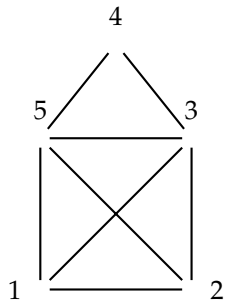
# Das Haus des Nikolaus

Hier ist das *„Haus des Nikolaus“* mit einer bestimmten Nummerierung der Eckpunkte vorgegeben. Es sollen alle Lösungen zum Zeichnen der Figur in einem Zug gefunden werden. Eine Lösung könnte dann in der Form 123451352 ausgegeben werden. Das Programm soll eine einfache Anpassung an andere Graphen ermöglichen. Der Ausschluss von gespiegelten Lösungen ist nicht gefordert.




**Exkurs: Backtracking**
Eine Lösung lässt sich nach dem Prinzip *Versuch und Testen* ermitteln. Eine vermutete Teillösung muss wieder verworfen werden, wenn ein Test ihre Ungültigkeit nachgewiesen hat. Man nennt diesen Ansatz deshalb auch *Rückverfolgen* oder *Backtracking*. Mit diesem Ansatz lassen sich eine ganze Reihe von Problemen in der Informatik sehr elegant formulieren und lösen. Hier eine kleine Auswahl (Genaueres dazu später):

**Acht-Damen-Problem:** Acht Damen sollen so auf ein Schachbrett gestellt werden, dass keine Dame eine andere bedroht.

**Vier-Farben-Problem:** Eine Landkarte soll mit vier Farben so gefärbt werden, dass benachbarte Länder immer unterschiedliche Farben bekommen.

**Labyrinth-Problem:** Ein Labyrinth mit Sackgassen und Verzweigungen ist zu durchlaufen, um den Ausgang zu finden.

Konkreter:

(a) Man versucht, eine Kante (Verbindungsstrecke) zu zeichnen, wenn sie zulässig ist oder noch nicht gezeichnet wurde.

(b) Ist das nicht möglich, muss die zuletzt gezeichnete Kante gelöscht werden.

(c) Ist es möglich, dann hat man das Problem um eine Stufe vereinfacht.

(d) Hat man durch dieses Verfahren insgesamt 8 Kanten zeichnen können, hat man eine Lösung gefunden. Jetzt löscht man wieder die zuletzt gezeichnete Kante und sucht nach weiteren Lösungen.

## Realisierung des Programms

### Datenstrukturen

Die folgende Tabelle gibt an, welche Verbindungslinien zulässig sind (durch X markiert). Die erste Zeile bedeutet also, dass von Punkt 1 zu den Punkten

2, 3 und 5 Strecken gezeichnet werden dürfen. Eine solche Tabelle heißt auch Adjazenzmatrix (von adjazieren; lat.: anwohnen, anliegen). Eine solche Tabelle lässt sich durch `boolean[][] kanteZulaessig;` in einem zweidimensionalen Feld speichern. Eine entsprechende Tabelle `boolean[][] kanteGezeichnet;` erfasst dann die schon gezeichneten Kanten. In einem weiteren eindimensionalen Feld wird jeweils eine Lösung erfasst.

### Methoden

Es bieten sich folgende Methoden zur Strukturierung des Programmes an:

(a) `void initialsiereFelder()`

(b) `void zeichneKante(int von, int nach)`

(c) `void löscheKante(int von, int nach)`

(d) `void gibLösungAus()`

(e) `void versucheKanteZuZeichnen(int start)`: Die rekursive Methode soll vom Punkt start weitere Kanten zeichen.

