Aufgabe 4: Tomograph

Team-ID: 00572

Team-Name: Ralli

Bearbeiter/-innen dieser Aufgabe:
David Adam

11. November 2025

Inhaltsverzeichnis

1 Lösungsidee

Wir haben ein $n \times n$-Gitter mit Variablen $x_{ij} \in \{0, 1\}$ (1 steht für "X", 0 für "."). Gegeben sind Summen für Zeilen (r_i), Spalten (c_j) und zwei Diagonalrichtungen ($d_k^{\nearrow}$) und ($d_k^{\searrow}$).

Formal:

$$\sum_{j=0}^{n-1} x_{ij} = r_i \quad \text{für alle } i,$$

$$\sum_{i=0}^{n-1} x_{ij} = c_j \quad \text{für alle } j,$$

$$\sum_{(i,j)\in D_k^{\nearrow}} x_{ij} = d_k^{\nearrow} \quad (k = 0, \ldots, 2n-2),$$

$$\sum_{(i,j)\in D_k^{\searrow}} x_{ij} = d_k^{\searrow} \quad (k = 0, \ldots, 2n-2),$$

wobei die Diagonalindizes über

$$D_k^{\nearrow} = \{(i,j) \mid i - j = k - (n-1)\}, \qquad D_k^{\searrow} = \{(i,j) \mid i + j = k\}$$

definiert sind. Eine Belegung ist zulässig, wenn alle vier Summenfamilien stimmen.

Idee: Backtracking mit harter Früherkennung von Unmöglichkeit. Für jede Zeile/Spalte/Diagonale tracke ich "wie viele Einsen noch übrig sind"(left) und "wie viele Felder dort noch frei sind"(free). Ein Zustand ist sofort unmöglich, wenn für irgendeine Nebenbedingung gilt: left $<$ 0 oder left $>$ free. So schneide ich viele Pfade früh ab.

Ziel laut Aufgabenstellung ist auch, Eindeutigkeiten zu erkennen ("?"bei Mehrdeutigkeit). Dafür müsste man mehrere Lösungen einsammeln. In meiner Abgabe konzentriere ich mich wegen Laufzeit erst auf das Finden einer gültigen Figur; Mehrdeutigkeit kann optional erkannt werden, wenn mehrere Lösungen innerhalb eines Schrittlimits gefunden werden.

2 Umsetzung

Die Summenbedingungen halte ich explizit nach, damit die Suche gezielt bleibt. Über `diag1_index` und `diag2_index` ordne ich jedes Feld den passenden Diagonalen zu; in `rows/cols/d1/d2_left` und `rows/cols/d1/d2_free` stehen jeweils die noch zu setzenden Einsen und die freien Felder. Vor jedem Schritt prüft `check_bounds`, dass `0 <= left <= free` überall eingehalten ist. `propagate` setzt erzwungene Werte (bei `left==0` nur 0, bei `left==free` nur 1). Die nächste Zelle wähle ich mit einer MRV-ähnlichen Heuristik und teste 1 bzw. 0; Änderungen protokolliere ich in `log` und mache sie bei Bedarf mit `undo` rückgängig. Ein `step_limit` begrenzt die Laufzeit, und nach der ersten gültigen Figur beende ich mit `found_one`. Optional sammelt `possibilities` beobachtete Werte pro Feld; wenn innerhalb des Limits mehrere Lösungen auftreten, entstehen damit auch "?Felder.

3 Werkzeuge

extbfPython 3: Als Programmiersprache, weil Backtracking in Python einfach zu implementieren ist und die Sprache gut lesbar ist.

VS Code mit Inline-Vervollständigung: Die automatische Vervollständigung hat mir beim Schreiben der vielen ähnlichen Array-Zugriffe geholfen (z.B. `rows_left`, `cols_left`, usw.).

Backtracking-Muster: Ich habe mir verschiedene Backtracking-Implementierungen angeschaut, zum Beispiel für Sudoku-Solver und das N-Damen-Problem. Die Grundstruktur (rekursive Funktion, die sich selbst aufruft und Zustand zurücksetzt) habe ich daraus übernommen.

