Aufgabe 3: Konfetti

Teilnahme-ID: 00000

Bearbeiter/-in dieser Aufgabe:
Constantin

14. March 2025

Inhaltsverzeichnis

1. Lösungsidee....2
2. Laufzeitanalyse....3
3. Implementierung....3
    3.1 is_true....3
    3.2 read_file....4
    3.3 valid_lines....4
    3.4 is_valid....4
    3.5 score_table....4
    3.6 find_hotspot....5
    3.7 weighted_index....5
    3.8 random_index....5
    3.9 simulated_annealing....6
    3.10 Funktionsaufruf....7
    Teilaufgabe b)....7
    3.11 is_valid_line....7
    3.12 Funktionsaufruf....7
4. Beispiele....8
    4.1 Teilaufgabe a)....9
        4.1.1 Konfetti00....9
        4.1.2 Konfetti01....9
        4.1.3 Konfetti02....9
        4.1.4 Konfetti03....9
        4.1.5 Konfetti04....9
        4.1.6 Konfetti05....9
        4.1.7 Konfetti06....10
        4.1.8 Konfetti07....10
        4.1.9 Konfetti08....10
        4.1.10 Konfetti09....10
        4.1.11 Konfetti10....10
        4.1.12 Konfetti11....11
        4.1.13 Konfetti12....11
        4.1.14 small....11
    4.2 Teilaufgabe b)....12
        4.2.1 konfetti00....12

4.2.2 konfetti03....12
4.2.3 konfetti09....12
4.2.4 konfetti12....12
4.2.5 konfetti13....12
5. Quellcode....13
5.1 Teilaufgabe b)....15

1. Lösungsidee

Für diese Aufgabe war als erstes eine Implentierung mit Hilfe von Permutationsgeneration angedacht. Wie sich herausstellt hat diese jedoch eine Zeitkomplexität von O(n!). Deswegen wird stattdessen Simulated Annealing verwendet, ein heuristisches Approximationsverfahren, das eine bekannte Lösung für das Eucledian Travelling Salesman Problem ist. Der Algorithmus generiert dabei langsam immer bessere Lösung. Simulated Annealing hebt sich von anderen Approximationsverfahren ab, da es auch schlechtere Lösungen akzeptieren kann. Dies dient dazu lokalen Minimas zu entkommen. Zusätzlich wird ein Kühlungssystem verwendet, welches dafür sorgt, dass am Ende nur noch eindeutig bessere Lösungen akzeptiert werden, während am Anfang möglichst viele verschiedene Optionen in Betracht gezogen werden. Zusätzlich werden in der Lösung weitere Tricks angewendet, die für eine schnellere Lösung sorgen. Zu einem werden alle Spalten zuerst nach der Anzahl an True-Werten sortiert. True bedeutet in diesem Kontext, dass ein Feld entweder ein ‘y’ oder ‘?’ enthält. So werden die Spalten mit vielen True-Werten gebündelt und erhöhen damit sofort die Wahrscheinlichkeit für zwei anliegende True-Werte, also eine valide Zeile. Zusätzlich werden für größere Tabellen Gewichtelisten erstellt. Diese repräsentieren die Wahrscheinlichkeit, dass diese Spalte mit einer anderen getauscht wird. Die Gewichtelisten sind immer um die Spalte mit den meisten True-Werten zentriert, mit absteigenden Werten in beide Richtungen. Für kleinere Tabellen lohnt sich dieser Ansatz nicht und sorgt für eine langsamere Lösung. Sollte eine erste Zeile angegeben sein wird die Gewichteliste stattdessen um den Index der angegeben Zeile orientiert, dass auf beiden seiten gleich viele True-Wert zu finden sind. Dieser Index wird als Hotspot bezeichnet. Somit werden Spalten um diesen Hotspot öftern getauscht und erhöhen die Chance eine valide Lösung zu generieren. Außerdem ist es wichtig zu erwähnen, dass sobald die erste Zeile übereinstimmt nur noch Spalten mit dem gleichen Wert in der ersten Zeile getauscht werden können. Somit muss die erste Zeile nur einmal übereinstimmen und ist für den Rest des Durchlaufs sicher valide.

