Aufgabe 5: Stadtführung

Team-ID: 68848

Team: K7Bots

Bearbeiterin: Tabea Degenhardt

12. November 2023

Inhaltsverzeichnis

Lösungsidee

Man liest die Datei ein und speichert sie in einem Array. Danach wird die Länge der Datei, welche in der ersten Zeile ersichtlich ist, in einer Variable gespeichert und der erste Tourpunkt zu einem neuen Array hinzugefügt. Nun betrachtet man den nächsten Tourpunkt. Der Weg wird extra gespeichert.

Es läuft eine for-Schleife, bis alle Tourpunkte beachtet wurden. Starte bei dem zweiten Tourpunkt.

1. Überprüfung, ob Tourpunkt während Tour nochmal besucht wird. Ist dies:
   a. der Fall, speichere den Index des nächsten Besuchs des Tourpunktes.
      Prüfe, ob zwischen den beiden Besuchen einen essenziellen Tourpunkt gibt.
      i. Es gibt keinen essenziellen Punkt zwischen den beiden Besuchen:
         Prüfe, welche der Besuche des Tourpunktes essenziell sind.
         1. Erster Besuch ist essenziell → füge Punkt hinzu und beginne bei 1. mit Index des zweiten Besuchs + 1
         2. Zweiter Besuch oder keiner ist essenziell → füge zweiten Punkt hinzu und beginne bei 1. mit Index des zweiten Besuchs + 1

         Füge den Weg vom letztbesuchten Punkt bis zum ersten Besuch hinzu.
      ii. Es gibt essenzielle Punkte zwischen den beiden Besuchen:
         Prüfe, ob es bis zum essenziellen Punkt Tourpunkte gibt, die mehrmals besucht werden. Ist es:
         1. wahr, überspringe die Punkte zwischen den beiden und füge nur den hinteren hinzu, füge auch den Weg hinzu.
         2. falsch, füge jede Station und ihren Weg bis zur essenziellen Station hinzu.

         Ist der ursprüngliche doppelte Tourpunkt nicht essenziell, beginne bei 1., um weitere Doppelungen zu finden.
   b. nicht der Fall, kann der Tourpunkt zum zweiten Array und sein Weg zur Variablen hinzugefügt werden. Der nächste Punkt wird betrachtet. Beginne bei 1.

Umsetzung

Der Code wurde in Processing erstellt, einer Programmiersprache, die auf Java basiert.

Die Daten zur Tour werden mit dem Befehl *‚loadStrings()‘* aus der Datei, welche als Parameter mitgegeben wird, eingelesen und in einen Array *‚bedingung‘* des Typs String geschrieben. Der Inhalt, welcher sich am Index 0 befindet, wird mithilfe des Befehls *‚Integer.parseInt()‘* von einem String zu einem Integer umgeformt und in einer Variable *‚l‘* gespeichert. Er gibt Auskunft über die Anzahl der Stationen.

Danach wird ein zweiter Array des Typ String erstellt, diesmal mit der Bezeichnung *‚route‘*. Zu ihm wird das zweite Element aus *‚bedingung‘* hinzugefügt – die erste Station der Tour.

```
//Initialisierung
String[] bedingung;
String[] route;
String[] stationL;
String[] wegL;
int indexRoute = 1;
int laenge = 0;

void setup() {
  //Einlesen der Datei
  bedingung = loadStrings("tour5.txt");
  //Länge der ursprünglichen Route
  int l = Integer.parseInt(bedingung[0]);
  //Initialisiere das Array route mit der richtigen Größe
  route = new String[l];
  //Setze das erste Element von route
  route[0] = addInfo(1);
```

