wahrscheinlichkeit P_{S2}:

$$P_{S2} = 0 + 0 = 0$$

Weil $P_{S2} < P_{S1}$, wird der Wunsch des zweiten Schülers *S2* erfüllt. Solche erfüllten Wünsche markieren wir mit blau. Der 1. Wunsch des Schülers *S1* wird somit auch nicht mehr erfüllbar.

Analog vergleicht man die Wahrscheinlichkeiten der Schüler *3*, *6*, und *10*, die sich den Gegenstand *4* wünscht und der Schüler *4* und *5*, die sich den Gegenstand *3* wünscht.

Da $P_{S4} = P_{S5,}$ wird hier ein Schüler zufällig gewählt zwischen *S4* und *S5*, in diesem Fall dann *S4*.

```
10
-2 10  6
 2  7 -3
 4- 7  1
 3  4  9
-3  7  9
 4  3  2
 7  6 -2
10  2 -4
 9  8 -1
-4  9  6
```

Nachdem alle 1. Wünsche bearbeitet wurden, fängt man mit den 2. Wünschen an. **In der Regel** soll hier genau dasselbe Verfahren wie oben durchgeführt werden. Da in diesem Fall kein erfüllbarer 2. Wunsch mehr vorhanden ist, fängt man dann mit den 3. Wünschen an.

Hier werden alle 3. Wünsche, die noch erfüllbar (nicht durchgestrichen) sind, direkt erfüllt. Sofern es mehrere gleiche Wünsche gibt, wird einer zufällig gewählt. Die erfüllte 3. Wünsche markieren wir hier mit gelb.

```
10
-2 10  6
 2  7 -3
 4- 7  1
 3  4  9
-3  7  9
 4  3  2
 7  6 -2
10  2 -4
 9  8 -1
-4  9  6
```

Die Schüler, die bisher noch keinen erwünschten Gegenstand bekommen haben, werden einen noch freien (nicht gegebenen) Gegenstand verteilt. In diesem Fall sind sie Schüler *5* und *10* und die noch freien Gegenstände sind *5* und *8*.

```
10
~~2~~ ~~10~~  6
 2  ~~7~~ ~~3~~
 ~~4~~ ~~7~~  1
 3  4  ~~9~~
~~3~~  ~~7~~  ~~9~~    <- Gegenstand 5
 4  ~~3~~  ~~2~~
 7  ~~6~~ ~~2~~
10  ~~2~~ ~~4~~
 9  ~~8~~ ~~1~~
~~4~~  ~~9~~  ~~6~~    <- Gegenstand 8
```

Somit erhält man am Ende eine solche Verteilung (mit Farben markiert):

```
1 6
2 2
3 1
4 3
5 5nicht erwünscht
6 4
7 7
8 10
9 9
10 8nicht erwünscht

Grün: direkt erfüllte 1. Wünsche
Blau: nach Vergleich erfüllte 1. Wünsche
Gelb: erfüllte 3. Wünsche
```

Beispiel 2, vorgegeben, wichteln2.txt

Ausgabe:

```
1 4
2 5
3 6
4 1
5 2
6 3
7 7
8 8
9 9
10 10
```

Beispiel 3, vorgegeben, wichteln3.txt

Ausgabe:

```
1 2
2 20
3 29
4 8
5 1
6 3
7 5
8 12
9 4
10 28
11 11
12 13
13 14
14 23
15 26
16 30
17 9
18 7
19 16
20 10
21 19
22 18
23 27
24 15
25 17
26 22
27 24
28 21
29 6
30 25
```

Beispiel 4, vorgegeben, wichteln4.txt

Ausgabe:

```
1 2
2 20
3 29
4 8
5 1
6 3
7 5
8 12
9 4
10 28
11 11
12 13
13 14
14 23
15 26
16 30
17 9
18 7
19 16
20 10
21 19
22 18
23 27
24 15
25 17
26 22
27 24
28 21
29 6
30 25
```