Anders eine deterministische Lösung die alle n! Permutationen generiert, ist mit einem Approximationsverfahren keine Lösung garantiert, allerdings äußert wahrscheinlich und unfassbar schneller. Allein eine Tabelle mit 100 Spalten hat 9,33e+157 mögliche Permutationen. Eine deterministische Lösung ist daher einfach nicht umsetzbar.

Für den zweiten Teil der Aufgabe wird der Algorithmus minimal erweitert. Durch das Simulated Annealing kann bereits eine “beste” Lösung generiert werden. Sollte diese also nicht valide sein, oder die Eingabe nicht valide sein kann in jeder Zeile ein oder zwei Zeichen strategisch so ersetzt werden, dass eine valide Lösung entsteht. In Zeilen in denen bereits ein True-Wert vorhanden ist, kann einfach ein benachbartes Symbol ersetzt werden um eine valide Lösung zu schaffen. In Zeilen wo kein True-Wert vorhanden ist können zwei beliebige benachbarte Symbole ersetzt werden.

2. Laufzeitanalyse

Teilaufgabe a)

Insgesamt hat die Implementierung eine Laufzeit von O(R × C + T × log T),
wobei **R** die Anzahl der Zeilen, **C** die Anzahl der Spalten und **T** die Anzahl der Spalten (nach Transposition) ist.

- **Best-Case:** O(R × C), falls schnell eine gültige Lösung gefunden wird und nur wenige Spalten neu sortiert werden müssen.
- **Worst-Case:** O(R × C × Anzahl_Iterationen) ≈ O(R × C × Zeit_in_Sekunden), falls keine gute Lösung gefunden wird und viele Neuanordnungen nötig sind.

Teilaufgabe b)

Insgesamt hat die Implementierung eine Laufzeit von O(R × C + T × log T),
wobei **R** die Anzahl der Zeilen, **C** die Anzahl der Spalten und **T** die Anzahl der Spalten (nach Transposition) ist.

- **Best-Case:** O(R × C), falls schnell eine gültige Lösung gefunden wird und wenig Korrekturen am Ende nötig sind.
- **Worst-Case:** O(R × C × Anzahl_Iterationen) + O(R × C), falls nach der Annealing-Phase noch viele Zeilen händisch korrigiert werden müssen.

3. Implementierung

Um das Programm auszuführen reicht im Ordner mit der Quelldatei folgenden Befehl auszuführen.

```
Python aufgabe3.py “<beispiel>.txt”
```

z.B. *python aufgabe1.py “konfetti00.txt”* für Beispiel konfetti00.

Das Programm funktioniert folgendermaßen:

Teilaufgabe a):

1. Die Beispieldatei wird eingelesen (read_file)

2. Es wird überprüft ob jede Zeile mindestens zwei True-Werte enthält, wenn nicht wird die “Not a valid input” Fehlermeldung in die Ausgabe geschrieben (valid_lines)

3. Die Tabelle wird in Spalten übersetzt und die optimale Anordnung wird mit Hilfe von Simulated Annealing gefunden (simulated_annealing)

4. Die Lösung wird zurück in Zeilen übersetzt und es wird ein letztes mal überprüft ob die Tabelle valide ist. Das Ergebnis, die Laufzeit und Tabelle werden in die Ausgabe geschrieben.

