Aufgabe 1: Arukone

Team-ID: 00054

Team-Name: K7Bots

Bearbeiter/-innen dieser Aufgabe:
Konstantin Kuntzsch

18. November 2023

Inhaltsverzeichnis

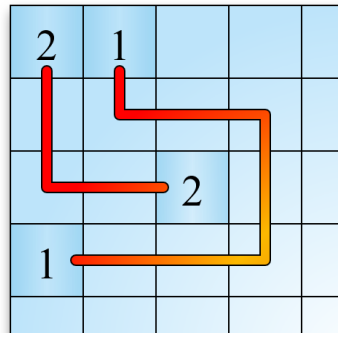


Abbildung 1: Die „Fallen“-Struktur und die mögliche Lösung

1 Lösungsidee

Die Lösungsidee besteht darin, zuerst eine spezielle Struktur von Zahlen in eine Ecke des Gitters zu platzieren. Der 2. Teil des Gitters wird dann so mit Zahlen gefüllt, dass dieser Teil für sich gesehen immer lösbar ist.

Der 1. Teil besteht dabei aus einer Struktur, die, wenn sie auf die richtige Art in einer Ecke platziert wird, zwar lösbar ist, an der das Lösungsprogramm aber scheitert.

Die Struktur, inklusive der theoretisch möglichen Lösung, ist in Abb. 1 dargestellt.

Der Lösungsalgorithmus funktioniert, indem er das Paar gleicher Zahlen heraussucht, bei dem der kürzestmögliche Weg zwischen den beiden Zahlen unter allen Paaren am kleinsten ist. Haben zwei Paare einen gleich langen kürzesten Weg, wird das Paar mit der kleineres Zahl bevorzugt. Die Zahlen des so ausgewählten Paares werden nun auf dem kürzestmöglichen Weg miteinander verbunden. Dieser Vorgang wird solange wiederholt, bis kein Paar übrig ist.

Im Fall der „Fallen“-Struktur bedeutet das, dass zuerst die beiden Zahlen 1 miteinander verbunden werden, da diese, wie die Zahlen 2, den Abstand 4 haben, die 1er jedoch aufgrund der kleineren Zahl priorisiert werden. Dadurch wird der, aufgrund der Platzierung in der Ecke, einzige Weg, die 2en zu verbinden, blockiert, sodass das Rätsel unlösbar wird, wie in Abb. 2 sichtbar ist.

Dadurch wird ein Rätsel erstellt, welches zwar lösbar ist, jedoch durch das Lösungsprogramm nicht gelöst werden kann, die Bedingungen der Aufgabenstellung werden also erfüllt.

Um mehr Variationsmöglichkeiten zu erzeugen, werden 1. die „Fallen“-Struktur gespiegelt, gedrekt und dementsprechend in einer anderen Ecke platziert und 2. die anderen Paare zufällig angeordnet.

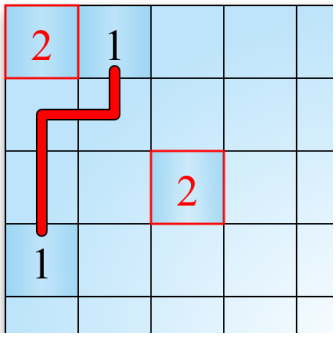


Abbildung 2: Die „Fallen“-Struktur und der Versuch des Lösungsprogramms

2 Umsetzung

Die Umsetzung dieser Aufgabe erfolgt in Python, einer Programmiersprache, die für ihre Einfachheit und Lesbarkeit bekannt ist. Der Code ist in folgende Bereiche unterteilt:

2.1 Datenstrukturen

SchemaPunkt

Die SchemaPunkt-Struktur repräsentiert einen Eintrag in einem Schema, bestehend aus X- und Y-Koordinate sowie der entsprechenden Zahl am angegebenen Gitterfeld.

Schema

Die Schema-Struktur repräsentiert ein Schema wie die „Fallen“-Struktur, bestehend aus einer Liste von SchemaPunkten.

