Buhnenrennen Dokumentation

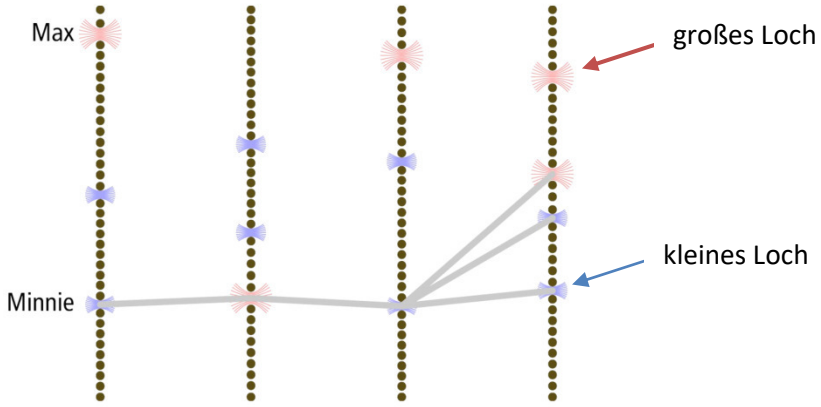
Bundeswettbewerb Informatik 2016

Johannes Sonn
2016

Inhaltsverzeichnis

Lösungsidee

Es soll ein Programm geschrieben werden, welches bestimmen soll, ob ein kleiner Hund (Minnie) vor einem großen Hund (Max) durch verschieden große Löcher in Buhnen abhauen kann. Falls es einen sicheren Weg für Minnie gibt, soll dieser ausgegeben werden. Die Buhnen haben einen Abstand zueinander von 70 Metern. Außerdem rennt Max mit 30 km/h und Minnie mit 20 km/h.



Quelle: https://git.bwinf.de/bwinf/bwinf35-runde1/raw/master/aufgabe5-Buhnenrennen/aufgabe5-Buhnenrennen.pdf

Um feststellen zu können, ob Minnie sicher von einem Loch zum nächsten Loch in der nächsten Buhne rennen kann, müssen die Zeiten, die Minnie und Max benötigen um zu diesem Loch zu gelangen verglichen werden.

Wenn die Zeit von Max *kleiner* ist als die Zeit die Minnie benötigt, wird Minnie von Max gefangen.

Wenn die Zeit von Max *größer* ist als die Zeit die Minnie benötigt, kann Minnie zu diesem Loch gelangen ohne gefangen zu werden. In diesem Fall ist dieses Loch sicher.

Von allen sicheren Löchern wird das Loch, welches am nächsten zu Minnies aktueller Position ist, als Minnies nächste Position gewählt. Max hingegen hat keine festen Positionen (außer seine Startposition). Seine Zeiten werden von dem Loch, welches geprüft wird, ob es sicher für Minnie ist, zu seiner Startposition zurück gerechnet. Für die Berechnung von den Zeiten für Max wird ebenso immer der kürzeste Weg, den Max nehmen kann genommen. Dadurch, dass immer der kürzeste Weg für Max gewählt wird, wird geprüft, ob Max, selbst wenn er direkt zu diesem Loch rennen würde, Minnie fangen könnte.

Umsetzung

Als Hochsprache wird Python verwendet. Das Programm wird durch die Kommandozeile, mit dem Pfad und Namen der Input-Datei, aufgerufen.

Die in den Argumenten angegebene Input-Datei wird als Liste (*input*) eingelesen. Dabei wird jede Zeile des Textdokuments, bzw. jedes Koordinatenpaar in einen separaten Eintrag in der Liste geschrieben. Anschließend werden die y-Koordinaten der Startlöcher von Max und Minnie in zwei Variablen geschrieben. Danach werden die Zeiten von der Startposition von Max und Minnie zu jedem Loch in der nächsten Buhne einzeln berechnet und verglichen. Für die Berechnung der Zeiten (in Sekunden) wird zuerst, mithilfe des Satzes des Pythagoras, die Strecke in Metern berechnet und durch die verschiedenen Geschwindigkeiten der beiden Hunde (in m/s) geteilt. Wenn Minnies Zeit kürzer ist als die von Max, wird die Zeit von Minnie und die y-Koordinate des Lochs, mit demselben Index, in die Listen *loesungList* und *tLoesungList* eingefügt. Falls aber Minnies Zeit länger als die von Max ist, wird Minnie gefangen und die Variable *minnieGefangen* wird auf *True* gesetzt.