Teilaufgabe b):

Für Teilaufgabe b wurde eine Anzahl minimaler Änderungen vorgenommen. Zu einem wurde eine is_valid_line Funktion ergänzt, außerdem bricht die simulated_annealing Funktion nicht ab, wenn eine valide Lösung gefunden wurde um Resourcen zu sparen und gibt stattdessen immer die beste Lösung aus. Zusätzlich wird selbstverständlich danach über jede Zeile iteriert und überprüft ob diese valid ist. Sollte dies nicht der Fall sein wird ein True-Wert mit anliegendem False-Wert identifiziert und der False-Wert wird durch einen True-Wert ersetzt, welches zu einer validen Zeile führt.

3.1 is_true

```
def is_True(string):
    return string == "y" or string == '?'
```

Gibt zurück ob ein Feld ein True Wert ist also entweder ein ‘y’ oder ‘?’.

3.2 read_file

```
def read_file():
    with open(directory+filename, encoding='utf-8') as file:
        content = [line.strip() for line in file.readlines()]
    first_row = content[0].split()
    rows = int(first_row[0])
    columns = int(first_row[1])
    is_line_specified = is_True(first_row[2])

    if is_line_specified:
        first_line = content[-1].split()
        table = [row.split() for row in content[1:-1]]
    else:
        first_line = []
        table = [row.split() for row in content[1:]]
    return rows, columns, is_line_specified, first_line, table
```

Liest die Beispieldatei mit Hilfe von open() ein und gibt die wichtigsten Informationen zurück:

Anzahl an Reihen, Anzahl an Spalten, ob eine erste Zeile spezifiert ist, die erste Zeile, die komplette Tabelle (ohne erste Zeile - wenn eine erste Zeile spezifiert ist).

3.3 valid_lines

```
def valid_lines(table):
    for row in table:
        count = sum(1 for cell in row if is_True(cell))
        if count < 2:
            return False
    return True
```

Überprüft ob jede Zeile mindestens zwei True-Werte hat. Wenn das nicht der Fall ist kann keine eine valide Anordnung gefunden keine. Dafür wird über jedes Feld iteriert und der Wert überprüft.

3.4 is_valid

```
def is_valid(table):
    if is_line_specified:
        if first_line != table[0]:
            return False
    for row in table:
        half = len(row) // 2
        count = sum(is_True(cell) for cell in row)
        if count > half: # If there are more than 50% trues than there have to be 2
adjacent.
            continue
        if not any(is_True(a) and is_True(b) for a, b in zip(row, row[1:])):
            return False
    return True
```

Überprüft ob eine Tabelle valide ist, also ob jede Zeile mindestens zwei zusammenhängende True-Werte hat und wenn nötig eine bestimmte erste Zeile hat.

3.5 score_table

```
def score_table(table):
    total_score = 0
    for row in table:
        valid_cells = sum(is_True(cell) for cell in row)
        row_score = valid_cells * 0.5

        row_score += sum(1 for a, b in zip(row, row[1:]) if is_True(a) and is_True(b))
        total_score += row_score
    return total_score
```

Kalkuliert einen Score für eine Tabelle. Jede valide Zelle erhält 0,5 Punkte und jedes Paar an validen Zellen entspricht 1 Punkt.

3.6 find_hotspot

```
def find_hotspot(lst):
    n = len(lst)
    min_diff = float('inf')
    hotspot_index = -1

    for i in range(n):
        left_count = sum(1 for x in lst[:i] if is_True(x))
        right_count = sum(1 for x in lst[i+1:] if is_True(x))

        diff = abs(left_count - right_count)

        if diff < min_diff:
            min_diff = diff
            hotspot_index = i

    return hotspot_index
```

Findet den Hotspot in einer Liste bzw. Zeile. Hotspot bedeutet, dass in beide Richtungen gleich viele True-Werte aufzufinden sind. Der Hotspot hat damit also die höchste Konzentration an True-Werten um sich herum.

