Dokumentation Aufgabe 5

Jugendwettbewerb Informatik 3.Runde

Aufgabenstellung:

Es soll ein Programm geschrieben werden, den kürzesten Weg durch die Quader zu finden. Die Quader haben eine Zeitbegrenzung und jede „n“ Minuten geht der Quader runter oder hoch. Man muss die schnellstmögliche Lösung zum Grabmal finden.

Die Regel:

Die Regel ist, dass man schaut, wann der nächste Quader offen ist und wann der jetzige Quader schließt und wenn der nächste Quader offen ist, geht man durch und wenn der jetzige Quader schließt, dann geht man zurück. Aber bei der Ausgabe darf man nicht das Zurückgehen ausgeben, also entfernen wir es dann. Am Ende formulieren wir noch einen schönen Text und geben es aus.

Umsetzung:

```
import sys
import math
Quaderliste = []
```

Hiermit importiere ich sys, um den Filenamen der Eingabedatei (Text) auf der Kommandozeile zu bekommen. Ich brauche auch math, denn ich muss später Zahlen aufrunden. Dann erstelle ich „Quaderliste“. Ich brauche es später für das Einlesen der Datei.

```
def wegbeschreibung(stack):
    text = f"Warte {stack.pop(0)} Minuten, laufe "
    zähler = 1
```

Das ist die Definition “wegbeschreibung”. Sie nimmt die Liste “stack” ein und erstellt den Text der ausgegeben werden muss. Zuerst wird der erste Schritt an “text” angehängt (Die Anzahl von Minuten die man warten muss). Er wird von dem ersten Wert von “stack” genommen. (Was “stack” ist, siehe unten.) Dann erstellen wir “zähler” mit dem Wert von 1, denn wir werden später noch die Abschnitte zu “text” addieren.

```
while len(stack) > 0:
    if stack[0] == 0:
     stack.pop(0)
    else:
        text += f"zu Abschnitt {zähler}. Warte\
                 {stack.pop(0)} Minuten, laufe "
    zähler += 1
```

Wir wiederholen diese Schleife die ganze Zeit, bis “stack” leer ist. Wir schauen ob das jetzige Element in “stack” 0 ist, denn sonst kann man direkt durchlaufen und nehmen dann das jeweilige Element aus “stack” raus. Wenn es nicht 0 ist, dann schreiben wir in “text” hinein, in welchen Abschnitt man gehen muss (mit zähler) und wie lange man jetzt warten muss, bis der nächste Quader aufgeht. Dann (egal ob das Element 0 war oder nicht) addieren wir 1 zu “zähler”.

```
text += f"zum Grabmal."
return text
```

Dann, aus der Schleife raus, ist “stack” ja leer und man ist beim Grabmal, deswegen hängen wir den letzten Abschnitt an “text” heran und geben “text” dann aus.

```
def finde_offene_intervalle(jetziges_intervall,
nächste_periode):
  nächste_iv_liste = []
  n = math.ceil(jetziges_intervall[0] /
   (2*nächste_periode))
  if (2*n-1)*nächste_periode <= jetziges_intervall[0]:
  nächste_iv_liste.append((jetziges_intervall[0],
                                2*n*nächste_periode))
```

Das ist die Definition “finde_offene_intervalle”. Sie liest das jetzige Intervall und die Periode des nächsten Quaders ein und gibt aus, welche Möglichkeiten es gibt, nach vorne zu gehen. Zuerst erstellt er die Liste “nächste_iv_liste”. Sie wird später die Möglichkeiten speichern, nach vorne zu gehen. Jetzt erstelle ich die Variable “n”, die uns sagt, die wievielte Periode des nächsten Quaders in unserem Intervall endet bzw. beginnt. Dabei steht ein ungerades ”n” für ein sich öffnenden Quader. Dann addieren wir die Intervalle, die Überlapp mit dem jetzigen Intervall haben zu nächste_iv_liste. Das ist der erste Fall, dass der nächste Quader schon offen ist und vor dem jetzigen Quader schließt.

```
min_n=math.ceil(jetziges_intervall[0]/
                nächste_periode)
if min_n % 2 == 0:
   min_n += 1
max_n = int(jetziges_intervall[1]/nächste_periode)
n = min_n
```

Hier finde ich den zweiten Fall heraus. Bei diesem Fall muss “n” oder “min_n” ungerade sein. “min_n” ist die schnellst mögliche Möglichkeit in den nächsten Quader zu gehen und “max_n” die späteste Möglichkeit. Dann speichern wir “min_n” als “n”, denn wir müssen “n” noch verändern und “min_n” dürfen wir nicht.

```
while n <= max_n:
  nächste_iv_liste.append((min_n*nächste_periode,
                    (min_n + 1)*nächste_periode))
  n += 2
return nächste_iv_liste
```