Nachdem die Zeiten von den Startpositionen zu jedem Loch in der nächsten Buhne berechnet sind, wird aus der Liste *tLoesungList* der Index der minimalsten Zeit herausgesucht. Diese Zeit wird zur Variablen *tMinnieGes* hinzuaddiert und der Eintrag aus der Liste *loesungList* mit demselben Index, in die Liste *loesungen* eingefügt. Nach der ersten Buhne wird dies für alle weiteren Buhnen wiederholt. Dabei wird die Strecke für Minnie von dem vorherigen Loch bis zum aktuellen Loch berechnet. Für Max wird wie oben beschrieben die Strecke rückwärts berechnet. Zur Zeit von Minnie wird zusätzlich der Wert der Variable *tMinnieGes* (Summe der Zeiten der vorigen Löcher) addiert.

Anschließend, wenn *minnieGefangen* gleich *False* ist, werden die Koordinaten, der Löcher, die zu dem sicheren Weg gehören ausgegeben. Diese Koordinaten werden im selben Pfad wie die Input-Datei als Textdokument mit demselben Namen wie die Input-Datei mit der Erweiterung „_Loesung.txt“ gespeichert. Ansonsten wird „Minnie wird gefangen!“ ausgegeben.

Beispiele

```
$ python BuhnenrennenV3.32.py /home/sonns/buhnenrennen1.txt

Loesung:

0, 42.95
70, 45.9
140, 42.2
210, 49.4
$ python BuhnenrennenV3.32.py /home/sonns/buhnenrennen2.txt

Loesung:

0, 42.1
70, 14.1
140, 28.0
210, 16.6
280, 17.45
350, 56.55
420, 50.3
490, 53.85
$ python BuhnenrennenV3.32.py /home/sonns/buhnenrennen3.txt
Minnie wird gefangen!
$ python BuhnenrennenV3.32.py /home/sonns/buhnenrennen4.txt

Loesung:

0, 41.8
70, 15.65
140, 16.55
210, 25.55
280, 34.4
350, 40.05
420, 27.6
490, 13.25
```

(Output-Dateien der angegebenen und weiteren Beispielen sind im Ordner *Loesungsdateien* zu finden)

Lösungstabelle

Input-Datei	Möglich / Unmöglich
buhnenrennen1.txt	Möglich
buhnenrennen2.txt	Möglich
buhnenrennen3.txt	Unmöglich
buhnenrennen4.txt	Möglich
buhnenrennen5.txt	Möglich
buhnenrennen6.txt	Möglich
buhnenrennen7.txt	Unmöglich
buhnenrennen8.txt	Unmöglich
buhnenrennen9.txt	Möglich
buhnenrennen10.txt	Unmöglich
buhnenrennen11.txt	Möglich
buhnenrennen12.txt	Möglich
buhnenrennen13.txt	Unmöglich
buhnenrennen14.txt	Möglich
Gesamt:	9x Möglich & 5x Unmöglich