3.7 weighted_index

```
def weighted_index(lst):
    hotspot = find_hotspot(lst)
    n = len(lst)
    weights = [0] * n

    if hotspot == -1:
        return weights

    max_weight = n // 2
    for i in range(n):
        weights[i] = max(0, max_weight - abs(hotspot - i))

    return weights
```

Gibt eine Liste zurück die n Werte hat, die Gewichten entsprechen. Die Gewichte sind um den Hotspot herum ausgelegt so, dass der Hotspot das höchste Gewicht hat und die Gewichte in beide Richtungen langsam abschwachen.

3.8 random_index

```
def random_index(weights):
    total_weight = sum(weights)

    if total_weight == 0:
        return random.randint(0, len(weights) - 1)

    chosen_index = random.choices(range(len(weights)), weights=weights)[0]
    return chosen_index
```

Gibt einen zufälligen Index zurück. Die Wahrscheinlichkeit für einen Index gewählt zu werden basiert auf seinem Gewicht. Indeces mit höhere Gewichten haben eine höhere Chance gewählt zu werden.

3.9 simulated_annealing

```
def simulated_annealing(col_table, max_time=30, initial_temp=100.0, cooling_rate=0.95):
    if is_line_specified:
        column_sums = [sum(1 for cell in col if is_True(cell)) for col in col_table]
        col_table = sorted(
            col_table,
            key=lambda col: (column_sums[col_table.index(col)],
-abs(find_hotspot(first_line) - col_table.index(col))),
            reverse=True
        )
    else:
        col_table = sorted(col_table, key=lambda x: sum(1 for cell in x if
is_True(cell)), reverse=True)
    if is_valid(list(zip(*col_table))):
        return col_table

    current_table = col_table
    current_score = score_table(list(zip(*current_table)))
    best_table = current_table
    best_score = current_score
    temperature = initial_temp
    if is_line_specified:
        weights = weighted_index(first_line)
    else:
        weights = [len(current_table[0])-i+1 for i in range(len(current_table[0]))]

    while time.time() - start_time < max_time:
```

```
        new_table = current_table[:]
        while True:
            if len(new_table) < 500:
                a = random.randint(0, len(new_table) - 1)
                b = random.randint(0, len(new_table) - 1)
            else:
                a = random_index(weights)
                b = random_index(weights)
            if a != b:
                break

        if is_line_specified:
            current_rows = [list(row) for row in zip(*new_table)]
            if first_line == current_rows[0]:
                while True:
                    if new_table[a][0] == new_table[b][0]:
                        break
                    a = random_index(weights)
                    b = random_index(weights)

        new_table[a], new_table[b] = new_table[b], new_table[a]

        new_rows = [list(row) for row in zip(*new_table)]
        new_score = score_table(new_rows)

        delta = new_score - current_score
        if delta > 0 or math.exp(delta / temperature) > random.random():
            current_table = new_table
            current_score = new_score
            if new_score > best_score:
                best_table = new_table
                best_score = new_score

        temperature *= cooling_rate
        if temperature < 1e-3:
            temperature = initial_temp

        if is_valid(new_rows):
            print(new_table)
            return new_table

    return best_table
```

Eine Implementierung eines Simulated Annealing Algorithmus. Es werden langsam bessere Lösungen generiert. Dafür werden immer zwei zufällige Spalten getauscht und dann überprüft ob die entstande Tabelle einen höheren Score hat als die vorherige. Sollte inzwischen eine valide Lösungen gefunden werden, wird der Prozess abgebrochen und diese Lösung zurückgegeben. Ansonsten wird nach MAX_TIME die Lösung mit dem höchsten Score zurückgegeben.

3.10 Funktionsaufruf

```
rows, columns, is_line_specified, first_line, table = read_file()
if not valid_lines(table):
    print("Not a valid input")
    with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
        file.write("Not a valid input!")
else:
    col_table = [list(col) for col in zip(*table)]
    result_table = simulated_annealing(col_table)
    row_table = [list(row) for row in zip(*result_table)]

    end_time = time.time()
```

```
    runtime = end_time - start_time
    print(f"Runtime: {runtime} seconds")

    start_string = "No solution found in"
    if is_valid(row_table):
        start_string = "Solution found in"
        print("Solution found")

    with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
        file.write(f"{start_string} {runtime}s:\n{row_table}")
```

Nutzt alle vorher erklärten Funktionen und ruft sie in der richtigen Reihenfolge auf. Zusätzlich wird die Laufzeit gemessen und zusammen mit der Lösung oder Fehlermeldungen in die Ausgabedatei mit open() geschrieben.

