Aufgabe 2

Teilnahme-ID: 00357

Alex Sterneckert

Informatik 4, SRZ

20. November 2020

Lösungsidee

Die Umsetzung sollte mittels Backtracking erfolgen. Der Algorithmus sollte alle Permutationen rekursiv generieren, wobei er diese dabei aus allen an dieser Stelle möglichen aus der Datei eingelesenen Puzzleteilen generiert. Wenn das Puzzleteil an dieser Stelle passt soll recursiv geschaut werden, ob man mit diesem Teil an dieser Stelle eine Lösung finden kann. Wenn schon wird diese ausgegeben. Wenn nicht wird das Teil rotiert und das Prozedere wird wiederholt. Wenn es immer noch nicht geht, wird nochmal rotiert und wenn das immer noch nicht passt, wird das Teil zurückgesetzt und das nächste mögliche Teil wird genommen, und so weiter bis eine Lösung gefunden wird. In dem Fall ist eine Lösung möglich und diese wird angezeigt.

Ich bin dabei von folgenden Indexen für die Struktur des Puzzles ausgegangen:

0

1 2

3 4 5

6 7 8 9

Umsetzung

Die Puzzleteile werden mit Objekten der Klasse Piece dargestellt, was die Atrribute a, b und c (welche die Figuren auf den 3 Seiten repräsentieren), x (was die ID des Puzzleteils speichert) und r (was die rotation gegen den Uhrzeigersinn speichert). Alle fünf Attribute sind Integer.

Zuerst wird im Driver nach dem Pfad zu der Textdatei mit den Puzzleteildaten gefragt. Diese werden dann abgespeichert, die ersten beiden Werte als Attributwerte der Klasse Puzzle und dann wird durch die restlichen Zeilen durchiteriert und deren Werte werden in ein jeweils neu generiertes Piece Objekt als a, b und c Werte. Diese werden in eine Liste gespeichert, deren Index +1 als ID des Puzzleteils im x Attribut des Puzzleteils eingespeichert. Danach wird die eigentliche Backtracking Methode recursion aufgerufen, mit drei Parametern: i = 0 (der Index), sol = [] (die Lösungsliste) und puzzle_piece = piece (die Liste der Puzzleteile). Zuerst kommt die Austrittsbedingung: wenn i gleich 9 wird sol wiedergegeben. Danach kommt eine for-Schleife von 0 bis 9 (die Anzahl der Puzzleteile). Jedes mal wird die Methode piece_not _puzzle aufgerufen, welche prüft ob das Teil in puzzle_piece an diesem Index schon in sol ist. Wenn schon wird bei dieser Iteration nichts getan. Wenn nicht so wird zuerst das Teil hinten an sol angehängt wenn das Teil an diese Stelle passt, was die Methode fit überprüft. Dannach wird die Methode recursion nochmal aufgerufen, nur mit i = i + 1. Wird immer sich selbst immer wieder aufrufen, bis entweder i gleich 9 ist und sol wiedergegeben wird oder es keine weiteren Konfigurationen von Puzzleteilen gibt, die möglich sind, dann wird eine leere Liste [] wiedergegeben. Wenn eine Liste mit Elementen wiedergegeben wurde, wird sol wiedergegeben. Ansonsten wird das letzte Element gelöscht. Dieser Prozess wird mit dann wiederholt, nur dass das aktuelle Puzzleteil current die Methode rotate aufruft, welche ein Piece wiedergibt, wessen Seiten gegen den Uhrzeigersinn rotiert wurden (dabei wird r erhöht). Dannach geschieht dies wieder, nur dass current die rotate Methode zwei mal aufruft. Nach der for-Schleife wird am Schluss eine leere Liste wiedergegeben. Am Ende wird ausgegeben, dass keine Lösung

existiert wenn die Liste leer war, ansonsten dass eine existiert. Dann wird durch die Lösungsliste durchiteriert und x und r des Puzzleteils wiedergegeben.

Beispiele

Z.B. die erste Beispieldatei mit den Text:

3

9

-1 -2 1

2 -1 -1

-1 -2 2

-1 3 1

2 -3 3

-1 3 -2

2 2 -1

-3 2 -1

-2 1 -3

Hierbei ist 3 die Anzahl an Figuren, welche jeweils negativ und positiv sein können. Neun ist die Anzahl an Teilen. Dannach sind die Seitenwerte von den neun Teilen gegeben.