Hier wiederholen wir für den zweiten Fall (dass der nächste Quader erst nach einer Zeit öffnet), zu schauen, ob es mehrere Möglichkeiten gibt, ins nächste freie Feld zu gehen. Wenn wir zu “n” jedes Mal 2 addieren (weil “n” gerade sein muss) und es immer noch kleiner ist als “max_n”, dann fügen wir das Intervall zu “nächste_iv_liste” hinzu und geben es dann aus.

```
def das_ägyptische_grabmal(file_name):
  file = open(file_name, "r")
  for line in file.readlines():
  Quaderliste.append(int(line))
  position = 0
  am_anfang = 1
  abschnitte = Quaderliste.pop(0)
  möglichkeiten = []
```

Das ist die Definition “das_ägyptische_grabmal”. Sie gibt einen Satz aus, wie man am schnellsten durch die Quader kommt. Sie nimmt eine Textdatei ein, öffnet sie und speichert sie als “file”. Dann liest sie die Datei “file” ein und speichert sie in “Quaderliste”. Jetzt erstellen wir

ein paar Variablen. “position” speichert die jetzige Position (am Anfang sind wir noch nicht in den Abschnitten, also ist es 0). “am_anfang” brauchen wir später um zu schauen wieviel Mal wir am Anfang waren. Da die erste Zeile in der Datei die Anzahl von Quadern ist, speichern wir sie in “abschnitte” und nehmen sie aus “Quaderliste” raus. Dann erstellen wir die Liste “möglichkeiten”. Sie speichert, wie viele Möglichkeiten es unter einem Quader gibt.

```
while position < abschnitte:
  if position == 0:
  stack = [Quaderliste[0]*(2 * am_anfang - 1)]
  jetzige_möglichkeiten =
  [(Quaderliste[0] * (2 * am_anfang - 1),
   (Quaderliste[0] * 2 * am_anfang))]
  position = 1
  am_anfang += 1
  else:
  jetzige_möglichkeiten = möglichkeiten.pop()
  if len(jetzige_möglichkeiten) == 0:
  stack.pop()
  position -= 1
  continue
```

Jetzt wiederholen wir die ganze Zeit, bis “position” größer als “abschnitte” ist (also bis zum Grabmal), die Schleife: Wir schauen ob unsere Position 0 ist (d.h. man ist am Anfang), denn im Laufe dieser Schleife kann man auch wieder ganz zurücklaufen zum Anfang. Wenn die Position 0 ist, dann fügen wir mit der Variable “am_anfang” zu “stack” das nächste Intervall hinzu, denn am Anfang kann man es immer herausfinden, wann der nächste Quader sich schließt. Wir brauchen einen Wert für “jetzige_möglichkeiten” (für später), also nehmen wir das Intervall, das mit am_anfang ausgerechnet wurde ein.

Jetzt gehen wir in den 1.Abschnitt, wenn der offen ist (also ist “position” jetzt 1) und addieren 1 zu “am_anfang”, weil wir jetzt schon wieder einmal am Anfang waren.

Wenn die jetzige Position nicht 0 ist, nehmen wir aus “möglichkeiten” (siehe weiter unten) das letzte raus und speichern es in “jetzige_möglichkeiten”, denn “möglichkeiten” kann sehr viele unterschiedliche Möglichkeiten enthalten und “jetzige_möglichkeiten” nur eins. Wir schauen dann ob “jetzige_möglichkeiten” eine leere Liste ist. Wenn es eine leere Liste ist, gibt es vorne keine Möglichkeiten und man muss zurück. Aus “stack” nehmen wir das letzte heraus, denn wir brauchen das nicht mehr, weil wir zurückgehen und bei “position” subtrahieren wir 1. Dann geht es mit “continue” am Anfang der Schleife wieder los.

```
jetziges_intervall = jetzige_möglichkeiten.pop(0)
intervalle =
    finde_offene_intervalle(jetziges_intervall,
                            Quaderliste[position])
if len(intervalle) == 0:
  stack.pop()
  position -= 1
  continue
```