Möglich ≙ Minnie wird nicht gefangen

Unmöglich ≙ Minnie wird gefangen

```
# -*- coding: utf-8 -*-

########################################
### Titel:           Buhnenrennen   ####
### VersionsNr:           V3.33     ###
### Autor:          Johannes Sonn   ##
####################################



###Import
import sys

###Variablendeklaration
inputfile = ""
input = ""
outputfile = ""
outputfilename = ""
startMax = 0 #Startposition vom Max
startMinnie = 0 #Startposition von Minnie
vMax = 25.0/3.0 #m/s
vMinnie = 50.0/9.0 #m/s
i = 1 #i = Buhnenzahl
j = 1 #wird für die Berechnung von tMax benötigt
k = 1 #wird für die Berechnung von tMax benötigt
y1 = 0 #wird für die Berechnung von tMinnie benötigt
y2 = 0 #wird für die Berechnung von tMinnie benötigt
tMax = 0 #Zeit die Max benötigt um von der Startposition zu einem bestimmten Loch zu gelangen
tMinnie = 0 ##Zeit die Max benötigt um von der Startposition zu einem bestimmten Loch zu gelangen
tLoesungMinnie = 0
tLoesungMax = 0
tMaxGes = 0
tMinnieGes = 0
maxPos = 0
loesungen = []
loesungList = []
tLoesungList = []
xLoecher = []
tXLoecher = []
minnieGefangen = False
USAGE = "Buhnenrennen [input-Datei]"

#Wenn das Programm nicht mit 2 Argumenten (der Programmname wird mitgezählt) aufgerufen wurde,...
if len(sys.argv) != 2:
    #...wird ausgegeben wie man das Programm richtig aufruft
    print USAGE

#Wenn es mit 2 Argumente aufgerufen wurde, wird die angegebene Datei als "file" geöffnet
#und als Liste in "input" eingelesen
else:
    file = open(sys.argv[1])
    input = file.readlines()

    #Es werden die Zeilenumbrüche der Input-Datei in "input" gelöscht
    for koor in input:
        input[input.index(koor)] = koor[0:-1]


    #Y-Koordinaten der Startloecher werden in "startMax" und "startMinnie" eingelesen
    startMax = float(input[0][4:])
    startMinnie = float(input[1][4:])
```

```
66
67      #Für alle Einträge (Koordinaten der Löcher) bzw. "koor" in "input", wird
        folgendes ausgeführt:
68      for koor in input:
69
70
71          #Wenn Minnie noch nicht gefangen wurde
72          if minnieGefangen == False:
73
74              #Wenn '70' mal "i" in "koor" vorkommt (bzw. '70' mal "i" nicht nicht in
                "koor" gefunden wurde)
75              if koor.find(" " + str(70 * i) + " ") != -1:
76
77                  #Der y-Wert aus "koor" wird in "y2" geschrieben
78                  y2 = float(koor[3+len(str(70 * i)):])
79
80                  #Wenn in "koor" 70 vorkommt, also bei der ersten Buhne
81                  if koor.find(" 70 ") == 1:
82
83                      #Die Zeiten, die Minnie und Max von dem letzten Loch zum Loch
                        mit den Koordinaten "koor" werden berechnet
84                      #Die Strecke wird durch den Satz des Pythagoras berechnet und
                        durch die Geschwindigkeit der beiden Hunde (in m/s) berechnet
85                      tMinnie = ((70**2 + (startMinnie - y2)**2)**0.5)/vMinnie
86                      tMax = ((70**2 + (startMax - y2)**2)**0.5)/vMax
87
88                      #Wenn Minnie schneller als Max zum nächsten Loch kommt (also
                        tMinnie kleiner al tMax ist),
89                      #werden die Zeit und die y-Koordinaten des Lochs zu
                        "loesungList" und in "tLoesungList" hinzugefügt
90                      if tMinnie < tMax:
91                          loesungList.append(y2)
92                          tLoesungList.append(((70**2 + (startMinnie - y2)**2)**0.5)/
                            vMinnie)
93
94
95
96
97                  #Bei jedem weiteren Loch, außer bei jedem ersten Loch in einer neuen
                    Buhne und bei der ersten Buhne
98                  elif koor.find(" 0 ") == -1:
99
100                     #Der letzte Eintrag aus "loesungen" (also Minnies letzte
                        Position) wird in y1 eingefügt
101                     y1 = loesungen[-1]
102
103                     #Die y-Koordinate aus "koor" wird in y2 eingefügt
104                     y2 = float(koor[3+len(str(70 * i)):])
105
106                     #Die Summe aus der Zeit, die Minnie von der aktuellen Position
                        zu "koor" benötigt und "tMinnieGes" (die Summe der vorigen
                        berechneten Zeiten) wird in "tMinnie" gespeichert
107                     tMinnie = ((70**2 + (y1 - y2)**2)**0.5)/vMinnie + tMinnieGes
108
109
110                     ###MAX Zeit rückwärts berechnen
111
112                     #In "j" wird der Index von "koor" in "input" um 1 verringert