Beispiel 5, vorgegeben, wichteln5.txt

Ausgabe:

```
1 25
2 6
3 7
4 19
5 27
6 2
7 4
8 18
9 5
10 9
11 14
12 13
13 16
14 15
15 10
16 8
17 26
18 23
19 20
20 3
21 1
22 21
23 22
24 11
25 24
26 28
27 30
28 12
29 17
30 29
```

Beispiel 6, vorgegeben, wichteln6.txt

Ausgabe:

```
1 1
2 27
3 53
4 49
5 54
6 21
7 37
8 45
9 55
10 51
11 87
12 65
13 86
14 26
15 38
16 84
17 28
18 80
19 15
20 35
21 6
22 23
23 31
24 2
25 14
26 12
27 74
28 39
29 78
30 25
31 4
32 5
33 7
34 19
35 34
36 36
37 43
38 44
39 42
40 50
41 32
42 83
43 29
44 22
45 81
46 85
47 52
48 56
49 16
50 57
51 40
52 59
53 24
54 60
55 62
56 61
57 64
58 46
59 13
60 66
61 33
62 75
63 68
64 67
65 10
66 69
67 9
68 47
69 70
70 71
71 18
72 90
73 3
74 30
75 72
76 73
77 58
78 11
79 17
80 48
81 76
82 63
83 77
84 79
85 82
86 41
87 20
88 88
89 89
90 8
```

Beispiel 7, vorgegeben, wichteln7.txt

Ausgabe:

Angesichts der Länge der Ausgabe wird diese auf eine Zeile reduziert, wobei jede ursprüngliche Zeile durch ein Semikolon ; getrennt wird.

