Aufgabe 3: Voll Daneben

Team-ID: 00126
Team-Name: GSO
Bearbeiter dieser Aufgabe: Niklas Degener

Inhaltsverzeichnis:

Lösungsidee:

Zuerst werden die gegeben Zahlen in eine geordnete Reihenfolge gebracht. Dann wird der Zahlenbereich von 0 bis zur höchsten Zahl in zehn gleich große Teile unterteilt. Danach wird für jeden der zehn Zahlenbereiche die Zahl genommen, die in der Mitte des Bereichs liegt. Somit sind nun zehn potentielle Gewinnzahlen ausgewählt. Da aber nicht garantiert ist, dass Al Capone so gewinn macht, wird für jede der Gewinnzahlen eine für jeden Durchgang zufällig negative oder positive Abweichung eingeführt.
Diese Abweichung wird von einem bestimmten Wert ausgehend so lange geringer, bis die Auszahlung unter der eingestzten Summe liegt und Al Capone gewinnt macht.

Da aber auch mit dieser Lösung nur relativ geringe Lösungen gefunden werden, wird noch eine Wiederholungskonstante eingeführt. Diese gibt an, wie oft der Vorgang des Findens von Gewinnzahlen wiederholt werden soll, nachdem Al Capone schon Profit macht. Falls bei einer der Wiederholungen eine bessere Lösung gefunden wird, werden die schon durchgeführten Wiederholungen zurückgesetzt und erneut so oft wiederholt, wie die Wiederholungskonstante angibt. Falls keine bessere Lösung gefunden wird, wird das Programm beendet.

Somit lässt sich nun durch die Wiederholungskonstante ein geeigneter Mittelweg mit zufriedenstellenden Lösungen und einer nicht all zu großen Laufzeit des Programms finden.

Umsetzung:

Die beschriebene Lösungsidee wurde in der Programmiersprache Java implementiert.

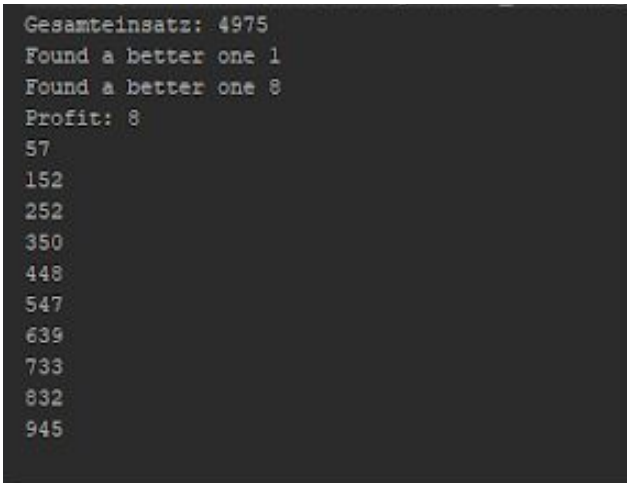
Um ein gefundenes Ergebnis zu speichern wird eine “Ergebnis” Klasse (siehe S.5) verwendet.
Zuerst wird in der main-Methode in der “VollDaneben” Klasse eine Datei, die die Gewinnzahlen enthält eingelesen und sortiert in eine Liste gespeichert.
In der “GewinnRechner” Klasse wird nun die “maxGewinn” methode (siehe S.4) ausgeführt. Diese führt die “getGewinnZahlen” methode (siehe S.5) so lange aus, bis der resultierende Gewinn, der mit der “getGewinn” Methode ermittelt wird, über 0 liegt und ein int “currentReturns” (siehe S.4) , der die Wiederholungen nachdem der Gewinn über 0 liegt speichert, unter dem Wert der Wiederholungskonstante “RETURNS” (siehe S.4) liegt.
Außerdem wird für jede Iteration die Abweichung (siehe S.4) verringert bis sie 0 beträgt, dann wird sie wieder auf 50 gesetzt.
Nach jeder Iteration wird der aus der “getGewinnZahlen” resultierende Gewinn mit dem Gewinn des gepeicherten “Ergebnis” bestErgebnis (siehe S.4) abgeglichen. Liegt der aktuelle Gewinn über dem von “bestErgebnis” wird “bestErgebnis” mit dem aktuellen Ergebnis überschrieben.

Die “getGewinnZahlen” Methode teilt den Zahlenbereich von 0 bis zur höchsten gegebenen Gewinnzahl in 10 Bereiche auf und speichert die Zahl, die jeweils in der Mitte eines Bereichs liegt. Dann wird von jeder dieser Zahlen noch die höchste gegebene Gewinnzahl geteilt durch den aktuellen Wert der Abweichung zufällig abgezogen oder hinzugefügt.

Zuletzt wird der Profit des letzten und besten Ergebnisses ausgegeben und nach Beendung der “maxGewinn” Methode die gefundenen Gewinnzahlen in der main-Methode ausgegeben.