Durch die Funktion ‚*addInfo()‘* wird der Name und das Jahr der Station übermittelt.
Der Funktion wird der Index der Station in ‚*bedingung‘* übergeben. Die Funktion teilt mit ‚*split()‘* den Inhalt am übermittelten Index des Arrays ‚*bedingung‘* am Trennzeichen `,` in mehrere Strings, welche zum Array ‚*stationL‘* hinzugefügt werden.
Danach wird dem String ‚*station‘* erst der Name zugewiesen (Index 0 des Arrays), dann ein Komma und schließlich die Jahresanzahl angehangen. Zum Schluss wird er zurückgegeben.

```
//Gibt Name der Station und das entsprechende Jahr zurück
public String addInfo(int index) {
  stationL = split(bedingung[index], ',');
  String station = stationL[0];
  station += ", ";
  station += (stationL[1]);
  return station;
}
```

Im nächsten Schritt wird eine for-Schleife erstellt, um alle Punkte der Tour zu überprüfen. Dafür wird 2 als Initialisierung des Zählvariable und *‚l‘* als Abbruchbedingung genutzt, weil die zweite Station der Liste beim Index 2 zu finden ist und *‚l‘* der Index der letzten Station ist.

Als erstes wird geprüft, ob die aktuelle Station im weiteren Verlauf der Tour noch einmal angelaufen wird. Für diese Prüfung werden zwei Indexe an eine Funktion ‚*doppelt()‘* weitergegeben und der return-Wert in einer Variablen gespeichert.

```
//alle Stationen abfragen
for (int i = 2; i < l+1; i++) {
  //Station wird mehrmals besucht (=doppelt)
  int duplicateIndex = doppelt(i, l);
  if (duplicateIndex != 0) {
```

Die Funktion *‚doppelt()‘* geht systematisch die Stationen durch und prüft, ob eine Station nochmal besucht wird. Dabei nutzt sie die Funktion *‚nameStation()‘*, welcher ein Index übermittelt wird. Wie schon bei *‚addStation()‘* wird der Inhalt am Index in Strings getrennt und in einem Array gespeichert. Im nächsten Schritt wird der String am Index 0 zum String *‚station‘* hinzugefügt und zurückgegeben, da die erste Information aus dem Array der Name der Station ist.

```
//Gibt Namen der Station zurück
public String nameStation(int index) {
  stationL = split(bedingung[index], ',');
  String station = stationL[0];
  return station;
}
```

Die Funktion *‚string1.equals(string2)‘* gibt einen boolean zurück. Haben beide Strings den gleichen Inhalt, so gibt sie *‚true‘* zurück, ansonsten *‚false‘*.
Wird eine Station innerhalb des angegebenen Bereichs noch einmal besucht, gibt die Funktion *‚doppelt()‘* den Index des nächsten Besuches zurück, andernfalls *‚0‘*.

```
//Prüft, ob die Station mehrmals besucht wird und gibt dann den Index des nächten Males zurück
public int doppelt(int start, int ende) {
  String geg = nameStation(start);
  //systematisches Abfragen aller weiteren Stationen
  for (int i = start + 1; i < ende; i++) {
    String ges = nameStation(i);
    if (geg.equals(ges)) {
      return i;
    }
  }
  return 0;
}
```

Fall a) Der Tourpunkt wird im weiteren Verlauf noch einmal besucht

Der *return*-Wert ist ungleich *‚0‘* und damit der Index des nächsten Besuches. Nun wird geprüft, ob sich zwischen den beiden Besuchen des gleichen Tourpunktes essenzielle Tourpunkte befinden. Dafür werden zwei Werte der Funktion *‚wichtigIndex()‘* übermittelt.

```
//kein wichtiger Stopp dazwischen
if (wichtigIndex(i, duplicateIndex)==0) {
```

Die Funktion prüft mit einer for-Schleife systematisch mithilfe der Funktion *‚wichtig()‘*, ob eine Station essenziell ist. Da die beiden übermittelten Werte Grenzwerte sind und nicht mit abgefragt werden sollen, startet die Schleife bei *‚start+1‘*.
Der Funktion *‚wichtig()‘* wird ein Index übermittelt und dann prüft sie, ob an diesem Index im Array *‚bedingung‘* ein *‚X‘* enthalten ist, die Station also essenziell ist. Ist die Station essenziell, wird *‚true‘* zurückgegeben, ansonsten *‚false‘*.

```
//Gibt zurück, ob die Station essenziell ist
boolean wichtig(int index) {
  if (bedingung[index].contains("X")) {
    return true;
  } else {
    return false;
  }
}
```

