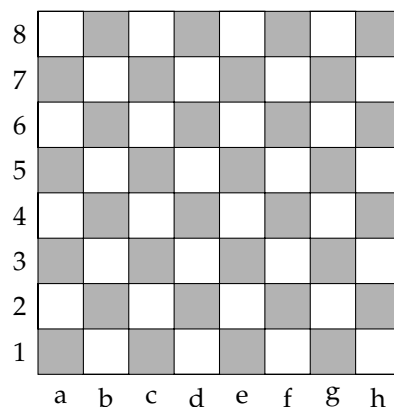


Aufgabe 5 (Backtracking)

Das *Springerproblem* ist ein kombinatorisches Problem, das darin besteht, für einen Springer auf einem leeren Schachbrett eine Route von einem gegebenen Startfeld aus zu finden, auf der dieser jedes Feld des Schachbretts genau einmal besucht.

Ein Schachbrett besteht aus 8×8 Feldern. Ein Springer kann bei einem Zug von einem Ausgangsfeld aus eines von maximal 8 Folgefelder betreten, wie dies in der folgenden Abbildung dargestellt ist. Der Springer darf selbstverständlich nicht über den Rand des Schachbretts hinauspringen.



Eine Lösung des Springerproblems mit Startfeld $h1$ sieht wie folgt aus. Die Felder sind in ihrer Besuchsreihenfolge durchnummeriert. Der Springer bewegt sich also von $h1$ nach $f2$, dann von $f2$ nach $h3$ usw.

41	10	29	26	49	12	31	16
28	25	40	11	30	15	50	13
9	42	27	56	61	48	17	32
24	39	58	47	64	53	14	51
43	8	55	62	57	60	33	18
38	23	46	59	54	63	52	3
7	44	21	36	5	2	19	34
22	37	6	45	20	35	4	1

Formulieren Sie einen rekursiven Algorithmus zur Lösung des Springerproblems von einem vorgegebenen Startfeld aus. Es sollen dabei alle möglichen Lösungen des Springerproblems gefunden werden. Die Lösungen sollen durch Backtracking gefunden werden. Hierbei werden alle möglichen Teilrouten systematisch durchprobiert, und Teilrouten, die nicht zu einer Lösung des Springerproblems führen können, werden nicht weiterverfolgt. Dies ist durch rekursiven Aufruf einer Lösungsfunktion $\text{huepf}(z, y, z)$ zu realisieren, wobei

- x und y die Koordinaten des als nächstes anzuspringenden Feldes sind, und
- z die aktuelle Rekursionstiefe enthält. Wenn die Rekursionstiefe 64 erreicht und das betreffende Feld noch unbesucht ist, ist eine Lösung des Springerproblems gefunden.

Der initiale Aufruf Ihres Algorithmus kann beispielsweise über den Aufruf

`huepf(1, 8, 1)`

erfolgen.

Wählen Sie geeignete Datenstrukturen zur Verwaltung der unbesuchten Felder und zum Speichern gefundener (Teil)Lösungen. Der Algorithmus soll eine gefundene Lösung in der oben angegebenen Form ausdrucken, also als Matrix mit der Besuchsreihenfolge pro Feld.

```
3  /**
4   * Nach <a href=
5   * "https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-
6   * ↪ 1">geeksforgeeks.org</a>
7   */
8   public class Springerproblem {
9       static int felderAnzahl = 8;
10
11       static int lösung[] [];
12
13       static int xBewegungen[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
14       static int yBewegungen[] = { 1, 2, 2, 1, -1, -2, -2, -1 };
15
16       static void druckeLösung() {
17           for (int x = 0; x < felderAnzahl; x++) {
18               for (int y = 0; y < felderAnzahl; y++) {
19                   System.out.print(lösung[x][y] + " ");
20               }
21               System.out.println();
22           }
23       }
24
25       /**
26        * Versuche zum angegebenen Feld zu hüpfen.
27        *
28        * @param x Die x-Koordinate des als nächstes anzuspringenden Feldes.
29        * @param y Die y-Koordinate des als nächstes anzuspringenden Feldes.
30        * @param z Die aktuelle Rekursionstiefe.
31        *
32        * @return Wahr wenn das „angehüpfte“ Feld in der Lösung ist, sonst
33        * ↪ falsch.
34        */
35       static boolean huepf(int x, int y, int z) {
36           // nächste x-Koordinate
37           int xN;
38           // nächste y-Koordinate
39           int yN;
40           if (z == felderAnzahl * felderAnzahl) {
41               return true;
42           }
43
44           for (int i = 0; i < 8; i++) {
45               xN = x + xBewegungen[i];
46               yN = y + yBewegungen[i];
47               if (xN >= 0 && xN < felderAnzahl && yN >= 0 && yN < felderAnzahl &&
48                   ↪ lösung[xN][yN] == -1) {
49                   lösung[xN][yN] = z;
50                   if (huepf(xN, yN, z + 1)) {
51                       return true;
52                   } else {
```

```

50         // backtracking
51         lösung[xN][yN] = -1;
52     }
53 }
54 }
55 return false;
56 }
57
58 static boolean löseSpringerproblem(int x, int y) {
59     lösung = new int[8][8];
60
61     for (int i = 0; i < felderAnzahl; i++) {
62         for (int j = 0; j < felderAnzahl; j++) {
63             lösung[i][j] = -1;
64         }
65     }
66
67     lösung[x][y] = 0;
68
69     if (!huepf(x, y, 1)) {
70         System.out.println("Es konnte keine Lösung gefunden werden.");
71         return false;
72     } else {
73         druckeLösung();
74     }
75
76     return true;
77 }
78
79 public static void main(String args[]) {
80     löseSpringerproblem(0, 0);
81 }
82 }

```

Code-Beispiel auf Github ansehen: [src/main/java/org/bschlangaul/examen/examen_46115/jahr_2018/herbst/Springerproblem.java](https://github.com/org/bschlangaul/examen/examen_46115/jahr_2018/herbst/Springerproblem.java)