1 484;2 28;3 5;4 146;5 367;6 662;7 1;8 980;9 531;10 866;11 571;12 309;13 868;14 73;15 854;16 660;17 752;18 2;19 304;20 861;21 3;22 722;23 143;24 58;25 977;26 616;27 994;28 352;29 80;30 960;31 602;32 44;33 745;34 476;35 438;36 347;37 949;38 594;39 85;40 683;41 753;42 6;43 604;44 131;45 455;46 181;47 968;48 318;49 182;50 467;51 705;52 773;53 630;54 9;55 390;56 14;57 743;58 756;59 109;60 471;61 15;62 235;63 696;64 697;65 730;66 885;67 234;68 509;69 138;70 18;71 997;72 561;73 402;74 116;75 686;76 526;77 279;78 891;79 462;80 798;81 480;82 831;83 956;84 121;85 415;86 644;87 677;88 175;89 490;90 99;91 932;92 836;93 475;94 613;95 560;96 126;97 260;98 770;99 156;100 129;101 321;102 233;103 857;104 313;105 206;106 188;107 13;108 769;109 589;110 341;111 620;112 177;113 647;114 63;115 35;116 435;117 910;118 23;119 92;120 24;121 817;122 325;123 151;124 82;125 631;126 715;127 682;128 25;129 168;130 711;131 558;132 899;133 694;134 655;135 31;136 429;137 246;138 202;139 790;140 783;141 933;142 32;143 33;144 893;145 579;146 894;147 37;148 506;149 108;150 39;151 40;152 42;153 781;154 487;155 96;156 731;157 607;158 922;159 671;160 733;161 962;162 420;163 106;164 273;165 75;166 450;167 469;168 46;169 734;170 801;171 495;172 551;173 167;174 684;175 577;176 189;177 902;178 923;179 964;180 48;181 889;182 49;183 52;184 947;185 580;186 564;187 963;188 991;189 919;190 492;191 698;192 54;193 488;194 364;195 489;196 566;197 639;198 45;199 499;200 707;201 338;202 593;203 288;204 818;205 337;206 710;207 383;208 605;209 973;210 708;211 60;212 281;213 74;214 290;215 562;216 10;217 47;218 651;219 934;220 380;221 22;222 64;223 76;224 57;225 68;226 70;227 72;228 209;229 799;230 161;231 512;232 838;233 606;234 633;235 78;236 86;237 535;238 888;239 427;240 534;241 884;242 999;243 548;244 615;245 89;246 91;247 335;248 98;249 155;250 441;251 100;252 628;253 1000;254 556;255 447;256 574;257 656;258 212;259 306;260 621;261 612;262 110;263 350;264 256;265 458;266 998;267 892;268 623;269 211;270 302;271 250;272 20;273 115;274 597;275 117;276 118;277 539;278 692;279 830;280 292;281 376;282 834;283 301;284 410;285 843;286 758;287 229;288 353;289 950;290 414;291 310;292 453;293 59;294 139;295 299;296 120;297 858;298 803;299 137;300 140;301 157;302 7;303 142;304 485;305 918;306 127;307 144;308 145;309 222;310 170;311 354;312 174;313 176;314 326;315 286;316 179;317 135;318 426;319 203;320 395;321 796;322 847;323 180;324 549;325 193;326 674;327 198;328 154;329 95;330 336;331 265;332 134;333 718;334 938;335 208;336 805;337 672;338 215;339 274;340 152;341 986;342 939;343 218;344 220;345 221;346 399;347 224;348 470;349 12;350 164;351 975;352 247;353 196;354 936;355 465;356 849;357 125;358 183;359 227;360 832;361 371;362 625;363 614;364 965;365 228;366 675;367 36;368 553;369 231;370 896;371 879;372 845;373 687;374 543;375 113;376 751;377 424;378 236;379 542;380 445;381 658;382 421;383 204;384 873;385 619;386 237;387 640;388 150;389 725;390 416;391 239;392 241;393 387;394 862;395 820;396 244;397 554;398 191;399 439;400 245;401 356;402 249;403 666;404 253;405 259;406 112;407 34;408 807;409 261;410 263;411 806;412 111;413 267;414 305;415 741;416 343;417 270;418 271;419 851;420 690;421 540;422 945;423 197;424 55;425 714;426 88;427 537;428 272;429 275;430 277;431 122;432 280;433 283;434 287;435 169;436 673;437 289;438 84;439 251;440 21;441 903;442 293;443 294;444 765;445 627;446 136;447 474;448 296;449 300;450 314;451 565;452 524;453 214;454 97;455 311;456 71;457 437;458 160;459 449;460 268;461 315;462 322;463 324;464 327;465 638;466 483;467 329;468 388;469 898;470 365;471 105;472 544;473 330;474 744;475 864;476 331;477 205;478 103;479 443;480 519;481 508;482 676;483 629;484 461;485 217;486 333;487 332;488 94;489 342;490 472;491 344;492 583;493 345;494 663;495 940;496 829;497 466;498 363;499 349;500 187;501 351;502 521;503 511;504 840;505 498;506 357;507 359;508 748;509 652;510 