Sobald ein essenzieller Tourpunkt gefunden wird, übermittelt die Funktion den Index dieses Punktes. Findet sie jedoch keinen essenziellen Punkt, gibt sie 0 zurück.

```
//Gibt Index einer essenziellen Station (in einem Bereich) wieder
public Integer wichtigIndex(int start, int ende) {
  for (int i=start+1; i<ende; i++) {
    if (wichtig(i)) {
      return i;
    }
  }
  return 0;
}
```

Fall i) Zwischen den beiden Besuchen befindet sich kein essenzieller Punkt

Da sich kein essenzieller Tourpunkt zwischen den beiden Besuchen befindet, wird nun überprüft, welcher der Besuche essenziell ist und hinzugefügt werden muss. Es gibt mehrere Fälle, die sich in zwei Oberfälle einteilen lassen. Die Abfrage, ob eine Station essenziell ist, funktioniert über die Funktion *‚wichtig()‘*, die bereits bei *‚wichtigAnz()‘* genutzt wurde.

Fall 1) Der erste Besuch ist essenziell

```
//erster Besuch wichtig
if (wichtig(i)) {
  route[indexRoute] = addInfo(i);
}
```

Seine Daten mithilfe von *‚addInfo()‘* zum Array *‚route‘* hinzugefügt. Die Variable *‚indexRoute‘* gibt den Index im Array an. Nach jedem Hinzufügen einer Station wird er um 1 erhöht.

Fall 2) Der erste Besuch ist nicht essenziell

Ist der zweite Besuch oder keiner der beiden essenziell, wird der zweite Besuch zum Array *‚route‘* hinzugefügt.

```
//zweiter Besuch oder keiner wichtig
if (wichtig(duplicateIndex)||!wichtig(i)) {
  route[indexRoute] = addInfo(duplicateIndex);
  indexRoute += 1;
}
```

Egal welcher der beiden Fälle eintritt, der Weg wird bei beiden von der vorherigen Station zum ersten Besuch mithilfe der Funktion *‚weg()‘* berechnet. Ihr werden zwei Werte übermittelt. In diesem Fall sind das die Indexe des Vorgängers des ersten Besuches und des ersten Besuches. Der Weg wird in der Variablen *‚laenge‘* gespeichert.

```
  //Weg bis zum ersten Besuch
  laenge = laenge + weg(i-1, i);
  i=duplicateIndex;
}
```

Dieser Schritt kann so vorgenommen werden, da es um den Abstand zwischen dem ersten Besuch und der Station davor geht. Nutzt man die Entfernung vom zweiten Besuch, würde man die nicht gelaufene Schleife über die Stationen zwischen den beiden Besuchen mit einberechnen.

Die Funktion *‚weg()‘* teilt wie *‚wichtig()‘* den Inhalt an einem Index am Trennzeichen Komma in Teilstrings, welche zu einem neuen Array hinzugefügt werden. Als nächstes wird der Inhalt des Arrays *‚bedingung‘* am Index 3, also die Entfernung, mithilfe der Funktion *‚trim()‘* von Leerzeichen am Anfang oder Ende des Strings befreit und mit *‚Integer.parseInt()‘* zu einem

Integer umgeformt.
Dieser Vorgang wird für zwei Indexe gemacht und anschließend wird die Differenz der beiden Werte übermittelt.

```
//Berechnet den Weg zwischen zwei Stationen
public Integer weg(int index1, int index2) {
  wegL = split(bedingung[index1], ',');
  //Aus dem Array die Länge auslesen und Leerzeichen entfernen
  int weg = Integer.parseInt(wegL[3].trim());
  wegL = split(bedingung[index2], ',');
  weg = Integer.parseInt(wegL[3].trim())-weg;
  return weg;
}
```

Fall ii) Zwischen den beiden Besuchen befinden sich essenziellere Punkte

```
//wichtiger Stopp dazwischen
else if (wichtigIndex(i, duplicateIndex)!=0) {
```

