Jonioraufgabe 2

Vorwort

Diese Aufgabe habe ich als qualifizierende Aufgabe für mich verstanden, da ich mich bisher noch nicht in der abiturrelevanten Phase der Oberstufe befinde.

Lösungsidee

Die Karte, die eingelesen werden soll, kann auch als zweidimensionales Koordinatensystem verstanden werden, mit den Zeilen als Werten und den Spalten als Stellen. Ich setze den Ursprung (0|0) in der ersten Koordinate der Datei, also dem ersten Zeichen der ersten Zeile nach dem Header, an. Dann kann man erst einmal die Koordinaten aller Hügel einlesen, also die Koordinaten jedes „X“ auf der Karte. Daraufhin kann man diese Koordinaten durchgehen, um von ihnen aus nach einem qualifizierenden „Baulwurfbau“ zu suchen. Ist dies der Fall, können alle Koordinaten des Gebildes auf eine Sperrliste geschrieben werden, da die Baue sich nicht überlappen dürfen, und der Zähler wird natürlich erhöht. Somit kann man also bei der Suche nach den Gebilden auch gegen diese Sperrliste prüfen, da keine Figur einen Hügel mit Koordinaten der Sperrliste innehaben darf. Sind die Bedingungen für das Gebilde nicht erfüllt, kann die Koordinate, an der kein Hügel gefunden wurde, ebenfalls auf die Sperrliste gesetzt werden. Dann wird mit der nächsten Koordinate der eingelesenen Hügel fortgefahren.

Umsetzung

Um diese Aufgabe zu lösen habe ich Python gewählt. Ansammlung von Koordinaten werden als zweidimensionale Liste geschrieben, mit dem Index einer Liste als Wert und den Zahlen in den Listen als Stellen. Zuerst wird der Header der Datei eingelesen, also Höhe und Breite. Dann werden mittels einer geschachtelten Wiederholung die Koordinaten jedes Hügels eingelesen und in einer globalen Liste gespeichert. Dann wird, wieder mittels einer geschachtelten Wiederholung, jede Koordinate als Ausgangskoordinate für einen Baulwurfbau überprüft, indem die Koordinaten, die für einen Baulwurfbau benötigt werden, relativ zur Ausgangskoordinate überprüft werden. Ist an mindestens einer der benötigten Koordinaten kein Hügel, wird die Überprüfung abgebrochen und die besagte Koordinate wird in ´blacklist´ gespeichert. Die Überprüfung einer Koordinate läuft sowohl gegen die Koordinaten der eingelesenen Hügel, als auch die ´blacklist´. Wird ein Baulwurfbau gefunden, werden alle Koordinaten der Hügel dieses Baus in der Blacklist gespeichert und der Zähler um 1 erhöht. Des Weiteren werden die Koordinaten zur späteren Ausgabe einer ‘aufgeräumten‘ Karte abgespeichert. Wurden alle Koordinaten abgearbeitet, werden die Koordinaten aller Baue in ´ausgabe.txt´ gespeichert, was also eine Karte mit den Standorten der Baue ergibt. Außerdem wird der Zähler ausgegeben.

Beispiele:

Als Beispiele habe ich die gegebenen Beispiele der BWInf-Materialien gewählt:

$ python3 junioraufgabe2.py karte0.txt liefert:

3

$ python3 junioraufgabe2.py karte1.txt liefert:

37

$ python3 junioraufgabe2.py karte2.txt liefert:

32

$ python3 junioraufgabe2.py karte3.txt liefert:

318

$ python3 junioraufgabe2.py karte4.txt liefert:

3189

$ python3 junioraufgabe2.py karte5.txt liefert:

16

Quellcode:

```
# Zum Einlesen von Kommandozeilenargumenten

import sys

blacklist = []



def einlesen(breite, hoehe):
    """Erstellt eine zweidimensionale Liste der Koordinaten, an denen sich Hügel befinden."""
    # Huegel ist auch außerhalb dieser Funktion relevant, also wird diese Variable global gebraucht
    global huegel
    # Initialisierung der Hügelkoordinaten
    huegel = []
    # Geh jede Zeile der Datei durch und speichere die Koordinaten, an denen sich ein ´X´ befindet
    for einlesenY in range(0, hoehe):
```

```
        # ´huegel´ und ´blacklist´ müssen genau so lang sein, wie die Karte y-Koordinaten hat
        huegel.append([])
        blacklist.append([])
        zeile = f.readline().strip("\n")
        for einlesenX in range(0, breite):
            if zeile[einlesenX] == "X":
                huegel[einlesenY].append(einlesenX)
    return huegel



def gebilde(ausgangX, ausgangY):
    """Bestimmt, ob die Figur, die bei ´ausgangX´/´ausgangY´ beginnt, ein legitimer Baulwurfbau ist."""
    # Initalisierung der Liste der Koordinaten, für die diese Figur auf einen Baulwurfbau passt
    moeglich = []
    # Die Liste bekommt genau so viele leere Listen als Inhalt, wie es Y-Koordinaten in der Datei gibt
    for moeglichY in range(0, hoehe):
        moeglich.append([])
    # Ist der Beginn der Figur weniger als 3 Einheiten vom Rand in y-Richtung entfernt, gib ´False´ aus
    if ausgangY+3 > hoehe-1:
        return False
    # Gibt es zwei weitere Hügel in x-Richtung?
    for mitlauf in range(0, 3):
        # Ist diese Koordinate bereits ausgeschlossen worden, gib ´False´ aus
        if ausgangX+mitlauf in blacklist[ausgangY]:
            return False
        # Ist an dieser Koordinate kein Hügel, gib False aus
        elif ausgangX+mitlauf not in huegel[ausgangY]:
            blacklist[ausgangY].append(ausgangX+mitlauf)
            return False
        # Ist dort ein Hügel, speichere diese Koordinate in ´moeglich´
        else:
            moeglich[ausgangY].append(ausgangX+mitlauf)
    # Gibt es drei Hügel 3 Y-Koordinaten unterhalb der ersten 3?
    for mitlauf in range(0, 3):
        if ausgangX+mitlauf in blacklist[ausgangY+3]:
            return False
```

```
            elif ausgangX+mitlauf not in huegel[ausgangY+3]:
                blacklist[ausgangY+3].append(ausgangX+mitlauf)
                return False
            else:
                moeglich[ausgangY+3].append(ausgangX+mitlauf)
        # Gibt es zwei Hügel unterhalb der Ausgangskoordinate?
        for mitlauf in range(1, 3):
            if ausgangX in blacklist[ausgangY+mitlauf]:
                return False
            elif ausgangX not in huegel[ausgangY+mitlauf]:
                blacklist[ausgangY+mitlauf].append(ausgangX)
                return False
            else:
                moeglich[ausgangY+mitlauf].append(ausgangX)
        # Gibt es zwei Hügel parallel dazu?
        for mitlauf in range(1, 3):
            if ausgangX+2 in blacklist[ausgangY+mitlauf]:
                return False
            elif ausgangX+2 not in huegel[ausgangY+mitlauf]:
                blacklist[ausgangY+mitlauf].append(ausgangX+2)
                return False
            else:
                moeglich[ausgangY+mitlauf].append(ausgangX+2)
        return moeglich


def finden(karte):
    """Ruft ´gebilde´ auf und gibt eine Karte mit dem Ergebnis aus."""
    # Initialisierung der Anzahl
    baulwuerfe = 0
    # Initialisierung der Lösungskoordinaten
    baul = []
    # Die Liste bekommt genau so viele leere Listen als Inhalt, wie es Y-Koordinaten in der Datei gibt
    for baulY in range(0, hoehe):
        baul.append([])
    # Gehe die durch ´einlesen´ bestimmten Hügel durch und überprüfe, ob sie Teil eines Baus sind
```

```
    for findenY in range(0, len(karte)):
        for findenX in karte[findenY]:
            # Speichere die Koordinaten des Baus in ´koordinaten´
            koordinaten = gebilde(findenX, findenY)
            if koordinaten:
                # Sind diese nicht ´False´, erhöhe ´baulwuerfe´ um 1
                baulwuerfe += 1
                # Ergänze die Lösungskoordinaten ´baul´ und die Sperrliste ´blacklist´ mit den Koordinaten des Baus, da diese sich nicht mit einem
                # anderen schneiden dürfen
                for blackY in range(0, len(blacklist)):
                    for koordX in koordinaten[blackY]:
                        baul[blackY].append(koordX)
                        if koordX not in blacklist[blackY]:
                            blacklist[blackY].append(koordX)
    # Damit man sich auch vorstellen kann, wo die echten Baue sind, wird eine Karte ausgegeben
    w = open("ausgabe.txt", "w")
    for y in range(0, hoehe):
        for x in range(0, breite):
            if x in baul[y]:
                w.write("X")
            else:
                w.write(" ")
        w.write("\n")
    return baulwuerfe


# Wenn genau zwei Argumente gegeben wurden, lies die ersten beiden Zeilen der gegebenen Datei ein und ruf
# finden damit auf, sonst weise auf die korrekte Syntax hin
if len(sys.argv) == 2:
    f = open(sys.argv[1], "r")
    global breite
    global hoehe
    breite = int(f.readline().strip("\n"))
    hoehe = int(f.readline().strip("\n"))
    print(finden(einlesen(breite, hoehe)))
```

```
    f.close()
else:
    print("Korrekte Nutzung: jonioraufgabe2.py <Textdatei>")
```