Wortsuche

Team-ID: 00940

Team: Lovely_Infestation

Bearbeiter/-innen dieser Aufgabe: Lars Noack

18. November 2020

Inhaltsverzeichnis

Lösungsidee

Wenn diese Dokumentation etwas kürzer als die anderen ausfällt, liegt das daran, dass die Hälfte des Algorithmus exakt der gleiche wie in der aufgabe Vollgeladen ist.

Ein Wortgitter besteht aus 2 Teilen.

1. Die "Wörtermaske". Nur die Wörter ohne Buchstaben dahinter.
    - um diese zu generieren, muss ich dafür sorgen, dass die Wörter sich nicht überschneiden.
    - dies mach ich wie in "Vollgeladen" mit einem Datenbaum

2. Der Rest. Alle zufälligen Buchstaben
    - Dafür fülle ich jede leere Zelle auf
    - schaue, ob in der Zeile und Reihe ein neues Wort entstanden ist
    - wenn nicht mache weiter
    - wenn probiere einen anderen Buchstaben.

Schwierigkeitsgrad

Die Aufgabe in einem Wortgitter ist es, ein Wort unter vielen Buchstaben zu finden. Wenn in dem Wortgitter mehr-nur Buchstaben, die in den jeweiligen Wörtern sind befinden, ist das Wortgitter deutlich schwerer, als wenn sich keine Buchstaben aus den Wörtern sonst im Wortgitter zu finden ist.

Umsetzung

Die Wörtermaske

Wie vorhin erwähnt habe ich den Datenbaum in Vollgeladen ausführlich dokumentiert.

Die zufälligen Buchstaben

Ich habe ein Grid mit Wörtern aber ohne Buchstaben.

E	**-**	**V**	**-**	**-**
V	-	O	T	-
A	-	R	O	R
-	-	-	R	A
-	-	-	F	D

Dann fange ich an einem beliebigen Punkt an (hier jetzt bei 0/0 bzw. oben rechts) und arbeite mich durch das Grid.

Bei jeder Zelle wird das gemacht:

- wenn die Zelle leer ist,
  - (dies ist rekursiv)
  - erzeuge eine Zufallszahl von 0 bis 100 und wenn diese Zahl größer als die Schwierigkeit(0-100) ist, dann
    - setzte an dieser Stelle einen zufälligen Buchstaben aus dem Alphabet ein.
  - sonst
    - setzte and dieser Stelle einen zufälligen Buchstaben aus einem Buchstabenpool in dem alle Buchstaben, die auch in den Wörtern drin sind ein.
  - dann wird überprüft, ob mit dem Einfügen dieses Buchstabens ein neues Wort in der Reihe und Spalte des Wortgitters entstanden ist. Wenn dies der Fall ist, erzeuge in der Zelle einen anderen Buchstaben (Rekursion)

Beispiele

In diesem Dokument sind nur Beispiel 1-3. Der rest hätte nicht auf die Seite gepasst. Der Rest ist hier zu finden.