                        gespeichert
113                     j = input.index(koor) - 1
114
115                     #In "k" wird "i" um 1 verringert gespeichert
116                     k = i - 1
117
118                     #tMax wird auf 0 gesetzt
119                     tMax = 0
120
121                     #Der Wert aus "y2" wird in "maxPos" gespeichert
122                     maxPos = y2
```

```
123
124                     #Solange "k" größer als 0 ist, wird folgendes ausgeführt
125                     while k > 0:
126
127                         #Wenn der Eintrag aus "input" mit dem Index "j" mit ("x " +
                            str( k * 70 ) + " ") beginnt (also der Eintrag ein großes
                            Loch mit der x-Position (k*70) beschreibt),
128                         #soll folgendes ausgeführt werden:
129                         if (input[j]).startswith("x " + str( k * 70 ) + " "):
130
131                             #Zur Liste "xLoecher" soll aus diesem Eintrag aus
                                "input" mit Index "j" der y-Wert des Max-Lochs hinzugefügt
132                             xLoecher.append( float( input[ j ] [ ( (input[j]).index(
                                " ", 2) +1 ) :]))
133
134                             #Außerdem soll die benötigte Zeit, die Max benötigt um
                                von der letzten Position (maxPos)
135                             #zu diesem Loch zu gelangen zu "txLoecher" hinzugefügt
                                werden
136                             tXLoecher.append(((70**2 + ( (xLoecher[-1]) - maxPos)**2
                                )**0.5)/vMax)
137
138                         #Wenn in dem Eintrag aus "input" mit dem Index "j"  ("x " +
                            str( k * 70 ) + " ") NICHT gefunden werden kann (also der
                            Eintrag ein Loch beschreibt, welches sich nicht in der
                            Buhne, mit der x-Koordinate (k+70) befindet)
139                         # UND in dem (" "+str( i * 70 )+" ") NICHT gefunden wird
                            (also der Eintrag ein Loch beschreibt, welches NICHT in der
                            selben Buhne vorkommt wie "koor"),
140                         #soll folgendes ausgeführt werden
141                         elif (input[j]).find(" "+str( k * 70 )+" ") == -1 and (input[
                            j]).find(" "+str( i * 70 )+" ") == -1:
142
143                             #Wenn "tXLoecher" nicht leer ist, soll folgendes
                                ausgeführt werden.
144                             if tXLoecher:
145
146                                 #"tMax" soll um das Minimum aus "tXLoecher" erhöht
                                    werden
147                                 tMax += min(tXLoecher)
148
149                                 #"maxPos" soll auf die Position des Lochs gesetzt
                                    werden, welches am nächsten zur vorherigen Position
                                    ist (also den Wert aus "xLoecher" mit dem Index des
                                    Minimums aus tXLoecher)
150                                 maxPos = xLoecher[tXLoecher.index(min(tXLoecher))]
151
152                             #"xLoecher" und "tXLoecher" werden geleert
153                             xLoecher = []
154                             tXLoecher = []
155
156                             #"k" wird um 1 verringert
157                             k -= 1
158
159
160                             #Wenn "k" gleich 0 ist, dann soll folgendes ausgeführt
                                werden:
161                             if k == 0:
162
163                                 #"tMax" wird um die Zeit erhöht, die Max benötigt um
                                    von seiner Startposition (startMax) zur ersten Buhne
                                    bzw. "maxPos" zu rennen
164                                 tMax += ((70**2 + ( startMax - maxPos)**2)**0.5)/vMax
165
166                         #"j" wird um 1 verringert
167                         j -= 1
168
169
170
```

```
171                         #Wenn Minnie schneller als Max zum nächsten Loch kommt (also
                            tMinnie kleiner al tMax ist),
172                         #werden die Zeit und die y-Koordinaten des Lochs zu
                            "loesungList" und in "tLoesungList" hinzugefügt
173                         if tMinnie < tMax:
174                             loesungList.append(y2)
