Aufgabe 4: Bibertomograph
ILP Lösung

Team-ID: 00001

Team-Name: Was?

Bearbeiter dieser Aufgabe:
Mathieu de Borman

28. November 2025

Inhaltsverzeichnis

1 Lösungsidee 1
1.1 Definitionen 1
1.2 Nebenbedigungen 2
1.3 Optimierungsziel 2
1.4 Aktualisierung des rekonstruierten Feldes 2
2 Umsetzung 2
2.1 Referenzen 2
2.2 ILP 3
2.3 Korrektheit 3
3 Komplexitätsanalyse 3
4 Werkzeuge 4
5 Beispiele 4
5.1 Standardbeispiele 4
5.2 Zusatzbeispiele 5
6 Quelltext 10

1 Lösungsidee

Das Problem wird mithilfe eines Integer Linear Program (ILP) gelöst. Dies ist in diesem Fall anwendbar, da das Problem aus der Entscheidungsfrage besteht, ob ein Feld markiert sollte oder nicht. Das ILP wird wiederholt verwendet, um möglichst verschiedene Belegungen des Feldes zu finden und dadurch nach und nach alle nicht eindeutigen Felder zu ermitteln.

1.1 Definitionen

Sei $n \in \mathbb{N}$ die Kantenlänge des Feldes.
Sei $F \in (\{0,1\}^n)^n$ ein Tupel, das eine mögliche Fläche darstellt. Dabei steht 1 für ein gefülltes und 0 für ein nicht gefülltes Rasterfeld.
Sei $F' \in (\{0,1,2\}^n)^n$ ein Tupel, das die bisher rekonstruierte Fläche darstellt. Dabei steht 2 für ein nicht eindeutiges, 1 für ein gefülltes und 0 für ein nicht gefülltes Rasterfeld.

Sei $A \in \{0, ..., n\}^n$ das Tupel, das die Summen der Spalten darstellt.
Sei $B \in \{0, ..., n\}^n$ das Tupel, das die Summen der Zeilen darstellt.
Sei $C \in \prod_{i=1}^{2n-1} \{0, ..., n - |n - i|)\}$ das Tupel, dass die Summen der blauen Diagonalen darstellt.
Sei $D \in \prod_{i=1}^{2n-1} \{0, ..., n - |n - i|)\}$ das Tupel, dass die Summen der gelben Diagonalen darstellt.

1.2 Nebenbedigungen

Die Summen der Spalten, Zeilen und Diagonalen müssen zu der Belegung des Felds passen:

$$\forall i : A_i = \sum_{j=1}^{n} F_{ij}$$

$$\forall i : B_i = \sum_{j=1}^{n} F_{ji}$$

$$\forall i : C_i = \sum_{j=1}^{n-|n-i|} F_{(1-j+\min(i,n),i+j-\min(i,n))}$$

$$\forall i : D_i = \sum_{j=1}^{n-|n-i|} F_{(1-j+\min(i,n),n+1-i-j+\min(i,n))}$$

1.3 Optimierungsziel

Im ersten Durchlauf gibt es kein Optimierungsziel. In den darauf folgenden Durchläufen wird der Unterschied (z) zu dem bisher rekonstruierten Feld maximiert:

$$\max z = \sum_{i=1}^{n} \sum_{j=1}^{n} \begin{cases} 0 & ,\text{für } F'_{ij} = 2 \\ -F_{ij} & ,\text{für } F'_{ij} = 1 \\ F_{ij} & ,\text{für } F'_{ij} = 0 \end{cases}$$

1.4 Aktualisierung des rekonstruierten Feldes

Ab der zweiten Verwendung des ILP wird F' entsprechend der neuen Informationen aus F aktualisiert:

$$\forall i, j : F'_{ij} := \begin{cases} F'_{ij} & ,\text{für } F'_{ij} = F_{ij} \\ 2 & ,\text{für } F'_{ij} \neq F_{ij} \end{cases}$$

Falls sich F' in diesem Durchlauf nicht geändert hat, bedeutet das, dass alle nicht eindeutigen Rasterfelder gefunden wurden. Das Ergebnis wird also ausgegeben. Andernfalls könnte es noch nicht eindeutige Felder geben, in dem Fall wird das ILP erneut ausgeführt.

