Jugendwettbewerb Informatik 2019

Runde 3

Junioraufgabe 2: Kacheln

24. November 2019

Inhaltsverzeichnis

Lösungsidee

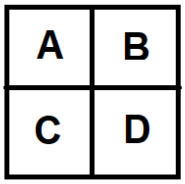
Hintergrund

Aus vierteiligen Kacheln (2x2 Bitmuster) soll eine unvollständige Landschaft vervollständigt werden. 1 steht für ein Quadrat mit Land und 0 für ein Quadrat mit Wasser an den Außenseiten. Beim Vervollständigen soll darauf geachtet werden, dass die Kacheln so zusammengefügt werden, dass Land auf Land und Wasser auf Wasser trifft.

Lösungsstrategie

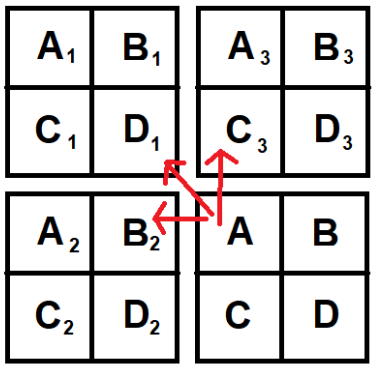
Jede Kachel besteht aus einem 2x2 Bitmuster. Es gibt also 4 verschiedene Positionen in der Kachel:

- oben links (A)
- oben rechts (B)
- unten links (C)
- unten rechts (D)



Um ein unbekanntes Element in der Kachel den Regeln gemäß vervollständigen zu können, muss mit den jeweils außen angrenzenden Elementen anderer Kacheln verglichen werden.

Ein Element vom Typ A wird dabei verglichen mit den Elementen anderer angrenzender Kacheln direkt drüber, diagonal links oben und direkt links.



- Element A wird also verglichen mit den Elementen C_3, D_1 und B_2.
- Ein Element vom Typ B wird verglichen mit den Elementen darüber (D_3), rechts und diagonal oben rechts.
- Ein Element vom Typ C wird verglichen mit den Elementen darunter, links (D_2) und diagonal unten links.
- Ein Element vom Typ D wird verglichen mit den Elementen darunter, rechts und diagonal unten rechts.

Wenn mindestens eines der Vergleichs-Elemente bekannt ist, oder falls mehrere Elemente bekannt sind (also 1 oder 0) und diese bekannten Elemente vom gleichen Typ sind (alle entweder Land (1) oder alle Wasser (0)), dann kann das Element eindeutig bestimmt werden.

Sind alle Vergleichselemente unbekannt (*), dann kann das Element nicht eindeutig bestimmt werden. Sowohl Wasser (0), als auch Land (1) wäre möglich.

Sind die angrenzenden Vergleichselemente von unterschiedlichen Typen z.B. eines Wasser (0) und ein anderes Land(1), dann liegt eine Regelverletzung vor und die Landschaft kann nicht vervollständigt werden.

Lösungsschritte

- einlesen der horizontalen und vertikalen Dimension der Landschaft (Kachel-Zeilen-Anzahl , Kachel-Spalten-Anzahl)
- einlesen der unvollständigen Kachel-Landschaft (0 für Wasser, 1 für Land, * für unbekannt)
- die Kachel-Landschaft zeilenweise durchgehen und unbekannte Elemente finden (*)

- bestimmen um welchen Typ Element es sich bei dem unbekannten Element handelt, also wo in der Kachel es sich befindet (Typ A, Typ B, Typ C, Typ D)

- die jeweiligen Vergleichselemente lokalisieren und die Vergleiche durchführen. Randkacheln haben weniger Vergleichselemente.

- ist nur ein Vergleichselement bekannt (also 0 oder 1), dann wird dem zu bestimmenden Element dieselbe Beschaffenheit zugewiesen (Land (1)oder Wasser (0))

- ist kein Vergleichselement bekannt kann nicht eindeutig bestimmt werden

- sind mehr als ein Vergleichselement bekannt und alle von der selben Sorte (nur Landelemente, oder alle nur Wasser) wird dem zu bestimmenden Element dieselbe Beschaffenheit zugewiesen. Gibt es zu wenig Information, das Element könnte also genauso gut Wasser wie Land sein, dann entscheide ich mich für Wasser. (Ich mag Inseln ;-))

- sind mehr als ein Vergleichselement bekannt und sie haben unterschiedliche Beschaffenheiten (z.B. 1x Wasser und 1x Land) dann liegt eine Regelverletzung vor und die Landschaft kann nicht vervollständigt werden.

Das Programm soll dann eine entsprechende Fehlermeldung ausgeben und abbrechen.

Umsetzung

Das Programm habe ich in Python 3 implementiert. Es hat den Namen kacheln.py. Es arbeitet mit der kompakten Version der Testdateien. Die Testdateien sollten sich im selben Verzeichnis befinden wie das Programm selbst. Der Name der einzulesenden Datei muss jeweils am Beginn des Programms entsprechend eingefügt werden.