Arukone

Die Arukone-Struktur repräsentiert ein fertiggestelltes Arukone-Rätsel, bestehend aus einer Liste von Zeilen, die jeweils als eine Liste von Zahlen dargestellt werden.

2.2 Eigentlicher Algorithmus

Die Funktion `erstelle_arukone()` ist dafür zuständig, ein Arukone-Rätsel der angegebenen Größe zu erstellen. Dabei wird die Funktion `erstelle_falle()` aufgerufen, welche eine zufällige Fallenstruktur erzeugt.

2.2.1 Erstellen der Fallenstruktur

Das Erstellen der Fallenstruktur erfolgt, indem zuerst eine lokale Variable `schema` mit einer Ursprungsvariation der Fallen-Struktur initialisiert wird.

Außerdem werden die Variablen `ist_oben` (mit dem Wert `True`) und `ist_links` (mit dem Wert `False`) initialisiert, welche angeben, in welcher Ecke die Struktur platziert werden soll.

Im Folgenden werden diese drei Variablen dann zufällig verändert, um zufällig eine der acht möglichen Variationen der Struktur zu erzeugen. Dabei wird in drei voneinander unabhängigen Schritten jeweils zufälllig entschieden, ob eine bestimmte Transformation auf das Schema und die Variablen angewendet wird, oder nicht.

Diese 3 Schritte sind:

1. Spiegeln des Schemas durch Vertauschen der X- und Y-Koordinaten aller SchemaPunkte
2. Drehen des Schemas um 180° durch getrenntes Spiegeln der X- und Y-Koordinaten aller SchemaPunkte
3. Drehen des Schemas um 90° durch Vertauschen der X- und Y-Koordinaten aller SchemaPunkte und anschließendes Spiegeln der X-Koordinaten

Durch diese Transformationen können sowohl alle vier möglichen Drehungen (0°, 90°, 180°, 270°) als auch gespiegelte bzw. nicht gespiegelte Variationen erzeugt werden.

Als letzter Schritt berechnet die Funktion `erstelle_falle()` ausgehend von `ist_oben` und `ist_links`, sowie der als Parameter an die Funktion übergebenen Größe des Rätsels, die endgültige Position der Falle im Rätsel und addiert die entsprechenden X- und Y-Werte zu allen SchemaPunkten.

Die Funktion gibt das Schema, `ist_oben`, `ist_links` und die Breite bzw. Höhe des Schemas als ein `tuple` zurück.

2.2.2 Vervollständigen des Arukone-Rätsels

Die Funktion `erstelle_arukone()` erstellt zuerst eine Liste bestehend aus n Listen, welche wiederum n mal die Zahl 0 enthalten. Diese Liste stellt die Kästchen und Ziffern im Rätsel dar. Außerdem wird die Anzahl der Zahlenpaare des zu erstellenden Rätsels als $n/2$, gerundet auf die nächstgrößere Ganzzahl, berechnet.

Daraufhin wird durch `erstelle_falle()` eine zufällige Fallenstruktur erzeugt und in die erstellte Struktur eingefügt.

Zum Schluss müssen genug weitere Zahlenpaare eingefügt werden, sodass insgesamt mindestens $n/2$ Zahlenpaare im Rätsel vorhanden sind. Dafür wird zuerst n durch 2 dividiert und das Ergebnis aufgerundet, um die Anzahl der Zahlenpaare zu erhalten.

Daraufhin wird mithilfe einer Zählschleife von 3 bis zu (inklusive) der Anzahl der Paare gezählt (Start bei 3, da die Fallenstruktur bereits zwei Zahlenpaare enthält). Dabei werden die Spalten des Rätsels von links nach rechts durchlaufen und je ein Zahlenpaar in die entsprechende Spalte eingetragen. Dabei beginnt das durchlaufen der Spalten mit der ersten Spalte, wenn die Falle rechts ist, und mit der 4. Spalte, wenn die Falle links ist.