890;511 800;512 360;513 924;514 362;515 875;516 557;517 368;518 428;519 497;520 303;521 941;522 901;523 369;524 370;525 162;526 679;527 653;528 373;529 457;530 87;531 377;532 379;533 382;534 966;535 372;536 517;537 384;538 132;539 185;540 810;541 178;542 51;543 440;544 186;545 386;546 780;547 19;548 269;549 394;550 346;551 276;552 378;553 124;554 65;555 398;556 754;557 913;558 678;559 114;560 576;561 844;562 240;563 400;564 366;565 166;566 403;567 417;568 404;569 133;570 406;571 505;572 412;573 418;574 486;575 943;576 230;577 603;578 432;579 419;580 423;581 786;582 777;583 883;584 56;585 430;586 433;587 552;588 434;589 252;590 431;591 921;592 83;593 361;594 243;595 436;596 223;597 448;598 451;599 738;600 452;601 456;602 931;603 720;604 460;605 525;606 463;607 867;608 732;609 689;610 77;611 464;612 904;613 959;614 473;615 200;616 897;617 478;618 479;619 477;620 409;621 298;622 828;623 481;624 515;625 688;626 491;627 190;628 496;629 422;630 573;631 503;632 972;633 927;634 504;635 81;636 729;637 41;638 43;639 510;640 405;641 514;642 559;643 848;644 601;645 516;646 634;647 522;648 582;649 536;650 26;651 242;652 50;653 523;654 527;655 201;656 53;657 355;658 500;659 529;660 713;661 774;662 530;663 532;664 533;665 778;666 538;667 816;668 396;669 192;670 541;671 446;672 397;673 219;674 766;675 546;676 348;677 680;678 547;679 550;680 482;681 860;682 248;683 567;684 569;685 93;686 454;687 841;688 493;689 62;690 700;691 101;692 958;693 870;694 572;695 581;696 584;697 255;698 586;699 590;700 592;701 706;702 811;703 442;704 591;705 297;706 595;707 596;708 598;709 407;710 599;711 920;712 723;713 312;714 319;715 608;716 320;717 609;718 611;719 254;720 907;721 617;722 528;723 375;724 618;725 328;726 877;727 622;728 624;729 173;730 635;731 226;732 946;733 643;734 153;735 792;736 649;737 645;738 632;739 739;740 648;741 650;742 213;743 232;744 659;745 374;746 661;747 664;748 667;749 808;750 285;751 518;752 668;753 669;754 681;755 691;756 411;757 755;758 610;759 693;760 699;761 701;762 600;763 702;764 172;765 323;766 791;767 494;768 703;769 709;770 712;771 716;772 717;773 984;774 797;775 719;776 724;777 967;778 520;779 727;780 339;781 728;782 735;783 4;784 66;785 626;786 393;787 141;788 737;789 878;790 578;791 641;792 171;793 881;794 740;795 742;796 747;797 912;798 749;799 750;800 988;801 761;802 762;803 763;804 996;805 468;806 69;807 771;808 768;809 772;810 128;811 736;812 408;813 258;814 775;815 507;816 27;817 929;818 779;819 785;820 587;821 788;822 67;823 928;824 148;825 210;826 789;827 11;828 793;829 795;830 809;831 953;832 835;833 804;834 102;835 812;836 813;837 819;838 642;839 38;840 821;841 29;842 787;843 646;844 291;845 784;846 760;847 194;848 685;849 824;850 776;851 308;852 987;853 637;854 8;855 17;856 782;857 926;858 825;859 389;860 880;861 262;862 401;863 159;864 670;865 826;866 340;867 636;868 833;869 839;870 502;871 842;872 501;873 585;874 846;875 850;876 852;877 767;878 853;879 855;880 295;881 184;882 563;883 856;884 225;885 815;886 859;887 284;888 746;889 278;890 863;891 726;892 104;893 865;894 869;895 871;896 872;897 874;898 882;899 908;900 886;901 130;902 358;903 887;904 895;905 695;906 900;907 909;908 149;909 827;910 802;911 316;912 757;913 905;914 665;915 906;916 195;917 545;918 199;919 914;920 307;921 264;922 575;923 207;924 16;925 915;926 916;927 917;928 381;929 413;930 822;931 266;932 814;933 30;934 704;935 119;936 925;937 334;938 930;939 935;940 163;941 937;942 974;943 989;944 942;945 657;946 944;947 948;948 425;949 513;950 568;951 951;952 238;953 969;954 654;955 282;956 90;957 952;958 954;959 955;960 837;961 957;962 158;963 961;964 317;965 123;966 970;967 971;968 976;969 978;970 216;971 385;972 979;973 61;974 147;975 392;976 444;977 981;978 165;979 257;980 982;981 459;982 983;983 876;984 985;985 990;986 570;987 721;988 588;989 107;990 911;991 79;992 992;993 993;994 759;995 823;996 391;997 764;998 555;999 995;1000 794