Lösungsbeispiele:

Mögliche Ausgabe des Programms:



Die erste Zeile der Ausgabe gibt den gesamten Einsatz der Spieler an.

Die “Found a better one” Zeilen, werden ausgegeben, wenn das Programm aufgrund der Wiederholungen durch die Wiederholungskonstante, eine bessere Lösung als die vorherige findet. Die Zahl am Ende dieser zeile gibt den Gewinn mit der neuen besseren Lösung an.
Diese Zeilen sind für die Verwendung des Programms relativ irrelevant.

Die Zeile unterhalb der vorher beschrieben Zeilen gibt den Profit der letzten und besten Lösung an.

Die letzen 10 Zeilen der Ausgabe sind die ausgewählten Gewinnzahlen.

Für die in „bwinf.de/fileadmin/user_upload/BwInf/2018/37/1._Runde/beispiel1.txt“ angegeben Zahlen, findet das Programm folgende Lösungen:

Für Widerholungskonstante “RETURNS” = 1000:
Gewinn von 23 mit den Gewinnzahlen: 53, 160, 255, 360, 460, 558, 655, 751, 850, 950
Hohe Laufzeit aufgrund von vielen Wiederholungen.

Für “RETURNS” = 10:
Gewinn von 14 mit den Gewinnzahlen: 57, 165, 262, 362, 454, 551, 649, 748, 844, 950
Niedrigere Laufzeit, dafür aber auch niedrigerer Gewinn.

Für die in „bwinf.de/fileadmin/user_upload/BwInf/2018/37/1._Runde/beispiel2.txt“ angegeben Zahlen, findet das Programm folgende Lösungen:

Für “RETURNS” = 10:
Gewinn von 246 mit den Gewinnzahlen: 65, 169, 235, 363, 429, 544, 671, 759, 833, 942

Da im Programm aber mit strukturiertem Zufall gearbeitet wird, können auch für die gleiche Konstante und gleiche Glückszahlen durchaus unterschiedliche Gewinnzahlen ermittelt und damit auch unterschiedlicher Gewinn erzielt werden.

Quelltext

GewinnRechner.java: Für Verständnis wichtige Variablen

```
public int einsatz = 25;
public int profit = 0;
int abweichung = 50;
final int RETURNS = 10;
int currentReturn = 0;
Ergebnis bestErgebnis = new Ergebnis();
```

GewinnRechner.java:maxGewinn

```
public List<Integer> maxGewinn(List<Integer> inputs){
   int size = inputs.size();
   int gewinn = size * einsatz;
   System.out.println("Gesamteinsatz: " + gewinn);

   while((profit <= 0 || RETURNS > currentReturn)){
       Ergebnis result = new Ergebnis();
       result.setGewinnZahlen(getGewinnZahlen(inputs,
abweichung));

result.setGewinn(getGewinn(inputs,result.getGewinnZahlen()));

       if(abweichung > 1) abweichung -=1;
       else abweichung = 50;

       if(result.getGewinn() > bestErgebnis.getGewinn()) {
           bestErgebnis = result;
           currentReturn = 0;
           System.out.println("Found a better one " +
           bestErgebnis.getGewinn());
       }
       profit = result.getGewinn();
       if(profit > 0)currentReturn++;
   }
   System.out.println("Profit: " + bestErgebnis.getGewinn());
   return bestErgebnis.getGewinnZahlen();
}
```

GewinnRechner.java:getGewinnZahlen

```
public List<Integer> getGewinnZahlen(List<Integer> inputs, int
abweichung){
   List<Integer> result = new ArrayList<>();
   int size = inputs.size() -1;
   for(int i = 0; i < 10; i++){
       Random rndm = new Random();
       int diff = rndm.nextInt(inputs.get(size)/abweichung);
       int negate = rndm.nextInt(2);
       if(negate == 1) diff = 0-diff;
       int val = (int)Math.ceil((double)inputs.get(size)/10)*i +
      (int)Math.ceil((double)inputs.get(size)/20 + diff);
       if(val < 0) val = 0;
       //if(val > inputs.get(size)) val = inputs.size() -1;
       result.add(val);
   }
   return result;
}
```

Ergebnis.java:

```
public class Ergebnis {

   List<Integer> gewinnZahlen;
   int gewinn;

   public Ergebnis(){
       gewinnZahlen = new ArrayList<>();
       gewinn = 0;
   }

   public Ergebnis(List<Integer> gewinnZahlen, int gewinn){
       this.gewinnZahlen = gewinnZahlen;
       this.gewinn = gewinn;
   }

   public int getGewinn(){return gewinn;}

   public void setGewinn(int gewinn){this.gewinn = gewinn;}

   public List<Integer> getGewinnZahlen(){return gewinnZahlen;}

   public void setGewinnZahlen(List<Integer> gewinnZahlen){
     this.gewinnZahlen = gewinnZahlen;
   }
}
```