- Zunächst liest das Programm die Dimensionen der Landschaft ein („vertical“, „horizontal“).

- dann wird die Kachel-Landschaft zeilenweise aus dem Testfile eingelesen und in einer Liste („landschaft“) gespeichert, deren Elemente wiederum Listen sind. Hierzu wurde die Funktion „feld_erstellen()“ definiert. Um später die Elemente Vergleiche bei Kacheln am Rand der Landschaft einfacher durchführen zu können, fügt die Funktion rund um die Kachel-Landschaft einen Rand aus # Symbolen zu.

			######
aus	0011	wird also	#0011#
	0111		#0111#
			######

- In den Funktionen top_left(), top_right(), bottom_left() und bottom_right() werden die jeweiligen Vergleichselemente identifiziert und mithilfe der Funktion stern_finden() versucht, das gesuchte Kachel-Element zu bestimmen. Die Funktionsnamen z.B. top_left() beziehen sich auf die Position des zu bestimmenden Elementes in der Kachel. top_left() z.B. wird für Elemente oben-links auf der Kachel verwendet.

- Die Funktion stern_finden() liefert als Ergebnis für das zu bestimmende Element entweder „1“ für Land, oder „0“ für Wasser, oder aber einen Fehlercode. Wird ein Fehlercode geliefert, bricht das Programm ab und gibt aus: 'Eine Ergänzung dieser Landschaft ist leider nicht möglich'.

Kann das Element nicht eindeutig bestimmt werden (zu wenig Information), aber ein Fehlercode lag nicht vor, dann gibt das Programm „0“ für Wasser für das Element zurück.

- konnte ein Element bestimmt werden, wird die Landschaft „landschaft“ entsprechend vervollständigt, d.h. an Stelle des * eine „1“ oder „0“ gesetzt.

- zum Schluss wird die vervollständigte Landschaft ausgegeben. Hierzu dient die Funktion „print_landschaft()“. Der Rand wird dabei zuvor wieder entfernt.

Beispiele

Links jeweils die zu vervollständigende Landschaft und rechts die durch das Programm vervollständigte Landschaft. Damit man die vom Programm bestimmten Elemente besser erkennen kann, habe ich sie blau hervorgehoben.

Beispiel 1: Ausgangsdatei map compact.txt

```
10**        1000
01**        0110
**10        0110
**11        0111
```

Beispiel 2: Ausgangsdatei map1 compact.txt

```
1001****111001          100110011110 01
0110****011001          011000000110 01
0110******1001          011000000110 01
1110******1111          111000000111 11
****000001****          111000000111 11
****100001****          000110000111 10
000110**0111**          000110000111 10
011111**1110**          011111111100 00
01111111111000          011111111110 00
11100000011001          111000000110 01
1110******1001          111000000110 01
0111******0001          011110000000 01
```

Beispiel 3: Ausgangsdatei map2 compact.txt

```
******          000000
******          000000
**00**          000000
**10**          011000
**10**          011000
**00**          000000
******          000000
******          000000
```

Beispiel 4: Ausgangsdatei map3 compact.txt

```
10000111100001                    10000111100001
11111001111000                    11111001111000
1111********00                    11111001111000
0001********11                    00011000000011
0001********11                    00011000000011
1110********01                    11100110000001
11100110000001                    11100110000001
11111110000111                    11111110000111
```

Beispiel 5: Ausgangsdatei map4 compact.txt

```
11100000011001111001111000011111  11100000011001111001111000011111
00011001111000011111100000011001  00011001111000011111100000011001
00011001111000011111100000011001  00011001111000011111100000011001
01111111111110000000011001111000  01111111111110000000011001111000
01111111******000000011001111000  01111111111110000000011001111000
11100000******011001111001111000  11100000000000110011111001111000
11100000******011001111001111000  11100000000000110011111001111000
00000000******111111111110000000  00000000000111111111111110000000
00000000000111111111111110000000  00000000000111111111111110000000
00000000111111111110000001111001  00000000111111111110000001111001
00000000111111111110****01111001  00000000111111111110000001111001
00011001111110000110****11100000  00011001111110000110000111100000
00011001**1110000110****11100000  00011001111110000110000111100000
00011001**0000000000****11111111  00011001100000000000111111111111
00011001****00000000011111111111  00011001100000000000011111111111
01111000****00000001111110000000  01111000000000000011111110000000
01111000******00000111111000000  01111000000000000011111110000000
00000000******00000000000000000  00000000000000000000000000000000
```

Beispiel 6: Ausgangsdatei map5 compact.txt

```
1111111111        Eine Erganzung dieser Landschaft ist leider nicht moglich
1000000111
100000**11
000110**00
**01**1000
**00**0000
**0001**00
**1111**11
0111111111
1110000111
```