Quellcode

```
1.  using System;
2.  using System.IO;
3.  using System.Collections.Generic;
4.  using System.Linq;
5.
6.  namespace Aufgabe5
7.  {
8.    public static class WichtelnSortierment
9.    {
10.     public static void Main(string[] args)
11.     {
12.       Console.WriteLine("Bitte Name der Test-Datei eingeben...");
13.       string testFilePath = Console.ReadLine();
14.
15.       if (File.Exists(testFilePath))
16.       {
17.         using (StreamReader sr = File.OpenText(testFilePath))
18.         {
19.           # region Einlesen und Speichern von Beispiel-Daten
20.           int studentCount = Convert.ToInt32(sr.ReadLine());
21.
22.           List<Person> students = new List<Person>(studentCount);
23.           for (int i = 0; i < studentCount; i++)
24.           {
25.             string[] line = sr.ReadLine().Trim().Split(' ');
26.             // Speichert, welche Geschenke sich dieser Schueler wuenscht
27.             int[] wishesWithIndexI = new int[3];
28.             for (int j = 0, wishI = 0; j < line.Length; j++)
29.             {
30.               // Die Eingabedateien sind offenbar nicht nur mit einem Leerze
    ichen getrennt
31.               // bspw. kommt 3  1 10 vor
32.               // Deshalb wird geprueft, nur wenn es wirklich eine Zahl entha
    elt, wird dies dann gespeichert
33.               // -1, weil der Index mit 0 anfaengt
34.               if (!string.IsNullOrWhiteSpace(line[j])) wishesWithIndexI[wish
    I++] = Convert.ToInt32(line[j].Trim()) - 1;
35.             }
36.             students.Add(new Person(i, wishesWithIndexI));
37.           }
38.           # endregion
39.
40.           // key: Gegenstandnummer
41.           // val: die Person, die diesen Gegenstand bekommen hat
42.           Dictionary<int, int> giftsTo = new Dictionary<int, int>();
43.
44.           // Prueft fuer jede Reihe (also der 1. 2. und 3. Wunsch)
45.           // welche der Wuensche erfuellt werden koennen
46.           // und wessen Wunsch genau erfuellt werden soll
47.           // Die erfuellte Wuensche werden in die Zuordnungstabelle giftsTo
    gespeichert
48.           processCol(0, ref students, ref giftsTo, GetAllGivenGiftInCol(0, s
    tudents));
49.           processCol(1, ref students, ref giftsTo, giftsTo.Keys.ToArray());

50.           processCol(2, ref students, ref giftsTo, giftsTo.Keys.ToArray());

51.
52.
53.           // key: die Person, die diesen Gegenstand bekommen hat
54.           // val: Gegenstandnummer
55.           Dictionary<int, int> personWithGift = new Dictionary<int, int>();
```

```
56.
57.           # region Teste, welche Geschenke noch nicht gegeben sind und verte
    ilen die
58.           for (int i = 0; i < studentCount; i++)
59.           {
60.             int personIndex;
61.             if (giftsTo.TryGetValue(i, out personIndex)) personWithGift.Add(
    personIndex, i);
62.             else
63.             {
64.               for (int j = 0; j < students.Count; j++)
65.               {
66.                 // Falls der Schueler nachher keinen gewuenschten Gegenstand
     bekommen hat
67.                 // wird ein noch nicht gegebener Gegenstand ihm gegeben
68.                 if (students[j].GotGift == -1)
69.                 {
70.                   students[j].GotGift = i;
71.                   giftsTo.Add(i, students[j].Index);
72.                   personWithGift.Add(students[j].Index, i);
73.                   break;
74.                 }
75.               }
76.             }
77.           }
78.           # endregion
79.
80.           # region Ausgabe des Ergebnises
81.           for (int i = 0; i < studentCount; i++)
82.           {
83.             // Ausgabe: S G
84.             // S ist die Nummer des/r SchuelerIn,
85.             // G ist die Nummer der Gegenstand, der an dem/der SchuelerIn S
    gegeben wird
86.             // Bsp: 1 3
87.             // Das heisst, der/die Schueler 1 bekommt den Gegenstand 3
88.             Console.WriteLine($"{i + 1} {personWithGift[i] + 1}");