Befinden sich zwischen den beiden Besuchen essenzielle Punkte, so können diese nicht einfach übersprungen werden. Stattdessen wird mithilfe einer for-Schleife geprüft, ob sich zwischen dem ersten Besuch und dem ersten essenziellen Punkt Stationen befinden, die innerhalb dieses Bereiches mehrmals besucht werden.

```
//doppelte Station zwischen erstem und zweitem Besuch
for (int j=i; j<wichtigIndex(i, duplicateIndex); j++) {
```

Fall 1) Die Station wird mehrmals besucht

Wird die Station mehrmals besucht, so können alle Stationen zwischen den beiden Besuchen und der zweite Besuch übersprungen werden, da sie definitiv nicht essenziell sind. Es wird nur der erste Besuch zum Array hinzugefügt und auch nur der Weg dahin beachtet. *‚i‘* aus der äußersten Zählschleife wird auf den Index des zweiten Besuchs gesetzt, damit nur noch die Stationen danach abgefragt werden.

```
//Station an Index j wird zweimal besucht
if (doppelt(j, wichtigIndex(i, duplicateIndex))!=0) {
  route[indexRoute] = addInfo(j);
  indexRoute += 1;
  laenge = laenge + weg(j-1, j);
  //i hat Index des zweiten Besuchs
  i=doppelt(j, wichtigIndex(i, duplicateIndex));
}
```

Fall 2) Die Station wird einmal besucht

Wird die Station in diesem Bereich nur einmal besucht, so muss sie zum Array hinzugefügt und ihr Weg beachtet werden. *‚i‘* aus der äußersten Zählschleife wird auf *‚j‘* aus dieser Zählschleife gesetzt, damit keine Station mehrmals überprüft wird.

```
//j wird nicht zweimal besucht, kann also nicht übersprungen werden
else {
  route[indexRoute] = addInfo(j);
  indexRoute += 1;
  laenge = laenge + weg(j-1, j);
  i=j;
}
```

Fall b) Der Tourpunkt wird nur einmal besucht

Wird der Punkt nur einmal besucht, so wird er zum Array *‚route‘* hinzugefügt und der Weg zwischen seinem Vorgänger und ihm berechnet, um zu *‚laenge‘* hinzugefügt zu werden.

```
//Station ist nicht doppelt, muss also besucht werden
else {
  route[indexRoute] = addInfo(i);
  indexRoute += 1;
  laenge = laenge + weg(i-1, i);
}
```

Wenn das Ende des Arrays *‚bedingung‘* erreicht wurde, kommt es zur Ausgabe der Route, welche sich in *‚route‘* befindet.
Eine while-Schleife läuft, damit nur Inhalt und nicht *‚null‘* ausgegeben wird (der Array *‚route‘* wurde nicht komplett gefüllt). Solange der Inhalt am nächsten Index nicht *null* ist, wird der Inhalt an diesem Index zusammen mit „ -> „ ausgegeben.
Durch den Befehl *‚print‘* wird alles in einer Zeile ausgegeben. Um auch den letzten Tourpunkt auszugeben, der durch das *‚index+1‘* aus der Schleife fällt (da er keinen Pfeil benötigt), wird er nach der while-Schleife ausgegeben.

Danach wird noch die Länge der Route ausgegeben.

```
print("Die Route besteht aus den folgenden Stationen: ");
int index=0;
while (route[index+1]!=null) {
  print(route[index] + " -> ");
  index += 1;
}
print(route[index]);
println(" und ist " + laenge + " Einheiten lang.");
```

Es wird der kürzeste Weg ausgegeben, da geprüft wird, ob sich zwischen einem doppelt besuchten Tourpunkt essenzielle Stationen befinden. Ist dies nicht der Fall, können die Stationen dazwischen ausgelassen werden. Befindet sich ein essenzieller Punkt dazwischen, wird wieder geschaut, ob es mehrfach besuchte Punkte dazwischen gibt, welche man als „Abkürzung“ nutzen könnte, indem alle Punkte dazwischen nicht besucht werden, da sie auf jeden Fall nicht essenziell sind. Das wird bis zum essenziellen Punkt geprüft.
Damit wird jede Möglichkeit zur Kürzung der Route beachtet.