Eigene Ideen: Die konkrete Anwendung auf das Tomograph-Problem mit den vier verschiedenen Summenarten und das Tracking der Mehrdeutigkeiten sind meine eigenen Ideen. Besonders die Heuristik für die Feldauswahl habe ich selbst entwickelt, nachdem ich gemerkt habe, dass das naive Durchgehen von links nach rechts zu langsam ist.

Constraint-Propagation: Die Idee, ständig zu prüfen, ob die Constraints noch erfüllbar sind, kommt aus dem Bereich der Constraint-Satisfaction-Probleme (CSP). Ich habe mir ein paar CSP-Beispiele angeschaut und die Idee dann auf mein Problem übertragen.

Vorgehensweise: Zuerst habe ich die Aufgabe analysiert und mir klar gemacht, dass es ein kombinatorisches Suchproblem ist. Dann habe ich eine naive Backtracking-Version geschrieben, die sehr langsam war. Danach habe ich die Constraint-Checks eingebaut und die Heuristik für die Feldauswahl hinzugefügt. Zum Schluss kam das Possibilities-Tracking für die Mehrdeutigkeitserkennung dazu.

4 Beispiele

4.1 tomograph00.txt

```
. . XXX. . .
. . X. X. . .
. . XXX. . .
X. . X. . . .
. XXXXXX.
. . . X. . X.
```

```
. . . X. . . X
. . X. X. . .
```

4.2 tomograph01.txt

```
. .
X.
```

4.3 tomograph02.txt

```
. . X.
XX. .
XXXX
X. . X
```

4.4 tomograph03.txt

```
. ??X
?XX?
?. . ?
. ??X
```

4.5 tomograph04.txt

```
. ???.
?????
??X??
?????
. ???.
```

4.6 tomograph05.txt

```
. ????.
??X???
?XXX??
??XX??
?????.
. ???. X
```

4.7 tomograph06.txt

```
. . XX. X. . .
X. ?. XX. ?X
X?XXXXXX?
X. . X?. X?X
X. . ?XXXX?
X. . XX. . . .
XXX. XXXX.
X?X?. . . XX
. X?X?. XXX
```

4.8 tomograph07.txt

```
. . XX. . X. XX
XXXXXX. . X.
. X. X. . . . . X
XXXX. X. . . X
. X. XX. . XX.
X. X. . XX. XX
```

```
X. . X. . XX. X
X. XX. . . . XX
. . XX. . . X. .
. XX. . . . . X.
```

4.9 tomograph08.txt

```
. XX. . X. X. XX
. . . XXX. X. XX
X. XXXXXXXXX
. . XXXX. X. XX
. . . X. . . X. . X
. X. XX. . . . XX
. XXX. X. X. XX
. . . . . . . . . . .
. . XX. . . X. XX
. X. XXX. X. XX
. . . XXX. X. XX
```

4.10 tomograph09.txt

```
. ????.
??X???
?XXX??
??XX??
?????.
. ???. X
```

4.11 tomograph10.txt

```
XXXXXX......
.XXXXX....XX
...X....XXXX
...XX.XXXX.X
......X.XXXX
.XXXXX.X..X.
XXX.X..X....
XXX.X.X..XX.
XX...XXX....
X.X..XXXXX..
X.......XXXX
...X.XXXX..X
```

5 Quellcode (Auszug)

```
def backtrack():
    nonlocal solutions, example_solution, found_one, steps
    if found_one:
        return
    steps += 1
    if steps > step_limit:
        return
    if not check_bounds():
        return

    base = len(log)
    if not propagate(log):
        undo(log, base)
        return

    if is_complete():
```

```
        if (all(x == 0 for x in rows_left) and
            all(x == 0 for x in cols_left) and
            all(x == 0 for x in d1_left) and
            all(x == 0 for x in d2_left)):
            solutions += 1
            for i in range(n):
                for j in range(n):
                    v = grid[i][j]
                    possibilities[i][j] |= 0b10 if v == 1 else 0b01
            if example_solution is None:
                example_solution = [row[:] for row in grid]
            found_one = True
    else:
        (i, j), dom = choose_cell()
        if dom == 0:
            undo(log, base)
            return
        a0, a1 = allow_vals(i, j)
        for val in ([1] if a1 else []) + ([0] if a0 else []):
            if set_cell(i, j, val, log):
                backtrack()
                undo(log, len(log) - 1)

    undo(log, base)
```