2 Umsetzung

Die Lösung wurde in JavaScript umgesetzt.

2.1 Referenzen

Um im Programm keine vier Fallunterscheidungen für Spalten, Zeilen und Diagonalen vornehmen zu müssen, wird in der Funktion `converter` eine Referenzliste erstellt, in der steht, welche Spalte, Zeile und Diagonale welchen Rasterfeldern entspricht. Die ersten n Einträge referenzieren die Spalten, die nächsten n die Zeilen, und die darauf folgenden $4n - 2$ Einträge referenzieren die Rasterfelder der beiden Diagonalen. Das Feld selbst wird in einer

eindimensionalen Liste gespeichert, wobei das Rasterfeld in der x-ten Spalte und y-ten Zeile die $(x-1)+n(y-1)$-te Position belegt. Dadurch können die Positionen auf dem Feld mit nur einer Zahl dargestellt werden. Beispiel für $n=3$:

0	1	2
3	4	5
6	7	8

Listenindizes der Rasterfelder

```
[
 [0, 3, 6], [1, 4, 7], [2, 5, 8],
 [0, 1, 2], [3, 4, 5], [6, 7, 8],
 [0], [3, 1], [6, 4, 2], [7, 5], [8],
 [6], [3, 7], [0, 4, 8], [1, 5], [2]
]
```

Referenzliste

2.2 ILP

Sobald die Referenzliste initialisiert wurde, wird die Funktion `ILPSolve` mit den zuvor initialisierten Variablen so lange aufgerufen, bis sich die Ergebnisse nicht mehr verändern. Unser ILP-Solver „GLPK“ nimmt (falls bereits eine Lösung gefunden wurde) alle Optimierungsziele (vgl. Lösungsidee) auf, um eine valide Lösung zu erhalten, die zuvor nicht berechnet wurde. Die Nebenbedingungen (die durch die Converter-Funktion generiert wurden) werden ebenfalls aufgenommen um sicherstellen zu können, dass die Lösungen auch gültig sind. Mithilfe von `glpk.solve` wird eine gültigen Lösung generiert, die auf Änderungen bzgl. zuvor berechneten Lösungen überprüft wird, um widersprüchliche Felder als „nicht eindeutig bestimmbar“ zu markieren.

2.3 Korrektheit

Durch das Optimierungsziel wird garantiert, dass für alle Felder bestimmt werden kann, ob ein Feld eindeutig oder nicht eindeutig bestimmbar ist. Betrachten wir beispielsweise eines der von n^2 vorhandenen Felder, wollen wir mit dem Optimierungsziel eine Gegenlösung (in anderen Worten: dieses Feld kann markiert sein und zugleich in anderen Lösungen nicht markiert sein - beides ist gültig) für das Feld erreichen. Es bevorzugt daher Lösungen mit vielen Gegenlösungen über Lösungen mit wenigen/keinen Gegenlösungen. Gibt die Funktion die Lösung unverändert zurück, so liegt es daran, dass bereits alle Belegungen berechnet wurden und keine weitere Gegenlösung für alle bestimmbaren Felder existiert. Deshalb gibt diese Lösung garantiert die richtige Lösung zurück.

3 Komplexitätsanalyse

Wir setzen voraus, dass die Grundrechenarten in konstanter Zeit durchführbar sind und dass Zahlen konstant viel Speicher belegen. Die Laufzeit des ILP liegt in $\mathcal{O}\left(2^{n^2}\right)$, da es aus n^2 Variablen besteht. Da das ILP höchstens einmal pro Änderung eines Rasterfeldes aufgerufen wird, wird es insgesamt höchstens $\mathcal{O}(n^2)$-mal ausgeführt. Damit ergibt sich eine gesamte Laufzeitkomplexität von $\mathcal{O}\left(2^{n^2} \cdot n^2\right)$.

Da das Problem NP-vollständig ist, ist eine wesentlich schnellere Lösung nicht möglich, außer P = NP.[1]

[1] R. Gardner, P. Gritzmann, und D. Prangenberg, „On the computational complexity of reconstructing lattice sets from their X-rays“, Discrete Mathematics, Bd. 202, Nr. 1, S. 45–71, 1999, doi: https://doi.org/10.1016/S0012-365X(98)00347-1.