Beispiele

Beispiel 1 (tour1.txt)

Die Route besteht aus den folgenden Stationen: Brauerei, 1613 -> Karzer, 1665 -> Rathaus, 1678 -> Rathaus, 1739 -> Euler-Brücke, 1768 -> Fibonacci-Gaststätte, 1820 -> Schiefes Haus, 1823 -> Theater, 1880 -> Emmy-Noether-Campus, 1912 -> Emmy-Noether-Campus, 1998 -> Euler-Brücke, 1999 -> Brauerei, 2012 und ist 1020 Einheiten lang.

Beispiel 2 (tour2.txt)

Die Route besteht aus den folgenden Stationen: Brauerei, 1613 -> Karzer, 1665 -> Rathaus, 1739 -> Euler-Brücke, 1768 -> Fibonacci-Gaststätte, 1820 -> Schiefes Haus, 1823 -> Theater, 1880 -> Emmy-Noether-Campus, 1912 -> Emmy-Noether-Campus, 1998 -> Euler-Brücke, 1999 -> Brauerei, 2012 und ist 1020 Einheiten lang.

Beispiel 3 (tour3.txt)

Die Route besteht aus den folgenden Stationen: Talstation, 1768 -> Wäldle, 1841 -> Bergstation, 1866 -> Observatorium, 1874 -> Piz Spitz, 1898 -> Panoramasteg, 1912 -> Ziegenbrücke, 1979 -> Talstation, 2005 und ist 2780 Einheiten lang.

Beispiel 4 (tour4.txt)

Die Route besteht aus den folgenden Stationen: Blaues Pferd, 1523 -> Alte Mühle, 1544 -> Marktplatz, 1562 -> Springbrunnen, 1571 -> Dom, 1596 -> Bogenschütze, 1683 -> Schnecke, 1698 -> Fischweiher, 1710 -> Reiterhof, 1728 -> Schnecke, 1829 -> Europapark, 1852 -> Große Gabel, 1874 -> Fingerhut, 1917 -> Stadion, 1934 -> Marktplatz, 1962 -> Baumschule, 1974 -> Polizeipräsidium, 1991 -> Blaues Pferd, 2004 und ist 2220 Einheiten lang.

Beispiel 5 (tour5.txt)

Die Route besteht aus den folgenden Stationen: Gabelhaus, 1638 -> Burgruine, 1654 -> Labyrinth, 1667 -> Hängepartie, 1672 -> Hexentanzplatz, 1681 -> Gabelhaus, 1699 -> Hexentanzplatz, 1703 -> Eselsbrücke, 1711 -> Dreibannstein, 1724 -> Alte Wache, 1733 -> Palisadenhaus, 1740 -> Dreibannstein, 1752 -> Schmetterling, 1760 -> Dreibannstein, 1781 -> Märchenwald, 1793 -> Eselsbrücke, 1877 -> Reiterdenkmal, 1880 -> Riesenrad, 1902 -> Dreibannstein, 1911 -> Olympisches Dorf, 1924 -> Haus der Zukunft, 1927 -> Stellwerk, 1942 -> Labyrinth, 1955 -> Gauklerstadl, 1961 -> Planetarium, 1971 -> Känguruhfarm, 1976 -> Balzplatz, 1978 -> Dreibannstein, 1998 -> Labyrinth, 2013 -> CO2-Speicher, 2022 -> Gabelhaus, 2023 und ist 3850 Einheiten lang.