89.           }
90.           # endregion
91.         }
92.       }
93.       else
94.       {
95.         Console.WriteLine("Die gegebene Datei-
    Namen ist nicht gueltig. Das Programm und die Datei muessen in demselben Ord
    ner stehen. ");
96.       }
97.       Console.WriteLine("\r\nDrueck eine beliebige Taste zu schliessen...");

98.       Console.ReadKey();
99.     }
100.
101.
102.           // Prueft fur die gegebenen Reihe col,
103.           // welche der Wuensche erfuellt werden koennen
104.           // und zu wem die Geschenke gegeben werden sollen
105.           // int col: die Reihe zu bearbeiten,
106.           // also ob man den 1. , 2. oder 3. Wunsch pruefen sollte
107.           // giftsTo: Zuordnungstabelle, vgl. Dokumentation
108.           // givenGifts: Die Gegenstaende, die bereits an den
109.           // anderen Schuelern gegeben wurden
110.           private static void processCol(int col, ref List<Person> students
    , ref Dictionary<int, int> giftsTo, int[] givenGifts)
111.           {
112.             int studentCount = students.Count;
113.
```

```
114.             # region Teste die Wuensche, die nur einmal in dieser Reihe ers
    cheinen
115.             for (int i = 0; i < studentCount; i++)
116.             {
117.               int[] allPersonWithSameWish = GetAllPersonIndexWithSameGiftWi
    sh(students[i].Wishes[col], students, col);
118.               // Sofern ein Wunsch nur einmal in der Reihe erscheint
119.               // (die Wuensche von den anderen, deren Wunsch bereits erfuel
    lt wurde,
120.               // wird hier nicht mehr betrachtet)
121.               // wird der gewuenschte Gegenstand diesem Schueler gegeben.

122.               if (allPersonWithSameWish.Length == 1 && allPersonWithSameWis
    h[0] == i)
123.               {
124.                 giftsTo.Add(students[i].Wishes[col].GiftWish, students[i].I
    ndex);
125.                 students[i].GotGift = students[i].Wishes[col].GiftWish;
126.                 students[i].SetAllWishToUnable();
127.               }
128.             }
129.             // Testet, ob die gegebenen Geschenke auch
130.             // von den anderen (mit niedriger Prioritaet) gewuenscht wurden

131.             // Ist so, aendert diese Wuensche (von den anderen Schuelern) a
    ls nicht erfuellbar
132.             for (int idx = 0; idx < studentCount; idx++)
133.             {
134.               students[idx].CheckAllWishesAvaliblility(givenGifts.Union(gif
    tsTo.Keys).ToArray(), col + 1);
135.             }
136.             # endregion
137.
138.
139.             # region Testet die Wuensche, die mehrmals in dieser Reihe ersc
    heinen
140.             for (int i = 0; i < studentCount; i++)
141.             {
142.               // Hier wird die Wahrscheinlichkeit gerechnet,
143.               // ob der Wunsch noch spaeter erfuellt werden kann
144.               // Der Wunsch mit der niedrigsten Wahrscheinlichkeit
145.               // Wird erfuellt
146.               if (students[i].Wishes[col].Avaliable)
147.               {
148.                 int[] allPersonWithSameWish = GetAllPersonIndexWithSameGift
    Wish(students[i].Wishes[col], students, col);
149.                 double lowestScore = double.MaxValue;
150.                 int personIndexWithLowestScore = -1;
151.
152.                 for (int j = 0; j < allPersonWithSameWish.Length; j++)
153.                 {
154.                   if (students[allPersonWithSameWish[j]].Calculate(students
    , col) < lowestScore)
155.                   {
156.                     lowestScore = students[allPersonWithSameWish[j]].Calcul
    ate(students, col);
157.                     personIndexWithLowestScore = allPersonWithSameWish[j];

158.                   }
159.                 }
160.
161.                 giftsTo.Add(students[i].Wishes[col].GiftWish, personIndexWi
    thLowestScore);
162.                 students[personIndexWithLowestScore].GotGift = students[i].