4 Werkzeuge

In dieser Arbeit wurden keine KIs oder LLMs verwendet. Es wurden GLPK (vgl. https://www.gnu.org/software/glpk/) (GNU GPL License) als ILP-Solver sowie eine modifizierte pako 2.0.4 (vgl. https://github.com/jvail/glpk.js/blob/master/dist/index.js) verwendet.

5 Beispiele

Wir rufen nun das JavaScript-Programm mit den verschiedenen BwInf-Eingabedateien auf. Das Programm wird mithilfe des Browsers ausgeführt. Über den Knopf „Datei hochladen“ können Beispieleingaben ausgewählt werden. Die Ausgabe erscheint auf der Webseite. Im Rasterfeld steht ein Punkt für ein nicht gefülltes, ein X für ein gefülltes Feld und ein Fragezeichen für ein nicht eindeutiges Rasterfeld. Das Programm terminiert für alle BwInf-Beispieleingabedateien in weniger als 500 ms auf einem gewöhnlichen PC. Alle Ergebnisse sind auch im Ordner der Programmdatei zu finden.

5.1 Standardbeispiele

tomograph00

```
Runtime: 0.121s
..XXX...
..X.X...
..XXX...
X..X....
.XXXXXX.
...X..X.
...X...X
..X.X...
```

tomograph01

```
Runtime: 0.11s
..
X.
```

tomograph02

```
Runtime: 0.127s
..X.
XX..
XXXX
X..X
```

tomograph03

```
Runtime: 0.215s
.??X
?XX?
?..?
.??X
```

tomograph04

```
Runtime: 0.252s
.???.
?????
??X??
?????
.???.
```

tomograph05

```
Runtime: 0.229s
.????.
??X???
?XXX??
??XX??
?????.
.???.X
```

tomograph06

```
Runtime: 0.35s
...???.?.
X.???.XX?
X?XX???XX
X?X??X?.?
????X????
?..???.?X
X.?XX?XX?
X.?.??X?X
XXX??X?X.
```

tomograph07

```
Runtime: 0.178s
..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.
```

tomograph08

```
Runtime: 0.103s
.XX..X.X.XX
...XXX.X.XX
X.XXXXXXXX
..XXXX.X.XX
...X...X..X
.X.XX....XX
.XXX.X.X.XX
..........
..XX...X.XX
.X.XXX.X.XX
...XXX.X.XX
```

tomograph09

```
Runtime: 0.128s
X.X....XXX..
X..X.X.XXXXX
X.....X..XXX
XX...X..XXXX
........X..X
X....XX.XXXX
..X.XXXXX.X.
XX..XX.XXX.X
X...X..XXX.X
..X.....X.X.
.X.X.X..XX.X
X..XX..XXXX.
```

tomograph10

```
Runtime: 0.334s
.??????????X
???????????
???????????
???????????
???????????
???????????
???????????
???????????
???????????
???????????
X??????????.
```

5.2 Zusatzbeispiele

Um die Leistungsfähigkeit unseres Programms darzustellen, haben wir uns für weitere Beispiele entschieden, die die Leistungsfähigkeit, Korrektheit sowie Kapazitätsfähigkeit des Programms verdeutlichen sollen. Alle zusätzlichen Eingabedateien sind auch im Ordner der Programmdatei zu finden.

bwinf

Das BWINF-Logo ist eindeutig. Es zeigt, dass sich realistische Bilder tatsächlich damit lösen lassen.

```
Runtime: 0.08s
...............XX.....
.............XXX......
............XXXXX......
..........XXXXXX.......
.........XXXXXXX.......
.........XXX...........
........XXX............
.......XXX.............
.....XXXX....XXXXX.....
....XXXX.....XXXXXX....
...XXXX.......XXXX.....
..XXXX.................
.XXXXXXXXXXXXX........
.XXXXXXXXXXXXX........
........XXXXX..........
.......XXXXXX..........
......XXXXXX...........
.....XXXXXX............
....XXXXXX.............
....XXXXXX.............
...XXXXXX..............
...XXXXXX..............
...XXXXX...............
```

OSU!