In jeder Spalte wird zuerst ein zufälliges Kästchen bestimmt, in welches die erste Zahl des Zahlenpaares eingetragen wird. Daraufhin wird ein weiteres zufälliges Kästchen unterhalb des ersten ausgewählt, in welches die zweite Zahl des Paares eingetragen wird. Dabei wird darauf geachtet, dass für die erste Zahl nicht das unterste Kästchen der Spalte ausgewählt wird, sodass stets für die zweite Zahl Platz bleibt.

Jetzt wurde ein vollständiges, gültiges, nicht vom Lösungsprogramm lösbares Arukone-Rätsel der festgelegten Größe mit mindestens $n/2$ Zahlenpaaren erstellt. Dieses wird nun zurückgegeben.

2.3 Eingabe, Ausgabe

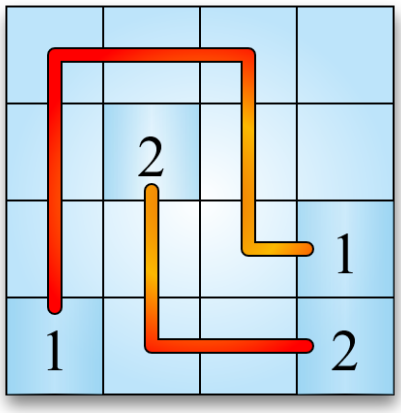
Das Hauptprogramm startet eine Schleife ohne Abbruchbedingung, welche die folgenden Schritte ausführt:

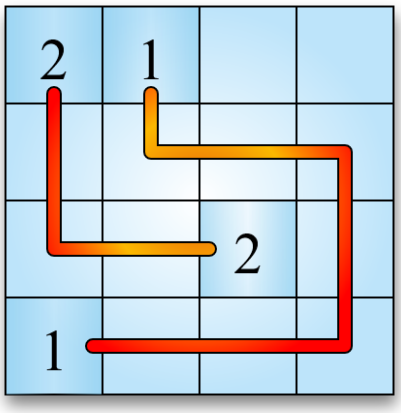
1. Abfragen der Größe des zu erstellenden Rätsels
2. Erstellen des Rätsels mithilfe von `erstelle_arukone()`
3. Umwandlung der Arukone-Datenstruktur und Ausgabe des Rätsels durch die Funktion `arukone_anzeigen()`

Drückt der Benutzer Strg+C, wird das Programm beendet.