Wenn man die Python Datei „main.py“ compiliert, wird zuerst wird nach dem Dateipfad für die Textdatei gefragt.

```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.exe "C:/Users/Admin/Documents/BWINF_NEW/10.11.2020/main_bibo - Kopie.py"
File path and name:
C:\Users\Admin\Documents\BWINF_NEW\10.11.2020\demo_0.txt
```

Dannach erscheint in diesem Fall, dass es keine Lösung gab. Ansonsten wird die Lösung hier ausgegeben.

```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.exe "C:/Users/Admin/Documents/BWINF_NEW/10.11.2020/main_bibo - Kopie.py"
File path and name:
C:\Users\Admin\Documents\BWINF_NEW\10.11.2020\demo_0.txt
no solution

Process finished with exit code 0
```

Quellcode

```
#     0
#    1 2
#   3 4 5
# 6 7 8 9


class Puzzle:
    def __init__(self, types, piece_nr):
        self.types = types
        self.pieceNr = piece_nr


class Piece:
    def __init__(self, new_a, new_b, new_c, new_x, new_r):
        self.a = new_a  # right / upside-down left
        self.b = new_b  # left / upside-down right
        self.c = new_c  # bottom / upside-down top
        self.x = new_x  # piece_number
        self.r = new_r  # rotation counter-clockwise

    def rotate(self):  # counter-clockwise rotation
        n_a = self.c
        n_b = self.a
        n_c = self.b
        n_r = self.r + 1
        return Piece(n_a, n_b, n_c, self.x, n_r)


def r_reset_rec(m_a, m_b, m_c, m_x, m_r):  # reset rotation
    if m_r == 0:
        return Piece(m_a, m_b, m_c, m_x, m_r)
    r_reset(m_b, m_c, m_a, m_x, m_r - 1)


def r_reset(m_a, m_b, m_c, m_x, m_r):  # reset rotation
    while m_r > 0:
        tmp = m_a
        m_a = m_b
        m_b = m_c
        m_c = tmp
        m_r = m_r - 1
    return Piece(m_a, m_b, m_c, m_x, m_r)


def index_exists(a_list, index):
    if index < len(a_list):
        return True
    return False


def fit(pieces, piece_origin, i):  # check 1 piece
    piece_main = r_reset(piece_origin.a, piece_origin.b, piece_origin.c,
piece_origin.x, piece_origin.r)
```

```
    y = 0  # 0
    if y == len(pieces):
        return True

    if i == y:
        return True

    y += 1  # 1
    if y == len(pieces):
        return False

    if i == y:
        return True

    y += 1  # 2
    if y == len(pieces):
        return False

    if i == y and piece_origin.a + pieces[1].a == 0 \
            and piece_main.c + pieces[0].c == 0:
        return True

    y += 1  # 3
    if y == len(pieces):
        return False

    if i == y and piece_origin.b + pieces[2].b == 0:
        return True

    y += 1  # 4
    if y == len(pieces):
        return False

    if i == y:
        return True

    y += 1  # 5
    if y == len(pieces):
        return False

    if i == y and piece_origin.a + pieces[4].a == 0 \
            and piece_main.c + pieces[1].c == 0:
        return True

    y += 1  # 6
    if y == len(pieces):
        return False

    if i == y and piece_origin.b + pieces[5].b == 0:
        return True

    y += 1  # 7
    if y == len(pieces):
        return False

    if i == y and piece_origin.a + pieces[6].a == 0:
        if piece_main.c + pieces[3].c == 0:
            return True

    y += 1  # 8
```

```
    if y == len(pieces):
        return False

    if i == y and piece_origin.b + pieces[7].b == 0:
        return True

    return False


def piece_not_puzzle(puzzle, piece):
    for i in range(0, len(puzzle)):
        if puzzle[i].x == piece.x:
            return False
    return True


def recursion(i, sol, puzzle_piece):
    if i == 9:
        return sol

    for j in range(0, 9):
        current = puzzle_piece[j]
        if piece_not_puzzle(sol, current):
            if fit(sol, puzzle_piece[j], j):
                sol.append(puzzle_piece[j])
                if not recursion(i + 1, sol, puzzle_piece) == []:
                    return sol
                sol.pop()

            if fit(sol, current.rotate(), j):
                sol.append(current.rotate())
                if not recursion(i + 1, sol, puzzle_piece) == []:
                    return sol
                sol.pop()

            if fit(sol, current.rotate().rotate(), j):
                sol.append(current.rotate().rotate())
                if not recursion(i + 1, sol, puzzle_piece) == []:
                    return sol
                sol.pop()

    return []


def output(final):
    if not final:
        print("no solution")
    else:
        print("\nSolution available:")
        for z in range(0, 9):
            print("Position ", z + 1, ", Piece", final[z].x, ", rotated counter-
clockwise", final[z].r, "times.")


# driver
# file read
def main():
    print("File path and name: ")
    x = input()
    file = open(x, "r")
```

```
    count_types = int(file.readline())
    count_pieces = int(file.readline())
    puzzle = Puzzle(count_types, count_pieces)
    piece = []
    for x in range(puzzle.pieceNr):
        cur_line = file.readline()
        tokens = cur_line.split()
        a = int(tokens[0])
        b = int(tokens[1])
        c = int(tokens[2])
        piece_n = Piece(a, b, c, x+1, 0)
        piece.append(piece_n)
    file.close()
    # output
    solution = recursion(0, [], piece)
    output(solution)
    # driver end


if __name__ == '__main__':
    main()
```