Aufgrund der runden Eigenschaft des OSU!-Logos besitzt es deutlich mehr Bereiche mit vielen gefüllten Feldern. Es besteht kein Zweifel, dass es problemlos gelöst wurde.

```
Runtime: 0.316s
........................................
........................................
............XXXXXXXXXXXX..............
.........XXXXXXXXXXXXXXXXXXX..........
.......XXXXXXXXXXXXXXXXXXXXXXX........
......XXXXXXXXXXXXXXXXXXXXXXXXX.......
.....XXXXXXXXXXXXXXXXXXXXXXXXXXXX.....
.....XXXXXXXXXXXXXXXXXXXXXXXXXX...XX.....
....XXXXXXXXXXXXXXXXXXXXXXXXXXX...XXX....
....XXXX......XX.....X...XX...X...XXXX...
...XXXX..XXX..XX..XXXXX..XX...X...XXXX...
...XXXX..XXX..XXXX...XX..XX...XXXXXXXX...
....XXX.......X......XX.......X...XXXX...
....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX....
.....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.....
......XXXXXXXXXXXXXXXXXXXXXXXXXXXX.....
.......XXXXXXXXXXXXXXXXXXXXXXXXXX.......
........XXXXXXXXXXXXXXXXXXXXXXXX........
.........XXXXXXXXXXXXXXXXXXXX.........
............XXXXXXXXXXXXX.............
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
........................................
```

R6Siege

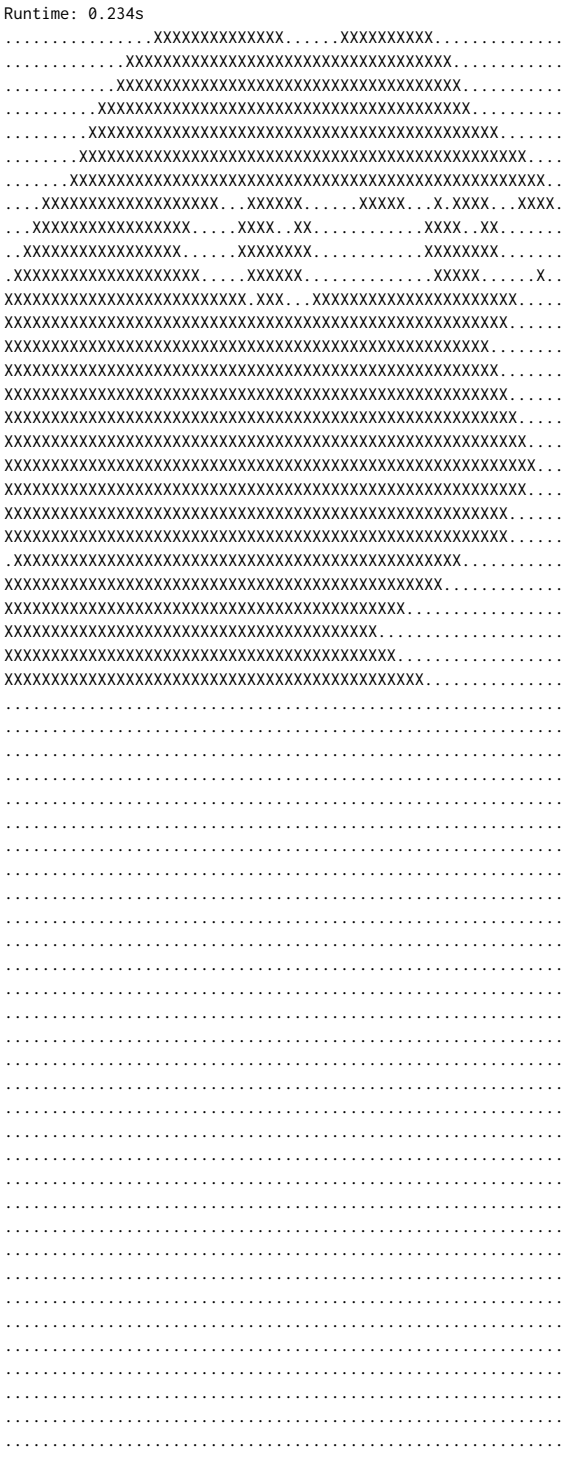
Das R6-Siege-Logo konnte ebenfalls dargestellt werden. Tatsächlich wurden auch einige nicht eindeutig zuweisbare Bereiche entdeckt und markiert.

```
Runtime: 0.928s
.................XXXXXXXXXXXXXX.................
..........XXXXXXXXXXXXXXXXXX??XXXXXX?..........
.......XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......
.....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.....
....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
...XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
...XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXXX??....XXXXXXXXXXXXXXXXXX?.
..XXXXXXXXXXXXXXXXX.......XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXX.......XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXX.......XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXX.......XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXX.......................
..XXXXXXXXXXXXXXXXX.......................
..XXXXXXXXXXXXXXXXXX?XXXXXXXXXXXXXXXX......?.
..XXXXXXXXXXXXXXXXXX?XXXXXXXXXXXXXXXXXXX...?.
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXXXXXX..XXXXXXXXXX..XXXXXXXXXXX.
..XXXXXXXXXXXXXXX.......XXXX...?....XX?XXXXXXX.
..XXXXXXXXXXXXXX.............?.....XX?XXXXXXX.
..XXXXXXXXXXXXXX.................XXXXXXXXXX.
..XXXXXXXXXXXXXX.................XXXXXXXXXX.
..XXXXXXXXXXXXXX..........XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX......XXXX.XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXX.XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXX.XXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.........XXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX......XXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXXXXXXX.
..XXXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXXXXXXX..
...XXXXXXXXXXXXX.....XXXXXXXXXXXXXXXXXXXXXX..
...XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX....
.....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.....
.......XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX........
.........XXXXXXXXXXXXXXXXXXXXXXXXXX...........
...................XXXXXXXXX...................
```