175                             tLoesungList.append(((70**2 + (y1 - y2)**2)**0.5)/vMinnie)
176
177
178
179             #Bei jedem ersten Loch in jeder Buhne, außer in der nullten (also " 0 "
                nicht in "koor" gfunden wird) und der ersten Buhne, soll folgendes
                ausgeführt werden:
180             elif koor.find(" 0 ") == -1:
181
182                 #Wenn die "loesungList" leer ist, also Minnie gefangen wurde und es
                    bei einer Buhne kein Loch gab zu dem Minnie schneller rennen konnte,
183                 #wird "minnieGefangen" auf 'True' gesetzt
184                 if loesungList == []:
185                     minnieGefangen = True
186
187                 #Wenn Minnie nicht gefangen wurde, soll folgendes ausgeführt werden:
188                 else:
189                     #Es wird der Liste "loesungen" die y-Koordinate des letzten
                        Lochs hinzugefügt
190                     loesungen.append(loesungList[tLoesungList.index(min(tLoesungList
                        ))])
191
192                     #"loesungList" wird geleert
193                     loesungList = []
194
195                     #Zu tMinnieGes wird das Minimum aus "tLoesungList" addiert
196                     tMinnieGes +=  min(tLoesungList)
197
198                     #"tLoesungList", "xLoecher" und "tXLoecher" werden geleert
199                     tLoesungList = []
200                     xLoecher = []
201                     tXLoecher = []
202
203                     #"i" wird um 1 erhöht
204                     i += 1
205
206                     #Der letzte Eintrag aus "loesungen" (also Minnies letzte
                        Position) wird in y1 eingefügt
207                     y1 = loesungen[-1]
208
209                     #Die y-Koordinate aus "koor" wird in y2 eingefügt
210                     y2 = float(koor[3+len(str(70 * i)):])
211
212                     #Die Summe aus der Zeit, die Minnie von der aktuellen Position
                        zu "koor" benötigt und "tMinnieGes" (die Summe der vorigen
                        berechneten Zeiten) wird in "tMinnie" gespeichert
213                     tMinnie = ((70**2 + (y1 - y2)**2)**0.5)/vMinnie + tMinnieGes
214
215
216                     #MAX Zeit rückwärts berechnen
217
218                     #In "j" wird der Index von "koor" in "input" um 1 verringert
                        gespeichert
219                     j = input.index(koor) - 1
220
221                     #In "k" wird "i" um 1 verringert gespeichert
222                     k = i - 1
223
224                     #tMax wird auf 0 gesetzt
225                     tMax = 0
226
227
228
```

```
229                     #Der Wert aus "y2" wird in "maxPos" gespeichert
230                     maxPos = y2
231
232                     #Solange "k" größer als 0 ist, wird folgendes ausgeführt
233                     while k > 0:
234
235                         #Wenn der Eintrag aus "input" mit dem Index "j" mit ("x " +
                            str( k * 70 ) + " ") beginnt (also der Eintrag ein großes
                            Loch mit der x-Position (k*70) beschreibt),
236                         #soll folgendes ausgeführt werden:
237                         if (input[j]).startswith("x "+str( k * 70 )+" "):
238
239                             #zur Liste xLoecher wird aus input aus dem "index(koor)
                                -j" der y-Wert des Max-Lochs hinzugefügt
240                             xLoecher.append( float( input[ j ] [ ( (input[j]).index(
                                " ", 2) +1 ) :]))
241                             #Außerdem soll die benötigte Zeit, die Max benötigt um
                                von der letzten Position (maxPos)
242                             #zu diesem Loch zu gelangen zu "txLoecher" hinzugefügt
                                werden
243                             tXLoecher.append(((70**2 + ( (xLoecher[-1]) - maxPos)**2
                                )**0.5)/vMax)
244
245                         #Wenn in dem Eintrag aus "input" mit dem Index "j"  ("x " +
                            str( k * 70 ) + " ") NICHT gefunden werden kann (also der
                            Eintrag ein Loch beschreibt, welches sich nicht in der
                            Buhne, mit der x-Koordinate (k+70) befindet)
246                         # UND in dem (" "+str( i * 70 )+" ") NICHT gefunden wird
                            (also der Eintrag ein Loch beschreibt, welches NICHT in der