Teilaufgabe b)

3.11 is_valid_line

```
def is_valid_line(row):
    if not any(is_True(a) and is_True(b) for a, b in zip(row, row[1:])):
        return False
    return True
```

Gibt zurück ob eine Zeile valide ist, also mindestens zwei anliegende True-Werte enthält.

3.12 Funktionsaufruf

```
rows, columns, is_line_specified, first_line, table = read_file()
col_table = [list(col) for col in zip(*table)]
result_table = simulated_annealing(col_table)
row_table = [list(row) for row in zip(*result_table)]
if not is_valid(row_table):
    for row in row_table:
        if is_valid_line(row):
            continue
        for i in range(len(row)):
            if not is_True(row[i]):
                if is_True(row[i-1]) and i != 0:
                    row[i] = 'y'
                    break
                elif i != len(row)-1 and is_True(row[i+1]):
                    row[i] = 'y'
                    break
        if not is_valid_line(row):
            row[0], row[1] = 'y', 'y'
end_time = time.time()
runtime = end_time - start_time
print(f"Runtime: {runtime} seconds")
start_string = "No solution found in"
if is_valid(row_table):
    start_string = "Solution found in"
    print("Solution found")
with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
    file.write(f"{start_string} {runtime}s:\n{row_table}")
```

Alle Funktionen werden in der richtigen Reihenfolge aufgerufen. Anders als in Teilaufgabe a) können hier Werte ersetzt werden um einen valide Lösung zu schaffen.

4. Beispiele

Für **Teilaufgabe a)** konnten für alle gewünschten Beispiele, außer konfetti12, eine Lösung generiert werden. Für **Teilaufgabe b)** konnte für alle Beispiele, für die vorher keine Lösung gefunden werden konnte, und konfetti13 eine Lösung gefunden werden. Auch in Teilaufgabe b) konnte keine Lösung für konfetti12 generiert werden. Der Grund dafür ist unbekannt.

Alle Tests wurden auf einem Laptop mit den folgenden Spezifikationen durchgeführt:

OS: Windows 11

CPU: Intel Pentium Silver N600 @1.10GhZ

RAM: 4 GB

GPU: Intel UHD Graphics

Python: 3.9.13

Alle Ausgaben folgen dem folgenden Muster:

Die erste Zeile zeigt an ob eine Lösung gefunden wurde und in welcher Laufzeit. Die zweite Zeile ist die valide / beste Tabelle, welche als genestete Liste der Reihen dargestellt ist.

Da manche Lösungsausgaben extrem lang sind, wird nur ein kleiner Abschnitt abgebildet. Die volle Ausgabe kann im Ordner ./ausgaben gefunden werden.

4.1 Teilaufgabe a)

4.1.1 Konfetti00

Not a valid input!

Da Konfetti00 nicht in jeder Zeile mindestens zwei True-Werte enthält kann keine valide Lösung gefunden werden.

4.1.2 Konfetti01

Solution found in 0.056147098541259766s:

[['y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'n'], ['y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n'], ['n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n'], ['y', 'y', 'n', 'n', 'y', 'y', 'n', 'n', 'n']]

4.1.3 Konfetti02

Solution found in 0.05780601501464844s:

[['n', 'y', 'n', 'n', 'y', 'y', 'n', 'n', 'y'], ['n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n'], ['n', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n'], ['y', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'y'], ['n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y']]

4.1.4 Konfetti03

Not a valid input!