AMOGUS

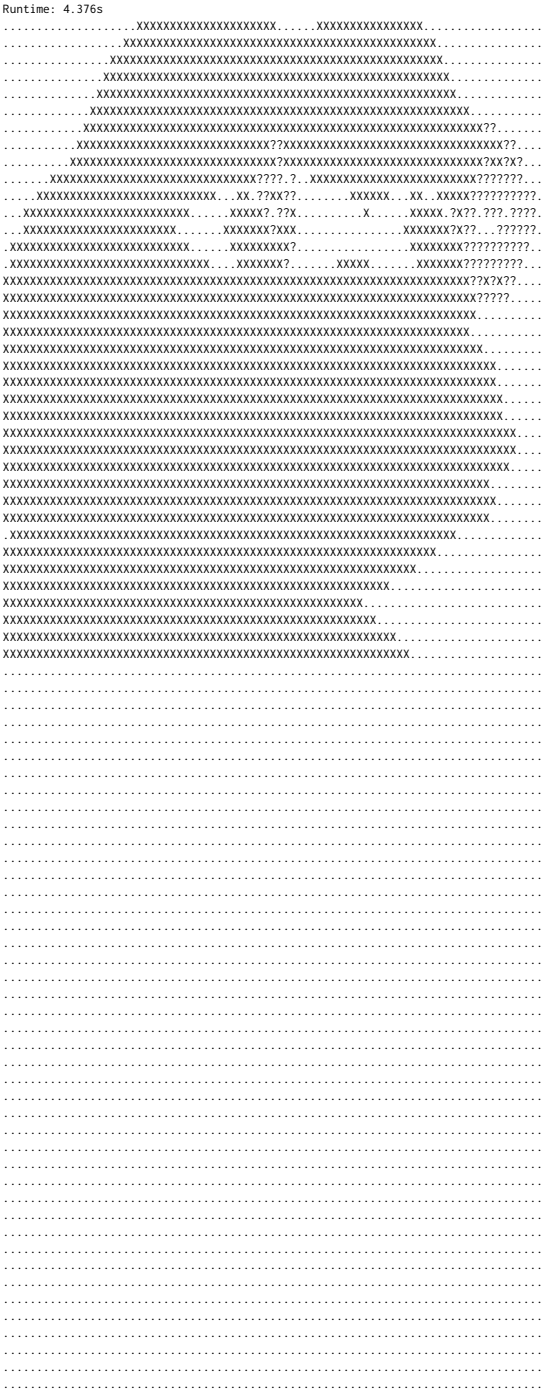
```
Runtime: 0.369s
................XXXXXXXXXXXXXXXXXX.............
..............XXXXXXXXXXXXXXXXXXXXXX...........
............XXXXXXXXXXXXXXXXXXXXXXXXX.........
..........XXXXXXXXXXXXXXXXXXXXXXXXXXXXX.......
.........XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......
.........?XXXXXXXXXXXXXXXXX?XXXXXXXXXXXXXXX....
.........XXXXXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXX...
.........XXXXXXXXXXXXXXXX.................XXXXX..
.........XXXXXXXXXXXXXXXX.................XXXXX.
...XXX?XXXXXXXXXXXXXXXXXX......?...........XXXXX
.XXXXXXXXXXXXXXXXXXXXXXXXXX.................XXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......XXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX..
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
..XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
....XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX....
......?..XXXXXXXXXXXXXXXXXX..XXX?XXXXXXXXXXX....
.........XXXXXXXXXXXXXXXXXX..XXXXXXXXXXXXXXX....
.........XXXXXXXXXXXXXXXXXX..XXXXXXXXXXXXXXX....
.........XXXXXXXXXXXXXXXXXX..XXXXXXXXXXXXXXX.....
.........?XXXXXXXXXXXXXXXX..?.XXXXXXXXXXX.......
...........XXXXXXXXXXXXXXX.....................
..............XXXXXXXXXX.......................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
```