TORF; VOR; RAD; EVA

schwierigkeit 0

E	**L**	**V**	**C**	**I**
V	T	O	T	J
A	C	R	O	R
A	M	F	R	A
Z	T	U	F	D

schwierigkeit 25

V	**O**	**R**	**E**	**I**
X	V	B	K	R
T	O	R	F	A
X	X	Y	X	D
E	V	A	E	V

schwierigkeit 50

R	**C**	**X**	**R**	**T**
V	E	V	A	O
O	X	R	F	R
R	O	A	R	F
J	N	D	W	Y

schwierigkeit 75

V	**F**	**O**	**O**	**V**
E	V	A	D	R
T	O	R	F	R
R	E	J	C	A
D	V	O	R	D

schwierigkeit 100

E	**V**	**R**	**F**	**T**

E	V	R	F	T
A	T	V	A	R
T	O	O	V	A
R	R	R	R	D
R	F	E	V	A

INFO; EIN; UND; DA; ER; DU

schwierigkeit 0

Q	Y	C	J	I	V
H	E	I	N	N	R
M	F	X	S	F	Y
C	E	R	T	O	R
R	D	A	Q	A	H
U	N	D	T	D	U

schwierigkeit 25

H	I	U	N	D	N
D	V	X	Q	B	U
A	X	E	L	X	N
U	E	R	D	E	E
U	R	O	U	F	I
I	I	N	F	O	N

schwierigkeit 50

Q	V	D	U	N	D
N	T	U	E	N	N
S	F	G	I	D	D
E	E	B	N	A	T
R	N	C	A	L	U
I	N	F	O	I	A

schwierigkeit 75

E	N	A	D	I	H
P	E	F	T	D	D
E	I	U	O	E	U
N	N	N	I	N	I
D	A	D	E	E	E
I	N	F	O	R	U

schwierigkeit 100

U	D	U	U	N	D
I	U	E	N	D	R
E	D	A	R	I	I
F	N	E	R	N	N
I	D	D	N	F	F
D	O	E	I	N	O

BILDSCHIRM; FESTPLATTE; TASTATUR; COMPUTER; MAUS; USB

schwierigkeit 0

Q	T	H	C	R	K	F	E	E	B
U	S	B	O	F	M	E	J	E	I
A	Z	Z	M	T	W	S	V	Y	L
R	F	K	P	A	O	T	A	P	D
O	Y	M	U	S	B	P	L	F	S
M	G	A	T	T	K	L	N	Z	C
X	G	U	E	A	W	A	U	O	H
D	O	S	R	T	R	T	X	O	I
O	D	C	B	U	D	T	J	Q	R
E	J	P	Z	R	R	E	I	J	M

schwierigkeit 25

V	B	U	R	M	F	G	I	G	Q
U	I	A	C	C	E	Q	Q	S	J
Q	L	L	O	P	S	L	G	T	Q
V	D	B	M	I	T	N	E	A	M
P	S	S	P	R	P	V	H	S	A
M	C	I	U	U	L	S	E	T	U
X	H	I	T	J	A	V	L	A	S
Z	I	L	E	J	T	W	G	T	R
F	R	W	R	T	T	T	P	U	H
Y	M	U	S	B	E	G	H	R	P

schwierigkeit 50

J	Z	D	P	M	R	R	K	T	S
B	I	L	D	S	C	H	I	R	M
L	R	C	O	M	P	U	T	E	R
U	P	U	I	I	T	U	I	A	W
T	A	S	T	A	T	U	R	P	E
E	R	M	J	D	T	Y	C	X	D
F	E	S	T	P	L	A	T	T	E
E	E	M	T	M	A	U	S	T	T
U	S	V	I	W	U	N	T	C	O
U	S	B	A	T	S	D	U	U	S

schwierigkeit 75

I	R	T	Z	A	M	B	F	U	F
D	E	C	A	A	T	I	E	P	C
B	T	O	P	T	S	L	S	U	I
R	E	M	P	A	E	D	T	T	A
W	U	P	U	S	C	S	P	M	I
N	S	U	P	T	S	C	L	A	C

I	R	T	Z	A	M	B	F	U	F
C	B	T	S	A	R	H	A	U	E
P	K	E	B	T	B	I	T	S	D
R	V	R	T	U	M	R	T	S	F
A	A	D	M	R	X	M	E	B	G

schwierigkeit 100

T	I	F	A	F	F	B	T	P	M
A	M	E	P	W	L	I	A	C	S
F	A	S	R	U	A	L	S	O	A
R	C	T	T	S	T	D	T	M	M
T	A	P	F	E	S	S	A	P	A
I	T	L	B	U	M	C	T	U	U
U	U	A	E	I	A	H	U	T	S
U	M	T	T	H	A	I	R	E	C
S	A	T	O	R	E	R	S	R	T
B	F	E	E	E	L	M	B	T	U