Wir sind immer noch in der Schleife. Wir nehmen aus jetzige_möglichkeiten das erste heraus (ein Intervall) und speichern es als “jetziges_intervall”. Und dann benutzen wir unsere Definition von vorher – finde_offene_intervalle – mit den Argumenten “jetziges_intervall” und “Quaderliste” (“Quaderliste” mit dem Wert der jeweiligen Position und speichern den Output in “intervalle”. Jetzt machen wir das Gleiche wie vorher, wir schauen ob “intervalle” leer ist, wenn ja gibt es vorne keine Möglichkeiten mehr und man muss

wieder zurück, indem man das Letzte von “stack” rausnimmt und “position” um 1 verringert. Dann geht es wieder mit der Schleife von vorne los.

```
  else:
    möglichkeiten.append(jetzige_möglichkeiten)
  nächstes_intervall = intervalle[0]
  möglichkeiten.append(intervalle)
  stack.append(nächstes_intervall[0]-
   jetziges_intervall[0])
  position += 1
return wegbeschreibung(stack)
```

Wenn “intervalle” nicht leer ist, dann fügen wir “jetzige_möglichkeiten” zu “möglichkeiten” hinzu für die nächste Runde der Schleife. Dann nehmen wir von intervalle das erste raus und speichern es in nächstes_intervall. Das ist unser nächstes Ziel. Dann fügen wir “möglichkeiten” auch noch “intervalle” hinzu, denn wir erst nur eins von den Intervallen in “intervalle” herausbekommen haben und es noch welche gibt. Dann müssen wir noch dem “stack” das Intervall hinzufügen, das geklappt hat und gehen einen Abschnitt weiter (d.h. wir addieren 1 zu “position”). Dann wiederholen wir die Schleife nochmal. Wenn wir endlich durch alle Quader durchgekommen sind und “position” größer ist als “abschnitte”, dann sind wir beim Grabmal und geben etwas zurück. Dafür nutzen wir die Definition “wegbeschreibung” mit dem Argument “stack” (siehe ganz oben) und geben das Output aus.

```
if __name__ == "__main__":
print(das_ägyptische_grabmal(sys.argv[1]))
```

Das ist der „main“-code. Wenn man das Programm auf der Kommandozeile aufruft, nimmt es die Definition “das_ägyptische_grabmal” mit dem Argument sys.argv[1] (das 1.Argument auf der Kommandozeile) und gibt dann den zurückgegebenen “text” aus.

ENDE UMSETZUNG

Aufrufen:

Das Programm ruft man auf, indem man das Terminal benutzt und die Datei Das_ägyptische_Grabmal mit python3 aufruft und nach einem Leerzeichen noch eine der grabmal0.txt bis grabmal5.txt hinschreibt. Beispiel: python3 Das_ägyptische_Grabmal grabmal2.txt

Beispiele:

grabmal0.txt

3

5

8

12

Output: Warte 5 Minuten, laufe zu Abschnitt 1.
Warte 3 Minuten, laufe zu Abschnitt 2. Warte 4 Minuten, laufe zum Grabmal.

grabmal1.txt

5

17

13

7

9

13

Output: Warte 51 Minuten, laufe zum Grabmal.

grabmal2.txt

5

170000

130000

70003

90000

130001

Output: Warte 510000 Minuten, laufe zum Grabmal.

grabmal3.txt

10

4

22

16

4

7

19

1

7

6

26

Output: Warte 244 Minuten, laufe zu Abschnitt 4. Warte 1 Minuten, laufe zu Abschnitt 5. Warte 2 Minuten, laufe zum Grabmal.

grabmal4.txt

10
72063
579
29235
43690
88575
60750
11051
86884
25781
92781

Output: Warte 3963465 Minuten, laufe zu Abschnitt 3. Warte 12325 Minuten, laufe zu Abschnitt 4. Warte 10085 Minuten,

laufe zu Abschnitt 6. Warte 3536 Minuten, laufe zu Abschnitt 8. Warte 6644 Minuten, laufe zum Grabmal.

Grabmal5.txt (Achtung! Die Datei und die Ausgabe ist sehr lang)

```
85
5988
581
375
2723
9397
7921
6791
6127
7303
4052
2328
7636
6819
7269
2227
2741
7511
3947
6552
1960
7505
7824
7215
7331
```

6540
3322
1668
5764
2253
9810
9429
3996
9095
9872
6559
5056
426
8168
5823
272
812
3632
4718
4028
6665
213
4590
1830
5287
5096
3427
3228
6517
2673

9223
7284
330
5111
6623
4694
6521
3610
5089
6010
5722
5768
1571
4746
6025
3882
9773
4519
798
8897
8237
5187
6098
6171
8682
7320
1729
7943
4890
4227