    Wishes[col].GiftWish;
```

```
163.                 students[personIndexWithLowestScore].SetAllWishToUnable();
164.
165.                 // Testet, ob die gegebenen Geschenke auch
166.                 // von den anderen (mit niedriger Prioritaet) gewuenscht wu
    rden
167.                 // Ist so, aendert diese Wuensche als nicht erfuellbar
168.                 for (int idx = 0; idx < studentCount; idx++)
169.                 {
170.                   students[idx].CheckAllWishesAvaliblility(giftsTo.Keys.ToA
    rray(), col);
171.                 }
172.               }
173.             }
174.             # endregion
175.           }
176.
177.
178.           // Methode, die zurueckgibt, welche Schueler
179.           // der ggb. Gegenstand "giftWish" als ihre col-
    1's Wunsch erwuenscht
180.           private static int[] GetAllPersonIndexWithSameGiftWish(Wish giftW
    ish, List<Person> students, int col)
181.           {
182.             List<int> persons = new List<int>();
183.             for (int i = 0; i < students.Count; i++)
184.             {
185.               if (students[i].Wishes[col].GiftWish == giftWish.GiftWish &&
    students[i].Wishes[col].Avaliable) persons.Add(students[i].Index);
186.             }
187.             return persons.ToArray();
188.           }
189.
190.
191.           // Methode, die zurueckgibt,
192.           // welche Gegenstaende in dieser Reihe gewuenscht sind
193.           private static int[] GetAllGivenGiftInCol(int col, List<Person> s
    tudents)
194.           {
195.             List<int> giftNums = new List<int>();
196.             for (int i = 0; i < students.Count; i++)
197.             {
198.               giftNums.Add(students[i].Wishes[col].GiftWish);
199.             }
200.             return giftNums.Distinct().ToArray();
201.           }
202.
203.
204.           // Erweiterungsmethode, die bestimmt,
205.           // ob die ggb. val in die ggb. arr ist
206.           private static bool Contains(this int[] arr, int val)
207.           {
208.             foreach (int i in arr)
209.             {
210.               if (i == val) return true;
211.             }
212.             return false;
213.           }
214.
215.
216.           // Klasse, die ein Wunsch eines Schuelers repraesentiert
217.           private class Wish
218.           {
219.             public int GiftWish
220.             {
221.               get;
```

```
222.             }
223.
224.             // Ob dieser Wunsch erfuellt werden kann
225.             // false, wenn bspw. ein anderer Schueler das Geschenk schon be
    kommt
226.             // Oder, dass ein anderer Wunsch des Schuelers bereits erfuellt
     ist,
227.             // und damit sind alle andere Wuensche des Schuelers nicht erfu
    ellbar
228.             public bool Avaliable
229.             {
230.               get;
231.               set;
232.             }
233.
234.             public Wish(int wish, int col)
235.             {
236.               Avaliable = true;
237.               GiftWish = wish;
238.             }
239.
240.             public static Wish[] ConvertToWishes(int[] wishesOfOnePerson)
241.             {
242.               return new Wish[] { new Wish(wishesOfOnePerson[0], 0), new Wi
    sh(wishesOfOnePerson[1], 1), new Wish(wishesOfOnePerson[2], 2) };
243.             }
244.           }
245.
246.
247.           // Klasse, die ein Schueler repraesetiert
248.           private class Person
249.           {
250.             public int Index
251.             {
252.               get;
253.             }
254.
255.             // Die drei Wuensche des Schuelers
256.             public Wish[] Wishes
257.             {
258.               get;
259.             }
260.
261.             // Der Gegenstand, das dem Schueler gegeben wird
262.             public int GotGift
263.             {
264.               get; set;
265.             }
266.
267.             public Person(int index, int[] wishes)
268.             {