pepe



PEPE2

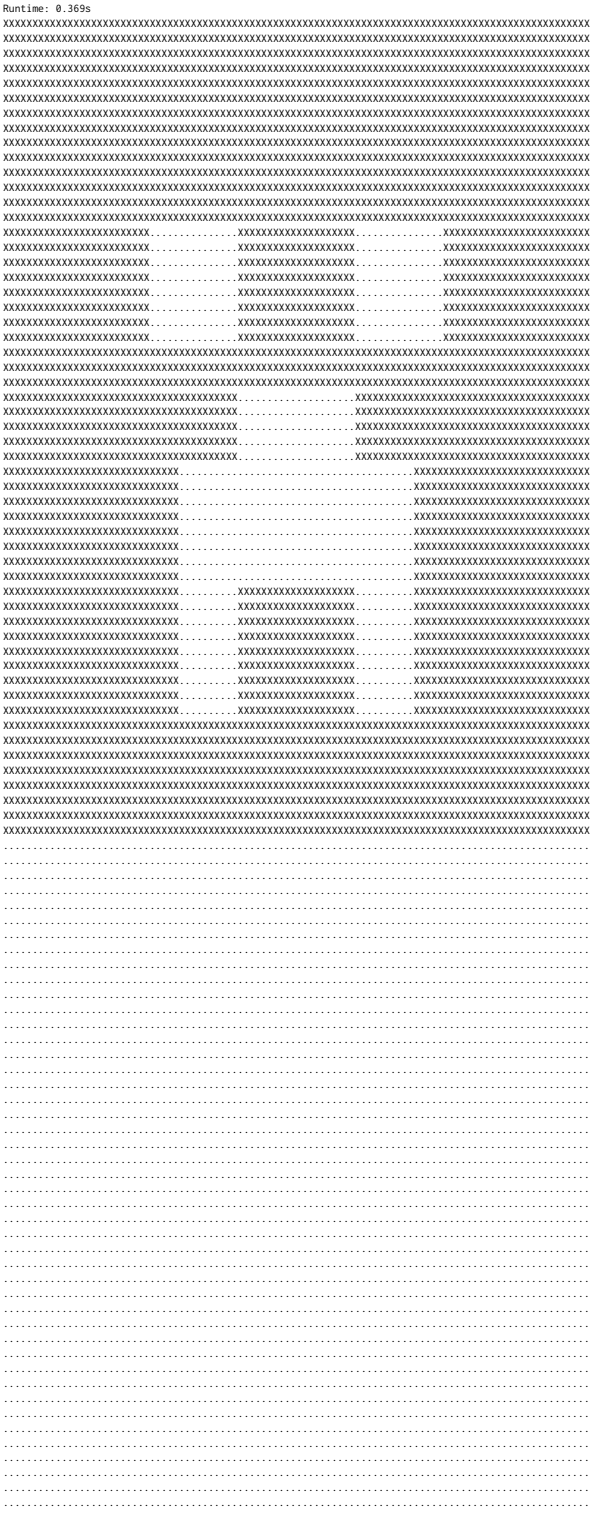
PEPE2 ist eine vergrößerte Variante von pepe. Trotz der Größe (81×81) konnte es innerhalb einer erwartbaren Zeit berechnet werden, und gleichzeitig wurden Bereiche bestimmt, die nicht eindeutig zuweisbar sind. Es ist daher leistungsstark für große Bilder, die einige nicht eindeutig zuweisbare Bereiche enthalten.