4.1.5 Konfetti04

Solution found in 0.07928824424743652s:

[['n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n'], ['y', 'y', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'n', 'n'], ['n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n'], ['y', 'y', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n'], ['y', 'n', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'n', 'y', 'n', 'y', 'n', 'y', 'y'], ['n', 'n', 'n', 'n', 'y', 'n', 'n', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n'], ['n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y']]

4.1.6 Konfetti05

Solution found in 1.0319616794586182s:

[['y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'y', 'n', 'y', 'y', 'y', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'y', 'y', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'n', 'y', 'n', 'y', 'n', 'y', 'y', 'n', 'n', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'n', 'y', 'n', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'y', 'n', 'y', 'n', 'n', 'y', 'n', 'y', 'n', 'n', 'y', 'n', 'y', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', …], …]

4.1.7 Konfetti06

Solution found in 1.0500516891479492s:

[['n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n'], ['n', 'n', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'y'], ['n', 'n', 'y', 'y', 'y', 'y', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'y'], ['n', 'y', 'n', 'n', 'n', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'n'], ['n', 'y', 'n', 'y', 'n', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n'], ['y', 'y', 'n', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'n'], ['n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'y', 'y', 'n'], ['n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n'], ['n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'y', 'n', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'n']]

4.1.8 Konfetti07

Solution found in 2.654334545135498s:

[['n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n'], ['y', 'y', 'y', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'y', 'y'], ['n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n'], ['y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['n',

'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y'], ['n', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n'], ['y', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'n', 'n', 'y', 'y', 'y'], ['n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'n', 'y', 'y', 'n', 'n', 'n'], ['n', 'y', 'y', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n']]

4.1.9 Konfetti08

Solution found in 0.663691520690918s:

[['n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n'], ['y', 'n', 'y', 'n', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', '?', 'y', 'n'], ['n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'y', 'y', 'y'], ['n', 'n', '?', 'n', 'y', 'n', 'n', 'n', 'y', 'y', 'y', 'n', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'n', 'n'], ['n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n'], ['n', 'y', 'n', 'y', 'y', '?', 'y', 'y', '?', 'y', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['y', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n']]

4.1.10 Konfetti09

Not a valid input!

4.1.11 Konfetti10

Solution found in 0.07183361053466797s:

[['y', 'y', 'n', 'n', 'n', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'n'], ['y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y'], ['n', 'n', 'n', 'y', 'n', 'n', 'n', '?', 'y', 'n', 'y', 'n', 'n'], ['n', 'n', 'y', 'y', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n'], ['n', 'n', 'n', 'n', 'n', '?', '?', 'n', 'y', 'n', 'n', 'y', 'y'], ['y', 'y', 'n', 'y', 'y', '?', 'n', 'y', 'n', 'y', 'y', 'y', 'n'], ['n', 'n', 'n', 'y', 'n', 'n', 'y', 'y', '?', 'n', 'y', 'n', 'n']]

4.1.12 Konfetti11

Solution found in 0.1402726173400879s:

[['n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', '?', 'n', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'y', …], …]

4.1.13 Konfetti12

No solution found in 127.63275861740112s:

[['y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', …], …]

Auch wenn konfetti12 eine ähnliche Größe zu konfetti11 und nur knapp 1.57% größer ist konnte keine Lösung gefunden werden. Zu einem weil eine erste Zeile spezifiziert ist und die möglichen korrekten Lösungen deshalb signifikant verkleinert wird. Zu anderem ist meine Hypothese, dass die Dichte an True-Werten einen Einfluss auf das Ergebnis hat. Eine Tabelle in der 80% der Werte True sind hat eine höhere Wahrscheinlichkeit 2 anliegende True-Werte zu haben als ein Beispiel mit nur 40%.