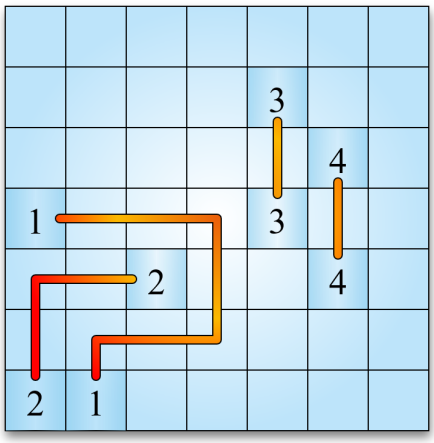
3 Beispiele

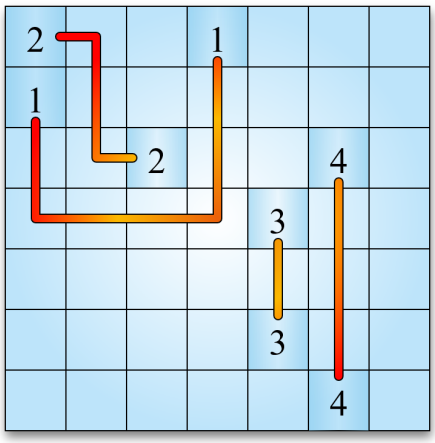
3.1 Bsp. für $n = 4$



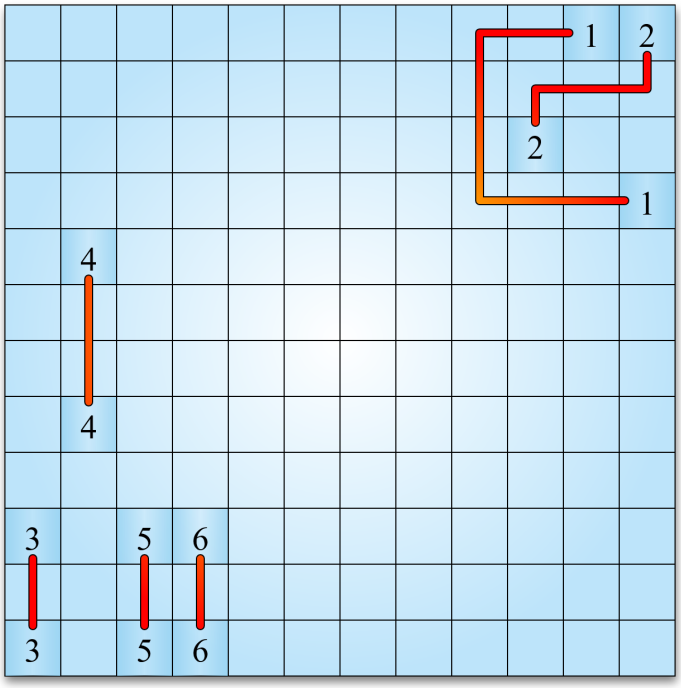


3.2 Bsp. für $n = 7$

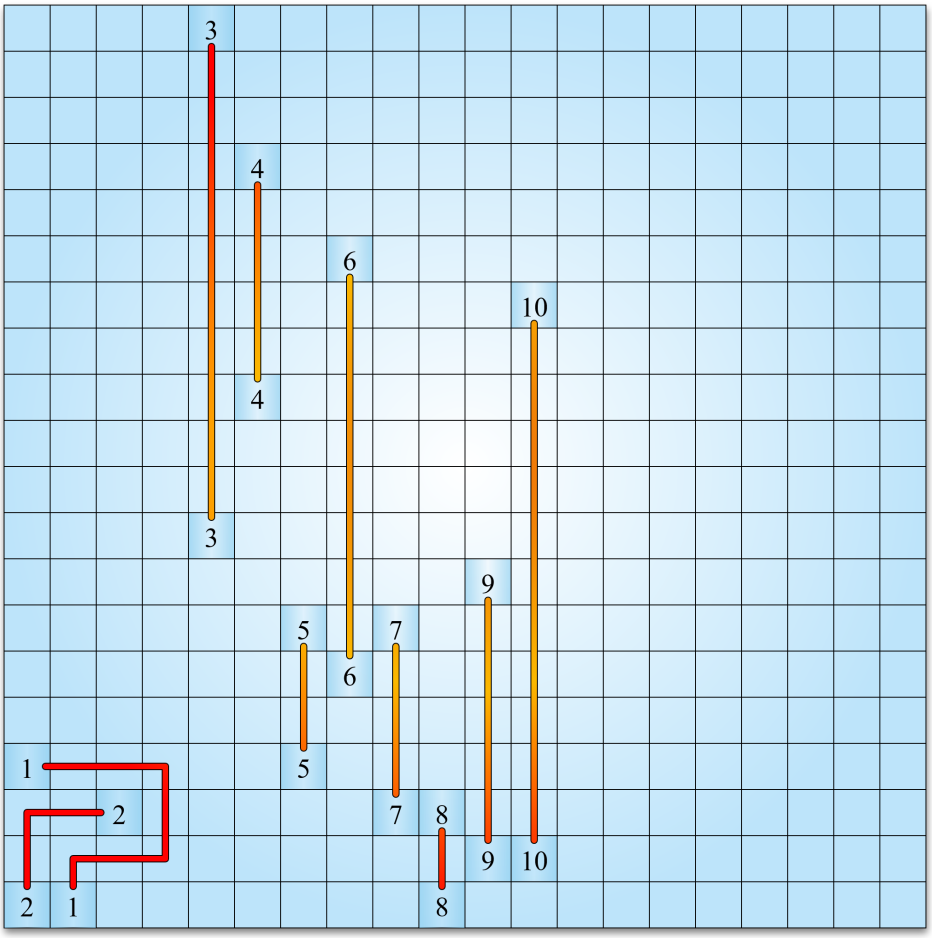




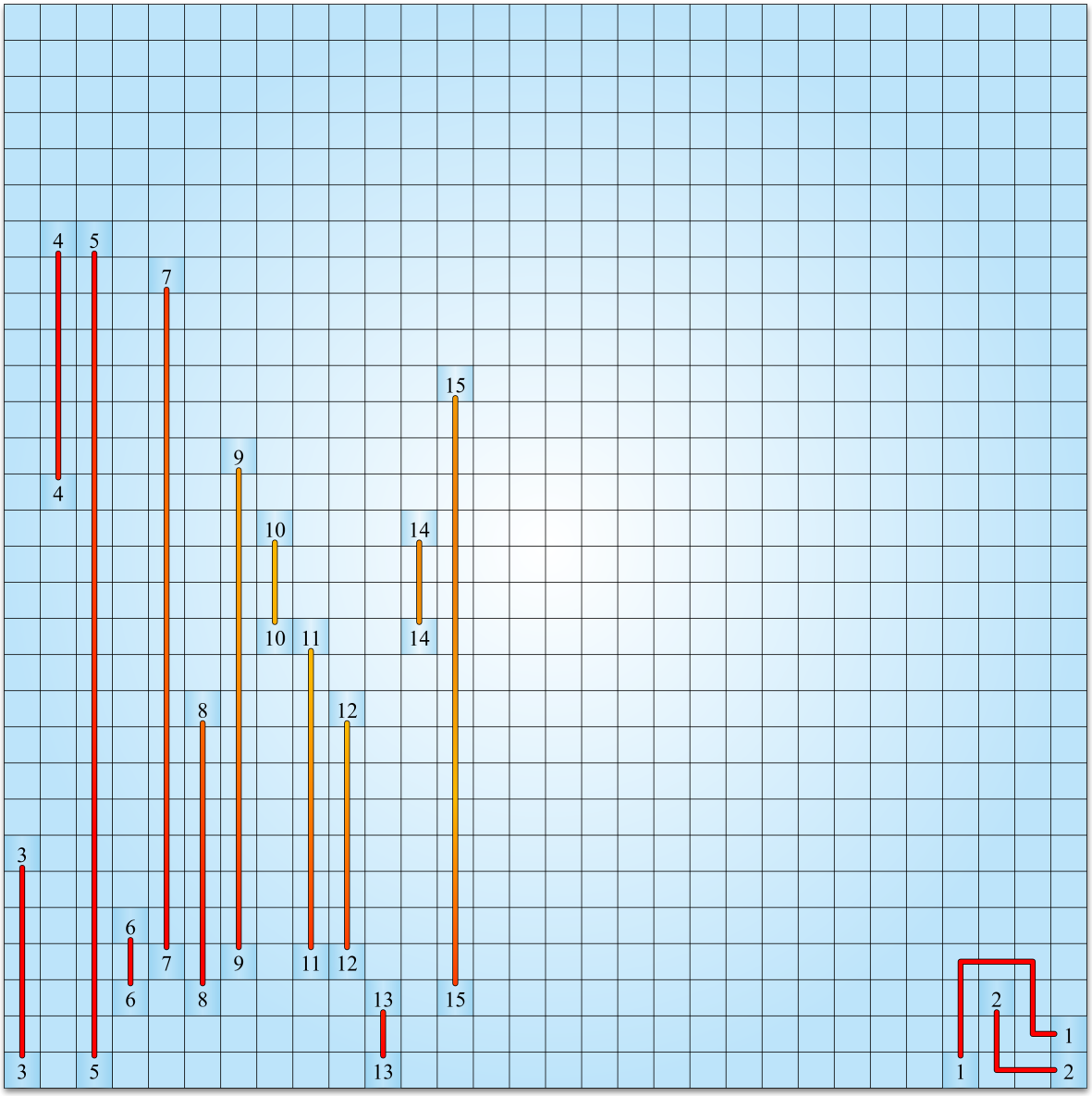
3.3 Bsp. für $n = 12$



3.4 Bsp. für $n = 20$



3.5 Bsp. für $n = 30$



4 Quellcode

```python
import math
import random

# Datenstrukturen

class SchemaPunkt:
    def __init__(self, x, y, wert) -> None:
        self.x = x
        self.y = y
        self.wert = wert

Schema = list[SchemaPunkt]
Arukone = list[list[int]]

# Algorithmus

# Rückgabewert: (breite_hoehe, ist_links, ist_oben, schema)
def erstelle_falle(arukone_groesse: int) -> tuple[int, bool, bool, Schema]:
    schema_breite_hoehe = 4
    schema = [
        SchemaPunkt(0, 0, 1),
        SchemaPunkt(3, 1, 1),
        SchemaPunkt(3, 0, 2),
        SchemaPunkt(1, 2, 2)
    ]

    ist_oben = True
    ist_links = False

    # Spiegeln
    if random.getrandbits(1):
        temp = ist_oben
        ist_oben = ist_links
        ist_links = temp

        for eintrag in schema:
            temp = eintrag.y
            eintrag.y = eintrag.x
            eintrag.x = temp

    # Drehen um 180°
    if random.getrandbits(1):
        ist_oben = not ist_oben
        ist_links = not ist_links

        for eintrag in schema:
            eintrag.x = (schema_breite_hoehe - 1) - eintrag.x
            eintrag.y = (schema_breite_hoehe - 1) - eintrag.y

    # Drehen um 90°
    if random.getrandbits(1):
        temp = ist_oben
        ist_oben = ist_links
        ist_links = not temp

        for eintrag in schema:
            temp = eintrag.y
            eintrag.y = eintrag.x
            eintrag.x = (schema_breite_hoehe - 1) - temp

    # an Position bewegen

    x_position_falle = 0 if ist_links else arukone_groesse - schema_breite_hoehe
    y_position_falle = 0 if ist_oben else arukone_groesse - schema_breite_hoehe

    for eintrag in schema:
        eintrag.x += x_position_falle
        eintrag.y += y_position_falle

    return (schema_breite_hoehe, ist_links, ist_oben, schema)

```

```python
# Annahme: n >= 4
def erstelle_arukone(n: int) -> Arukone:
    arukone = [[0 for _ in range(n)] for _ in range(n)]
    paare = math.ceil(n / 2)

    # Falle übertragen

    falle_groesse, falle_links, falle_oben, falle_schema = erstelle_falle(n)

    for eintrag in falle_schema:
        arukone[eintrag.y][eintrag.x] = eintrag.wert

    # mit restlichen Paaren auffüllen

    for i in range(3, paare + 1):
        y_position_start = random.randint(0, n - 3)
        y_position_ende = random.randint(y_position_start + 2, n - 1)

        x_position = (falle_groesse if falle_links else 0) + i - 3

        arukone[y_position_start][x_position] = i
        arukone[y_position_ende][x_position] = i

    return arukone


# Testen

def arukone_anzeigen(arukone: list[list[int]]):
    n = len(arukone)
    paare = max([max(zeile) for zeile in arukone])
    daten = "\n".join([" ".join([str(feld) for feld in zeile]) for zeile in arukone])

    print(n)
    print(paare)
    print(daten)
    print()


try:
    while True:
        n = int(input("n: "))
        arukone = erstelle_arukone(n)
        arukone_anzeigen(arukone)


except KeyboardInterrupt:
    exit()
```

arukone.py