CREEPER

CREEPER soll zeigen, dass das Programm auch für extrem große Bilder funktioniert, die kaum oder keine nicht eindeutig zuweisbaren Bereiche enthalten. Das Bild ist insgesamt 100×100 groß.

```
Runtime: 0.369s
```



USBistan

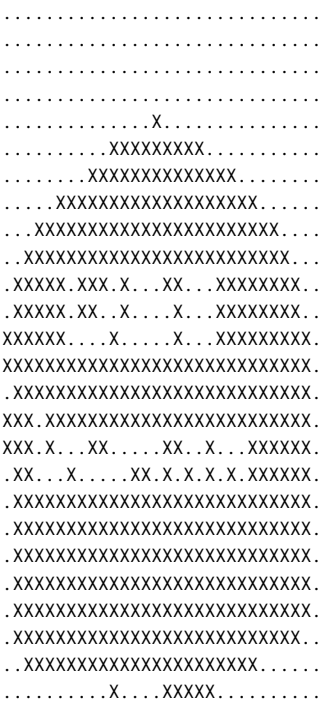
Auch bei Problemen, die sehr viele nicht eindeutig zuweisbare Bereiche enthalten, können alle bestimmt werden.

```
Runtime: 29.884s
```

Eingabe (Referenzbild):

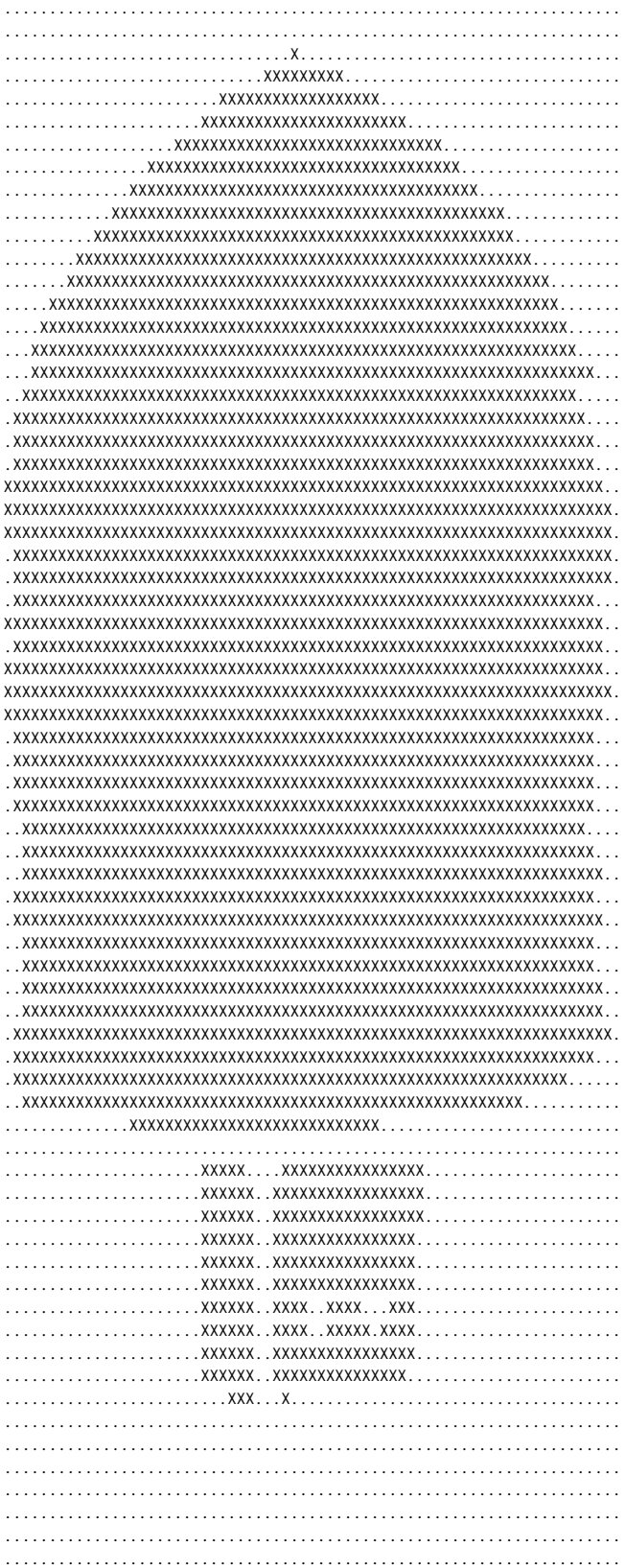
USB-30

Runtime: 0.162s



USB-Stick

Runtime: 1.608s



6 Quelltext

script.js JavaScript

```
27 //Erstelle Referenztabellen, die angeben, welche Spalte, Zeile und Diagonale welchen Positionen im Feld entspricht
28 function converter(n) {
29  let ind = (x, y, i) => [x, y, x + y, x - y + n - 1][i]
30  let sumToField = [[[]], [[]], [[]], [[]]]
31  for (let i = 1; i < n; i++) {
32   sumToField[0].push([])
33   sumToField[1].push([])
34   sumToField[2].push([], [])
35   sumToField[3].push([], [])
36  }
37  for (let x = 0; x < n; x++) {
38   for (let y = 0; y < n; y++) {
39    for (let i = 0; i < 4; i++) {
40     sumToField[i][ind(x, y, i)].push(x + y * n)
41    }
42   }
43  }
44  sumToField = [...sumToField[0], ...sumToField[1], ...sumToField[2], ...sumToField[3]]
45  return sumToField
46 }
47 //Finde eine möglichst unterschiedliche Belegung im Vergleich zum bisher rekonstruierten Feld
48 async function ILPSolve(sumToField, sums, field) {
49  let glpk = await GLPK()
50  let problem = {
51   name: "A4_ILP",
52   objective: { direction: glpk.GLP_MAX, vars: [] },
53   subjectTo: [],
54   bounds: [],
55   binaries: []
56  }