4.1.14 small

Solution found in 0.07094573974609375s:

[['y', 'n', 'y'], ['?', 'y', 'n']]

4.2 Teilaufgabe b)

4.2.1 konfetti00

Solution found in 10.00094747543335s:

[['y', 'y', 'n', 'n'], ['y', '?', 'y', 'n'], ['n', 'n', 'y', 'y'], ['n', 'y', 'y', 'n']]

4.2.2 konfetti03

Solution found in 10.002346277236938s:

[['y', 'y', 'n', 'n', 'n'], ['y', 'y', 'y', 'y', 'y'], ['y', 'y', 'n', 'n', 'n']]

4.2.3 konfetti09

Solution found in 10.016033411026001s:

[['n', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['y', 'y', 'y', 'y', 'y', 'y', '?', 'y', '?', '?', 'n', 'n', 'n'], ['y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n'], ['n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y'], ['n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'n', 'n', 'n', 'n'], ['?', 'n', 'n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n'], ['n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'n', 'n', 'n', 'n']]

4.2.4 konfetti12

No solution found in 157.06112051010132s:

[['y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', '?',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', '?', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y',
'?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', …], …]

4.2.5 konfetti13

Solution found in 0.10878944396972656s:

[['y', 'n', 'n', 'n', 'n', 'y', 'y', 'y', '?', 'y', 'y', 'y', 'y', 'y', '?', 'y'], ['y', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', '?', 'y', 'y', 'y', 'y', 'y', 'n'], ['n', 'n', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'n', 'y', 'y', 'n', 'y'], ['n', 'n', 'n', 'n', 'n', 'n', 'n', 'y', 'n', 'n', 'n', 'y', 'y', 'n', 'n', 'n'], ['y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', '?', 'y', 'y', '?', 'n', 'y', 'n'], ['y', 'y', 'y', 'n', 'y', 'y', 'y', 'n', 'y', 'y', 'y', 'y', 'y', 'y', 'y', 'y'], ['n', 'y', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'y'], ['n', 'n', '?', 'n', 'n', 'y', 'y', 'n', 'y', 'n', '?', 'y', 'n', 'y', 'n', 'n']]