Für dieses Beispiel war ein Vervollständigen der Landschaft nicht möglich und das Programm hat eine entsprechende Meldung ausgegeben. Lassen wir nochmal genau schauen, woran es lag:

```
1111111111        Bei den Elementen, die mit einem X markiert sind,
1000000111        kommt es zu Konflikten und es kann nicht bestimmt
100000**11        werden.
000110X*00
**01*X1000
**00*X0000
**0001X*00
**1111**11
0111111111
1110000111
```

Beispiel 7: Ausgangsdatei map6 compact.txt

Dies ist ein Beispiel um zu verdeutlichen, was das Programm macht, wenn theoretisch sowohl die Wahl Land (1) oder Wasser (0) den Regeln nach möglich wäre. Ich mag Inseln und habe mich entschieden in Fällen, wo ein Element sowohl Wasser als auch Land sein könnte, immer Wasser (0) zu wählen, in der Hoffnung, dass sdann Landschaften entstehen, die einen höheren Wasseranteil besitzen. Wenn jedoch wie in diesem Beispiel kein einziges Kachelelement bekannt ist, entsteht bei meinem Programm eine reine Wasserlandschaft (offenes Meer?).

Man hätte natürlich auch anders vorgehen können und dann z.B. zufällig zwischen Wasser und Land wählen. Die Aufgabenstellung hat dazu nichts vorgegeben, also bleibe ich bei meiner Vorliebe für Wasser.

```
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
**********                0000000000
```

Quellcode

```
#the program uses the compact version of the text files
#uses open function to open the file in read mode
file1 = open(r"map compact.txt", "r") #use the respective path of the text file you want to read
vertical = int(file1.readline())
horizontal = int(file1.readline())
kachel_landschaft = file1.readlines()
file1.close() #closing the file


def print_landschaft(landkarte):
    for o in range (1,len(landkarte)-1):
        zeile=""
        for m in range(1,len(landkarte[o])-1):
            zeile+=landkarte[o][m]
        print(zeile)


def feld_erstellen(landkarte,laenge):
    #turn the str rows of landkarte into lists and remove \n
    for i in range(0,len(landkarte)):
        landkarte[i] = list(landkarte[i])
        landkarte[i].pop()

    #add hashtags around landkarte
    stern = []
    for p in range(0,2*laenge):
        stern.append('#')

    landkarte.insert(0,stern)
    landkarte.append(stern[:])

    for i in range(0, len(landkarte)):
        landkarte[i].append('#') #add hashtag as first and last item in all lists in landkarte
        landkarte[i].insert(0,'#')
```

```
    return landkarte


def stern_finden(richtung1, richtung2, richtung3):
    if '1' in (richtung1, richtung2, richtung3):
        if '0' in (richtung1, richtung2, richtung3):
            z = False
        else:
            z = '1'
    else: z = '0'
    return(z)


def top_left(landkarte, koordinate1, koordinate2):
    oben = landkarte[koordinate1-1][koordinate2]
    links = landkarte[koordinate1][koordinate2-1]
    diaOL = landkarte[koordinate1-1][koordinate2-1]
    z = stern_finden(oben,links,diaOL)
    return z


def top_right(landkarte, koordinate1, koordinate2):
    oben = landkarte[koordinate1-1][koordinate2]
    rechts = landkarte[koordinate1][koordinate2+1]
    diaOR = landkarte[koordinate1-1][koordinate2+1]
    z = stern_finden(oben,rechts,diaOR)
    return z


def bottom_left(landkarte, koordinate1, koordinate2):
    unten = landkarte[koordinate1+1][koordinate2]
    links = landkarte[koordinate1][koordinate2-1]
    diaUL = landkarte[koordinate1+1][koordinate2-1]
    z = stern_finden(unten,links,diaUL)
    return z


def bottom_right(landkarte, koordinate1, koordinate2):
    unten = landkarte[koordinate1+1][koordinate2]
```

```
    rechts = landkarte[koordinate1][koordinate2+1]
    diaUR = landkarte[koordinate1+1][koordinate2+1]
    z = stern_finden(unten,rechts,diaUR)
    return z


landschaft = feld_erstellen(kachel_landschaft, horizontal)


zahl = True
for x in range(0,len(landschaft)):
    for y in range(0,len(landschaft[x])):
        if landschaft[x][y] == '*':
            if x%2!=0 and y%2!=0: #when it is the top left of the tile
                zahl = top_left(landschaft, x, y)
            elif x%2!=0 and y%2==0: #when it is the top right of the tile
                zahl = top_right(landschaft, x, y)
            elif x%2==0 and y%2!=0: #when it is the bottom left of the tile
                zahl = bottom_left(landschaft, x, y)
            elif x%2==0 and y%2==0: #when it is the bottom right of the tile
                zahl = bottom_right(landschaft, x, y)
            landschaft[x][y] = zahl
        if zahl == False:
            break
    if zahl == False:
        print('Eine Ergänzung dieser Landschaft ist leider nicht möglich')
        break
if zahl != False:
    print_landschaft(landschaft)
```