269.               Index = index;
270.               Wishes = Wish.ConvertToWishes(wishes);
271.               GotGift = -1;
272.             }
273.
274.             public void SetAllWishToUnable()
275.             {
276.               for (int i = 0; i < 3; i++) this.Wishes[i].Avaliable = false;

277.             }
278.
279.
280.             // Diese Methode aendert das Attribut "Avaliable"
281.             // von den Wuenschen des Schluelers zu false,
282.             // wenn diese gewuenschte Geschenke schon gegeben sind,
```

```
283.             // welche durch int[] givenGift gegeben wird
284.             public void CheckAllWishesAvaliblility(int[] givenGift, int sta
    rtCol)
285.             {
286.               for (int i = startCol; i < 3; i++)
287.               {
288.                 if (givenGift.Contains(Wishes[i].GiftWish)) Wishes[i].Avali
    able = false;
289.               }
290.             }
291.
292.
293.             // Hier wird gerechnet, wie hoch die Wahrscheinlichkeit ist,
294.             // dass die andere Wuensche dieses Schluelers (also die 2. und
    3. Wuensche)
295.             // spaeter noch erfuellt werden koennen.
296.             // Ausserdem wird geprueft, ob diese anderen Wuensche des Schlu
    elers
297.             // auch von den anderen Schluelern erwuenscht sind.
298.             // Ist es der Fall, dann ist die Wahrscheinlichkeit,
299.             // dass diese Wuensche des Schuelers nicht erfuellt werden koen
    nen,
300.             // dementsprechend hoch, denn die koennen auch von den Anderen
    genommen werden.
301.             // Je hoeher diese Wahrscheinlichkeit ist, dass der 2. bzw. 3.
    Wunsch
302.             // des Schluelers spaeter erfuellt werden kann,
303.             // desto groesser wird die zurueckgegebene Zahl "score".
304.             // Die konkrekte Rechenformel befindet sich in der Dokumentatio
    n.
305.             // Selbstverstaendlich wird der 2. Wuensche der anderen Schuele
    r nicht mehr betrachtet,
306.             // wenn es schon ueber den 2. Wunsch bestimmen wird
307.             public double Calculate(List<Person> students, int col)
308.             {
309.               double score1 = 0;
310.               double score2 = 0;
311.
312.               // Also wird dieser If-Block nicht durchgefuehrt,
313.               // wenn jetzt ueber den 2. Wunsch bestimmt werden soll
314.               // denn es wird schon in processCol gemacht
315.               // und hier ist es unnoetig
316.               if (col == 0 && this.Wishes[1].Avaliable)
317.               {
318.                 score1 = 10000d / (double)GetAllPersonIndexWithSameGiftWish
    (this.Wishes[1], students, 1).Length;
319.                 int appearTimesIn2 = GetAllPersonIndexWithSameGiftWish(this
    .Wishes[1], students, 2).Length;
320.                 score1 = score1 - appearTimesIn2 > 1 ? score1 - appearTimes
    In2 : 1;
321.               }
322.
323.               if (col < 2 && this.Wishes[2].Avaliable)
324.               {
325.                 score2 = 100d / (double)GetAllPersonIndexWithSameGiftWish(t
    his.Wishes[2], students, 2).Length;
326.                 int appearTimesIn1 = GetAllPersonIndexWithSameGiftWish(this
    .Wishes[2], students, 1).Length;
327.                 score2 = score2 - 10 * appearTimesIn1 > 1 ? score2 - 10 * a
    ppearTimesIn1 : 1;
328.               }
329.
330.               // Dementsprechend wird fuer den letzten Wunsch gar nichts ge
    rechnet
331.               // weil es sich kein Sinn mehr ergibt
```

```
332.                 // Falls es in der 3. Reihe noch mehrere Moeglichkeiten gibt,
333.                 // waehlt man dann eins zufaellig aus
334.                 // Das beeinflusst nicht, wie gut diese Verteilung ist
335.                 // (denn alle der 3. Wunsch ist und nur einer erfuellt werden
    kann)
336.
337.                 return score1 + score2;
338.             }
339.           }
340.         }
341.       }
```