5. Quellcode

```
def is_True(string):
    return string == "y" or string == '?'

def read_file():
    with open(directory+filename, encoding='utf-8') as file:
        content = [line.strip() for line in file.readlines()]
    first_row = content[0].split()
    rows = int(first_row[0])
    columns = int(first_row[1])
    is_line_specified = is_True(first_row[2])

    if is_line_specified:
        first_line = content[-1].split()
        table = [row.split() for row in content[1:-1]]
    else:
        first_line = []
        table = [row.split() for row in content[1:]]
    return rows, columns, is_line_specified, first_line, table

def valid_lines(table):
    for row in table:
        count = sum(1 for cell in row if is_True(cell))
        if count < 2:
            return False
    return True

def is_valid(table):
    if is_line_specified:
```

```
        if first_line != table[0]:
            return False
    for row in table:
        half = len(row) // 2
        count = sum(is_True(cell) for cell in row)
        if count > half:
            continue
        if not any(is_True(a) and is_True(b) for a, b in zip(row, row[1:])):
            return False
    return True

def score_table(table):
    total_score = 0
    for row in table:
        valid_cells = sum(is_True(cell) for cell in row)
        row_score = valid_cells * 0.5
        row_score += sum(1 for a, b in zip(row, row[1:]) if is_True(a) and is_True(b))
        total_score += row_score
    return total_score

def find_hotspot(lst):
    n = len(lst)
    min_diff = float('inf')
    hotspot_index = -1
    for i in range(n):
        left_count = sum(1 for x in lst[:i] if is_True(x))
        right_count = sum(1 for x in lst[i+1:] if is_True(x))
        diff = abs(left_count - right_count)
        if diff < min_diff:
            min_diff = diff
            hotspot_index = i
    return hotspot_index

def weighted_index(lst):
    hotspot = find_hotspot(lst)
    n = len(lst)
    weights = [0] * n

    if hotspot == -1:
        return weights
    max_weight = n // 2
    for i in range(n):
        weights[i] = max(0, max_weight - abs(hotspot - i))
    return weights

def random_index(weights):
    total_weight = sum(weights)
    if total_weight == 0:
        return random.randint(0, len(weights) - 1)
    chosen_index = random.choices(range(len(weights)), weights=weights)[0]
    return chosen_index

def simulated_annealing(col_table, max_time=30, initial_temp=100.0, cooling_rate=0.95):
    if is_line_specified:
        column_sums = [sum(1 for cell in col if is_True(cell)) for col in col_table]
        col_table = sorted(col_table, key=lambda col: (column_sums[col_table.index(col-
abs(find_hotspot(first_line) – col_table.index(col))), reverse=True)
    else:
        col_table = sorted(col_table, key=lambda x: sum(1 for cell in x if
is_True(cell)), reverse=True)
    if is_valid(list(zip(*col_table))):
        return col_table

    current_table = col_table
    current_score = score_table(list(zip(*current_table)))
```

```
    best_table = current_table
    best_score = current_score
    temperature = initial_temp
    if is_line_specified:
        weights = weighted_index(first_line)
    else:
        weights = [len(current_table[0])-i+1 for i in range(len(current_table[0]))]

    while time.time() - start_time < max_time:
        new_table = current_table[:]
        while True:
            if len(new_table) < 500:
                a = random.randint(0, len(new_table) - 1)
                b = random.randint(0, len(new_table) - 1)
            else:
                a = random_index(weights)
                b = random_index(weights)
            if a != b:
                break
        if is_line_specified:
            current_rows = [list(row) for row in zip(*new_table)]
            if first_line == current_rows[0]:
                while True:
                    if new_table[a][0] == new_table[b][0]:
                        break
                    a = random_index(weights)
                    b = random_index(weights)
        new_table[a], new_table[b] = new_table[b], new_table[a]
        new_rows = [list(row) for row in zip(*new_table)]
        new_score = score_table(new_rows)
        delta = new_score - current_score
        if delta > 0 or math.exp(delta / temperature) > random.random():
            current_table = new_table
            current_score = new_score
            if new_score > best_score:
                best_table = new_table
                best_score = new_score
        temperature *= cooling_rate
        if temperature < 1e-3:
            temperature = initial_temp
        if is_valid(new_rows):
            print(new_table)
            return new_table
    return best_table

rows, columns, is_line_specified, first_line, table = read_file()

if not valid_lines(table):
    print("Not a valid input")
    with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
        file.write("Not a valid input!")
else:
    col_table = [list(col) for col in zip(*table)]
    result_table = simulated_annealing(col_table)
    row_table = [list(row) for row in zip(*result_table)]
    end_time = time.time()
    runtime = end_time - start_time
    print(f"Runtime: {runtime} seconds")
    start_string = "No solution found in"
    if is_valid(row_table):
        start_string = "Solution found in"
        print("Solution found")
    with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
        file.write(f"{start_string} {runtime}s:\n{row_table}")
```

5.1 Teilaufgabe b)

```
def is_valid_line(row):
    """Returns whether a line is valid or not."""
    if not any(is_True(a) and is_True(b) for a, b in zip(row, row[1:])):
        return False
    return True


if not is_valid(row_table):
    for row in row_table:
        if is_valid_line(row):
            continue
        for i in range(len(row)):
            if not is_True(row[i]):
                if is_True(row[i-1]) and i != 0:
                    row[i] = 'y'
                    break
                elif i != len(row)-1 and is_True(row[i+1]):
                    row[i] = 'y'
                    break
        if not is_valid_line(row):
            row[0], row[1] = 'y', 'y'

end_time = time.time()
runtime = end_time - start_time
print(f"Runtime: {runtime} seconds")

start_string = "No solution found in"
if is_valid(row_table):
    start_string = "Solution found in"
    print("Solution found")
```