Quellcode

```
//Initialisierung
String[] bedingung;
String[] route;
String[] stationL;
String[] wegL;
int indexRoute = 1;
int laenge = 0;

void setup() {
  //Einlesen der Datei
  bedingung = loadStrings("tour1.txt");
  //Länge der ursprünglichen Route
  int l = Integer.parseInt(bedingung[0]);
  //Initialisiere das Array route mit der richtigen Größe
  route = new String[l];
  //Setze das erste Element von route
  route[0] = addInfo(1);
  //alle Stationen abfragen
  for (int i = 2; i < l+1; i++) {
    //Station wird mehrmals besucht (=doppelt)
    int duplicateIndex = doppelt(i, l);
    if (duplicateIndex != 0) {
      //kein wichtiger Stopp dazwischen
      if (wichtigIndex(i, duplicateIndex)==0) {
        //erster Besuch wichtig
        if (wichtig(i)) {
          route[indexRoute] = addInfo(i);
          indexRoute += 1;
        }
        //zweiter Besuch oder keiner wichtig
        if (wichtig(duplicateIndex)||!wichtig(i)) {
          route[indexRoute] = addInfo(duplicateIndex);
          indexRoute += 1;
        }
        //Weg bis zum ersten Besuch
        laenge = laenge + weg(i-1, i);
        i=duplicateIndex;
      }
```

```
      //wichtiger Stopp dazwischen
      else if (wichtigIndex(i, duplicateIndex)!=0) {
        //doppelte Station zwischen erstem und zweitem Besuch
        for (int j=i; j<wichtigIndex(i, duplicateIndex); j++) {
          //Station an Index j wird zweimal besucht
          if (doppelt(j, wichtigIndex(i, duplicateIndex))!=0) {
            route[indexRoute] = addInfo(j);
            indexRoute += 1;
            laenge = laenge + weg(j-1, j);
            //i hat Index des zweiten Besuchs
            i=doppelt(j, wichtigIndex(i, duplicateIndex));
          }
          //j wird nicht zweimal besucht, kann also nicht übersprungen werden
          else {
            route[indexRoute] = addInfo(j);
            indexRoute += 1;
            laenge = laenge + weg(j-1, j);
            i=j;
          }
        }
      }
    }
    //Station ist nicht doppelt, muss also besucht werden
    else {
      route[indexRoute] = addInfo(i);
      indexRoute += 1;
      laenge = laenge + weg(i-1, i);
    }
  }
  print("Die Route besteht aus den folgenden Stationen: ");
  int index=0;
  while (route[index+1]!=null) {
    print(route[index] + " -> ");
    index += 1;
  }
  print(route[index]);
  println(" und ist " + laenge + " Einheiten lang.");
}

//Funktionen
/*Prüft, ob die Station mehrmals besucht wird und gibt
dann den Index des nächten Besuchs zurück*/
public int doppelt(int start, int ende) {
  String geg = nameStation(start);
  //systematisches Abfragen aller weiteren Stationen
  for (int i = start + 1; i < ende; i++) {
    String ges = nameStation(i);
    if (geg.equals(ges)) {
      return i;
    }
  }
  return 0;
}
```

```
92  //Gibt Namen der Station zurück
93  public String nameStation(int index) {
94    stationL = split(bedingung[index], ',');
95    String station = stationL[0];
96    return station;
97  }
98  //Gibt zurück, ob die Station essenziell ist
99  boolean wichtig(int index) {
100   if (bedingung[index].contains("X")) {
101     return true;
102   } else {
103     return false;
104   }
105 }
106 //Gibt Index einer essenziellen Station (in einem Bereich) wieder
107 public Integer wichtigIndex(int start, int ende) {
108   for (int i=start+1; i<ende; i++) {
109     if (wichtig(i)) {
110       return i;
111     }
112   }
113   return 0;
114 }
115 //Gibt Name der Station und das entsprechende Jahr zurück
116 public String addInfo(int index) {
117   stationL = split(bedingung[index], ',');
118   String station = stationL[0];
119   station += ", ";
120   station += (stationL[1]);
121   return station;
122 }
123 //Berechnet den Weg zwischen zwei Stationen
124 public Integer weg(int index1, int index2) {
125   wegL = split(bedingung[index1], ',');
126   //Aus dem Array die Länge auslesen und Leerzeichen entfernen
127   int weg = Integer.parseInt(wegL[3].trim());
128   wegL = split(bedingung[index2], ',');
129   weg = Integer.parseInt(wegL[3].trim())-weg;
130   return weg;
131 }
```