Quellcode

```
import random
from anytree import Node


class WordData:
    def __init__(self, path: str):
        # read and declaring some variables
        with open(path, 'r', encoding='utf-8') as words_file:
            word_list_raw = words_file.read().split('\n')[:-1]

            self.GRID_SIZE = tuple(map(int, word_list_raw[0].split(' ')))
            self.GRID_X, self.GRID_Y = self.GRID_SIZE
            self.WORD_LIST = sorted(word_list_raw[2:], key=len, reverse=True)

            self.letters = []

            for word in self.WORD_LIST:
                for char in word:
                    self.letters.append(char)
```

```
self.CURRENT_WORD_KEY = 'current word'
self.POS_KEY = 'pos'
self.EXIF_KEY = 'exif'
self.NEXT_WORD_KEY = 'next word'
self.EMPTY_CELLS_KEY = 'empty cells'
self.POSSIBLE_GUESSES_KEY = 'possible guesses'

self.root = Node({self.EMPTY_CELLS_KEY: list(range(self.GRID_Y * self.GRID_X)),
                  self.EXIF_KEY: {}})

self.current_node = self.root

# iterates through word list
i = 0
while i < len(self.WORD_LIST):
    word = self.WORD_LIST[i]
    self.current_node.name[self.EXIF_KEY][self.CURRENT_WORD_KEY] = word
    if self.POSSIBLE_GUESSES_KEY not in self.current_node.name[self.EXIF_KEY]:
        possible_pos = self.get_possible_positions(word, self.current_node.name[self.EMPTY
        self.current_node.name[self.EXIF_KEY][self.POSSIBLE_GUESSES_KEY] = possible_pos
    else:
        possible_pos = self.current_node.name[self.EXIF_KEY][self.POSSIBLE_GUESSES_KEY]


    # if no remaining places to put word then go one node back in tree
    if len(possible_pos['x']) <= 0 and len(possible_pos['y']) <= 0:
        i += -1
        self.current_node = self.current_node.parent
        continue
    else:
        # go to random position and get all possible positions from this node on
        if len(possible_pos['x']) <= 0:
            possible_pos.pop('x')
        elif len(possible_pos['y']) <= 0:
            possible_pos.pop('y')

        dir_keys = list(possible_pos)
        dir_key = dir_keys[random.randrange(len(dir_keys))]
        pos = possible_pos[dir_key][random.randrange(len(possible_pos[dir_key]))]

        new_cells = self.delete_cells(word, pos, dir_key, self.current_node.name[self.EMPT

        node_dict = {
            self.CURRENT_WORD_KEY: word,
            self.POS_KEY: (dir_key, pos),
            self.EMPTY_CELLS_KEY: new_cells,
            self.EXIF_KEY: {}
        }

        self.current_node = Node(node_dict, parent=self.current_node)
        i += 1

word_positions_raw = {}
for i in range(len(self.WORD_LIST)):
    word_positions_raw[self.current_node.name[self.CURRENT_WORD_KEY]] = self.current_node.
    self.current_node = self.current_node.parent

self.grid = [[None] * self.GRID_X for _ in range(self.GRID_Y)]
for key in word_positions_raw:
    pos_x = word_positions_raw[key][1] % self.GRID_Y
    pos_y = int(word_positions_raw[key][1] / self.GRID_Y)
```

```
            if word_positions_raw[key][0] == 'x':
                for i, char in enumerate(key):
                    self.grid[pos_x + i][pos_y] = char
            else:
                for i, char in enumerate(key):
                    self.grid[pos_x][pos_y + i] = char

    def delete_cells(self, word, pos, dir, empty_cells):
        if dir == 'x':
            for i in range(pos, pos + len(word), 1):
                empty_cells.remove(i)

        else:
            for i in range(pos, pos + (len(word) * self.GRID_Y), self.GRID_Y):
                empty_cells.remove(i)

        return empty_cells

    def get_possible_positions(self, word, empty_cells):

        x_list = []
        y_list = []


        for empty_cell in empty_cells:
            if empty_cell % self.GRID_Y + (len(word) - 1) < self.GRID_Y:
                add = True
                for i in range(empty_cell, empty_cell + len(word), 1):
                    if i not in empty_cells:
                        add = False
                        break
                if add:
                    x_list.append(empty_cell)

            if empty_cell + (self.GRID_Y * (len(word) - 1)) in empty_cells:
                add = True
                for i in range(empty_cell, empty_cell + (len(word) * self.GRID_Y), self.GRID_Y):
                    if i not in empty_cells:
                        add = False
                        break
                if add:
                    y_list.append(empty_cell)

        return {'x': x_list, 'y': y_list}


def randomize_list(ordered_list_ref: list):
    ordered_list = list(ordered_list_ref)
    random_list = []
    for i in range(len(ordered_list)):
        rand_ind = random.randrange(len(ordered_list))
        random_list.append(ordered_list[rand_ind])
        ordered_list.pop(rand_ind)

    return random_list


def get_word_count(word_list: list, grid: list):
    word_counter = 0
    for word in word_list:
        if word in grid[0] or word in grid[1]:
            word counter += 1
```

```
    return word_counter


def letter_is_valid(word_list: list, letter: str, row_int: int, column_int: int, grid):
    current_row = ''
    current_col = ''
    after_row = ''
    after_col = ''

    # get the row and column to check for new words
    for r in range(len(grid)):
        row = grid[r]

        for c in range(len(row)):
            column = row[c]
            if row_int == r and row[c] is not None:
                current_row += row[c]
                after_row += row[c]

            if column_int == c and row[c] is not None:
                current_col += row[c]

                after_col += row[c]

            if row_int == r and column_int == c:
                after_row += letter
                after_col += letter

    # compare the frequency of words
    prev_word_count = get_word_count(word_list, (current_row, current_col))
    after_word_count = get_word_count(word_list, (after_row, after_col))

    if after_word_count != prev_word_count:
        return False
    else:
        return True


def fill_empty_cells(word_obj: WordData, difficulty: int):
    grid = []
    for row in word_obj.grid:
        grid.append(row)
    letters = word_data.letters
    words = word_data.WORD_LIST

    # iterates through whole list, filling every empty elem with rand letters
    for row in range(len(grid)):
        for column in range(len(grid[row])):
            if grid[row][column] is None:

                # either fill in letter, that is is contained in the words or spawn one that isn't
                if random.randint(0, 100) < difficulty:

                    possible_letters = randomize_list(letters)

                    for letter in possible_letters:
                        # checks if it creates a new word if letter is placed here
                        if letter_is_valid(words, letter, row, column, grid):
                            # place letter
                            grid[row][column] = letter
                            break
```

```
                    if grid[row][column] is None:
                        # spawn random letter
                        grid[row][column] = chr(random.randrange(65, 91))

                else:
                    # try to spawn random letter
                    grid[row][column] = chr(random.randrange(65, 91))

    return grid


for FILE in range(6):
    PATH_INPUT = f'worte{FILE}.txt'

    # get the grid
    word_data = WordData(PATH_INPUT)
    print(f"<h3>{'; '.join(word_data.WORD_LIST)}</h3>")
    print("\n")
    for DIFFICULTY in range(0, 101, 25):
        # fill remaining spaces with random letters
        grid = fill_empty_cells(word_data, DIFFICULTY)


        print(f"*schwierigkeit {DIFFICULTY}*\n\n")

        # print results
        for i, row in enumerate(grid):
            print(' | '.join(row))
            if i == 0:
                print("|---"*len(row))
        print("\n\n\n")
        word_data = WordData(PATH_INPUT)
```

Dependencies

Die einzige library, die installiert werden muss ist anytree. Dafür gibt es 3 Möglichkeiten:

1. öffnen Sie cmd und geben Sie `pip install anytree` ein
2. öffnen Sie cmd und gehen mit cd in das Laufwerk, oder öfnnen es gleich in Power Shell und geben sie `pip install -r requirements.txt` ein
3. führen sie die `install Dependencies.bat` aus