57  //Füge das Optimierungsziel hinzu
58  for (let i = 0; i < field.length; i++) {
59   problem.binaries.push(`F(${i})`)
60   if (field[i] == 0) {
61    problem.objective.vars.push({ name: `F(${i})`, coef: 1 })
62   } else if (field[i] == 1) {
63    problem.objective.vars.push({ name: `F(${i})`, coef: -1 })
64   }
65  }
66  //Füge die Nebenbedingungen hinzu
67  for (let i = 0; i < sumToField.length; i++) {
68   let variables = []
```

```
    problem.subjectTo.push({
     name: `sum${i}`,
     vars: variables,
     bnds: { type: glpk.GLP_FX, lb: sums[i], ub: sums[i] }
    })
    for (let j = 0; j < sumToField[i].length; j++) {
     variables.push({ name: `F(${sumToField[i][j]})`, coef: 1 })
    }
   }
   //Aktualisiere das bisher rekonstruierte Feld
   let result = (await glpk.solve(problem)).result.vars
   let changed = false
   for(let i = 0;i<field.length;i++){
    if(field[i] == -1){
     field[i] = result[`F(${i})`]
     changed = true
    }
    if(field[i] != 2 && field[i] != result[`F(${i})`]){
     field[i] = 2
     changed = true
    }
   }
   return changed
  }
  //Visualisiere das Feld
  function draw(n, field) {
   let symbols = [".", "X", "?"]
   let out = ""
   for (let i = 0; i < field.length; i++) {
    out += symbols[field[i]]
    if (!((i + 1) % n) && n ** 2 > i + 1) {
     out += "\n"
    }
   }
   return out
  }
  //Ermittle eine möglichst passende Figur
  async function main(n, sums) {
   let sumToField = converter(n)
   sums = [...sums[0], ...sums[1], ...sums[2], ...sums[3]]
   let field = Array(n ** 2).fill(-1)
   while (await ILPSolve(sumToField, sums, field)) { }
   return draw(n, field)
  }
```