8535

Output: Warte 1828246237884 Minuten, laufe zu Abschnitt 5. Warte 1825 Minuten, laufe zu Abschnitt 6. Warte 3378 Minuten, laufe zu Abschnitt 7. Warte 3036 Minuten, laufe zu Abschnitt 10. Warte 1436 Minuten, laufe zu Abschnitt 12. Warte 2193 Minuten, laufe zu Abschnitt 13. Warte 3300 Minuten, laufe zu Abschnitt 18. Warte 3771 Minuten, laufe zu Abschnitt 23. Warte 2649 Minuten, laufe zu Abschnitt 24. Warte 1731 Minuten, laufe zu Abschnitt 28. Warte 1695 Minuten, laufe zu Abschnitt 29. Warte 1575 Minuten, laufe zu Abschnitt 30. Warte 8625 Minuten, laufe zu Abschnitt 42. Warte 3143 Minuten, laufe zu Abschnitt 43. Warte 3770 Minuten, laufe zu Abschnitt 47. Warte 542 Minuten, laufe zu Abschnitt 51. Warte 1266 Minuten, laufe zu Abschnitt 52. Warte 225 Minuten, laufe zu Abschnitt 53. Warte 1554 Minuten, laufe zu Abschnitt 55. Warte 477 Minuten, laufe zu Abschnitt 56. Warte 18 Minuten, laufe zu Abschnitt 58. Warte 2547 Minuten, laufe zu Abschnitt 59. Warte 269 Minuten, laufe zu Abschnitt 60. Warte 1267 Minuten, laufe zu Abschnitt 61. Warte 537 Minuten, laufe zu Abschnitt 62. Warte 2711 Minuten, laufe zu Abschnitt 68. Warte 144 Minuten, laufe zu Abschnitt 70. Warte 1860 Minuten, laufe zu Abschnitt 71. Warte 3178 Minuten, laufe zu Abschnitt 74. Warte 5294 Minuten, laufe zu Abschnitt 75. Warte 4882 Minuten, laufe zu Abschnitt 78. Warte 231 Minuten, laufe zum Grabmal.

Code:

```
import sys
import math
Quaderliste = []

def wegbeschreibung(stack):
        text = f"Warte {stack.pop(0)} Minuten, laufe "
        zähler = 1
        while len(stack) > 0:
        if stack[0] == 0:
stack.pop(0)
        else:
        text += f"zu Abschnitt {zähler}. Warte {stack.pop(0)} Minuten, laufe "
        zähler += 1
        text += f"zum Grabmal."
        return text

#Intervalle sind Paare von Zeitpunkten
def finde_offene_intervalle(jetziges_intervall, nächste_periode):
        nächste_iv_liste = []
        n = math.ceil(jetziges_intervall[0] / (2 * nächste_periode))
        if (2 * n - 1) * nächste_periode <= jetziges_intervall[0]:
        nächste_iv_liste.append((jetziges_intervall[0], 2 * n * nächste_periode))
        min_n = math.ceil(jetziges_intervall[0] / nächste_periode)
        if min_n % 2 == 0:
        min_n += 1
        max_n = int(jetziges_intervall[1] / nächste_periode)
        n = min_n
        while n <= max_n:
        nächste_iv_liste.append((min_n * nächste_periode, (min_n + 1) * nächste_periode))
        n += 2
        return nächste_iv_liste

def das_ägyptische_grabmal(file_name):
        file = open(file_name, "r")
        for line in file.readlines():
        Quaderliste.append(int(line))
        position = 0
        am_anfang = 1
        abschnitte = Quaderliste.pop(0)
        möglichkeiten = []
        while position < abschnitte:
        if position == 0:
        stack = [Quaderliste[0] * (2 * am_anfang - 1)]
        jetzige_möglichkeiten = [(Quaderliste[0] * (2 * am_anfang - 1), (Quaderliste[0] * 2 * am_anfang))]
        position = 1
        am_anfang += 1
        else:
                        jetzige_möglichkeiten = möglichkeiten.pop()
```

```
        if len(jetzige_möglichkeiten) == 0:
        stack.pop()
        position -= 1
        continue
        jetziges_intervall = jetzige_möglichkeiten.pop(0)
        intervalle = finde_offene_intervalle(jetziges_intervall, Quaderliste[position])
        if len(intervalle) == 0:
        stack.pop()
        position -= 1
        continue
        else:
        möglichkeiten.append(jetzige_möglichkeiten)
        nächstes_intervall = intervalle[0]
        möglichkeiten.append(intervalle)
        stack.append(nächstes_intervall[0] - jetziges_intervall[0])
        position += 1
        return wegbeschreibung(stack)

if __name__ == "__main__":
        print(das_ägyptische_grabmal(sys.argv[1]))
```