                            selben Buhne vorkommt wie "koor"),
247                         #soll folgendes ausgeführt werden
248                         elif (input[j]).find(" "+str( k * 70 )+" ") == -1 and (input[
                            j]).find(" "+str( i * 70 )+" ") == -1:
249
250                             #Wenn "tXLoecher" nicht leer ist, soll folgendes
                                ausgeführt werden.
251                             if tXLoecher:
252
253                                 #"tMax" soll um das Minimum aus "tXLoecher" erhöht
                                    werden
254                                 tMax += min(tXLoecher)
255
256                                 #"maxPos" soll auf die Position des Lochs gesetzt
                                    werden, welches am nächsten zur vorherigen Position
                                    ist (also den Wert aus "xLoecher" mit dem Index des
                                    Minimums aus tXLoecher)
257                                 maxPos = xLoecher[tXLoecher.index(min(tXLoecher))]
258
259                             #"xLoecher" und "tXLoecher" werden geleert
260                             xLoecher = []
261                             tXLoecher = []
262
263                             #"k" wird um 1 verringert
264                             k -= 1
265
266
267                             #Wenn "k" gleich 0 ist, dann soll folgendes ausgeführt
                                werden:
268                             if k == 0:
269
270                                 #"tMax" wird um die Zeit erhöht, die Max benötigt um
                                    von seiner Startposition (startMax) zur ersten Buhne
                                    bzw. "maxPos" zu rennen
271                                 tMax += ((70**2 + ( startMax - maxPos)**2)**0.5)/vMax
272
273                         #"j" wird um 1 verringert
274                         j -= 1
275
276
```

```
277                             #Wenn Minnie schneller als Max zum nächsten Loch kommt (also
                                tMinnie kleiner al tMax ist),
278                             #werden die Zeit und die y-Koordinaten des Lochs zu
                                "loesungList" und in "tLoesungList" hinzugefügt
279                             if tMinnie < tMax:
280                                 loesungList.append(y2)
281                                 tLoesungList.append(((70**2 + (y1 - y2)**2)**0.5)/vMinnie)
282
283
284         #Wenn Minnie nicht gefangen wurde, also "minnieGefangen" gleich "False" ist,
            soll folgendes ausgeführt werden:
285         if minnieGefangen == False:
286
287             #Zur Liste "loesungen" soll das Minimum aus der "loesungList" hinzugefügt
                werden
288             loesungen.append(loesungList[tLoesungList.index(min(tLoesungList))])
289
290             #Zur List "loesungen" soll an nullter Stelle die y-Koordinate Minnies
                Startposition (startMinnie) hinzugefügt werden
291             loesungen.insert(0, startMinnie)
292
293             #Es soll "Loesung;" ausgegeben werden
294             print ("")
295             print ("Loesung:")
296             print ("")
297
298             #Es werden Ort und Name der Output-Datei festgelegt und diese Datei geöffnet
299             outputfilename = sys.argv[1][0:-4] + "_Loesung.txt"
300             outputfile = open(outputfilename, "w")
301
302
303             #Zu allen y-Koordinaten in der Liste "loesungen", soll vor der y-Koordinate
                die entsprechende x-Koordinate #(Vielfaches von 70) eingefügt werden.
304             for loesungKoor in loesungen:
305                 loesungKoor= str(70 * i ) + ", " + str(loesungKoor)
306                 i += 1
307
308                 #Danach soll dieser Tupel (die x- und y-Koordinaten der Löcher durch die
                    Minnie sicher vor Max entkommen kann) ausgegeben werden
309                 print loesungKoor
310
311                 #Es soll "loesungKoor" und ein Zeilenumbruch in die Output-Datei
                    geschrieben werden
312                 outputfile.write(loesungKoor + "\n")
313
314             #Die Output-Datei wird geschlossen
315             outputfile.close()
316
317
318
319
320         #Wenn Minnie bei einer Buhne bei keinem Loch schneller war und somit gefangen
            wurde, soll ausgegeben werden "Minnie wird gefangen!"
321         else:
322             print ("Minnie wird gefangen!")
323
324     ###Programmende###
```