(f) Das Hauptprogramm:

```
1  public static void main(String[] arg) {
2    initialsiereFelder();
3    for (int punktNr = 1; punktNr <= maxPunktAnzahl; punktNr++) {
4      lösungsWeg[0] = punktNr; // Startpunkt eintragen
5      versucheKanteZuZeichnen(punktNr);
6    }
7    System.out.println();
8    System.out.println("Es ergaben sich " + lösungsAnzahl + " Loesungen.");
9  }
```

```
3  /**
4   * Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen:
5   * Aufgabenblatt 3: Algorithmenmuster.
6   *
7   * <a href="https://www.studon.fau.de/file2521908_download.html">Angabe: AB_3
8   * Greedy_DP_Backtracking.pdf</a>
9   * <a href="https://www.studon.fau.de/file2521907_download.html">Lösung: AB_3
10  * Greedy_DP_Backtracking_Lsg.pdf</a>
11  */
12 public class Nikolaus {
13   static final int maxPunktAnzahl = 5;
14   static final int maxKantenAnzahl = 8;
15   static boolean[][] kanteZulässig;
16   static boolean[][] kanteGezeichnet;
17   static int[] lösungsWeg;
18   static int aktuelleKantenAnzahl = 0;
19   static int lösungsAnzahl = 0;
20
21   /**
22    * Zulässige Kanten für das „Haus des Nikolaus" eintragen. Der Nummerierung
23    * liegt das Bild in main zu Grunde. Eine Anpassung an andere Graphen ist
     ↪ leicht
24    * möglich.
25    */
26   static void initialsiereFelder() {
```

```
27      kanteZulässig = new boolean[maxKantenAnzahl + 1][maxKantenAnzahl + 1];
28      kanteGezeichnet = new boolean[maxKantenAnzahl + 1][maxKantenAnzahl + 1];
29      lösungsWeg = new int[maxKantenAnzahl + 2]; // mit Startpunkt
30      // Erst mal alles auf false ;
31      for (int i = 1; i <= maxPunktAnzahl; i++) {
32        for (int k = 1; k <= maxPunktAnzahl; k++) {
33          kanteZulässig[i][k] = false;
34          kanteGezeichnet[i][k] = false;
35        }
36      }
37
38      kanteZulässig[1][2] = true; // von 1 nach 2 zulässig
39      kanteZulässig[2][1] = true;
40
41      kanteZulässig[1][3] = true;
42      kanteZulässig[3][1] = true;
43
44      kanteZulässig[1][5] = true;
45      kanteZulässig[5][1] = true;
46
47      kanteZulässig[2][3] = true;
48      kanteZulässig[3][2] = true;
49
50      kanteZulässig[2][5] = true;
51      kanteZulässig[5][2] = true;
52
53      kanteZulässig[3][4] = true;
54      kanteZulässig[4][3] = true;
55
56      kanteZulässig[3][5] = true;
57      kanteZulässig[5][3] = true;
58
59      kanteZulässig[4][5] = true;
60      kanteZulässig[5][4] = true;
61      for (int i = 0; i <= maxKantenAnzahl; i++) {
62        lösungsWeg[i] = 0;
63      }
64    }
65
66    static void zeichneKante(final int von, final int nach) {
67      kanteGezeichnet[von][nach] = true;
68      kanteGezeichnet[nach][von] = true;
69      // Anzahl bereits gezeichneter Kanten erhöhen
70      aktuelleKantenAnzahl++;
71      // neuen Wegpunkt in Lösung aufnehmen
72      lösungsWeg[aktuelleKantenAnzahl] = nach;
73    }
74
75    static void löscheKante(final int von, final int nach) {
76      kanteGezeichnet[von][nach] = false;
77      kanteGezeichnet[nach][von] = false;
78      aktuelleKantenAnzahl--;
79    }
80
81    static boolean fertig() {
82      return (aktuelleKantenAnzahl == maxKantenAnzahl);
83    }
84
85    static void gibLösungAus() {
86      for (int i = 0; i <= maxKantenAnzahl; i++) {
87        System.out.print(lösungsWeg[i]);
88        System.out.print(" ");
```

```
89        lösungsAnzahl++;
90        if (lösungsAnzahl % 8 == 0) {
91          System.out.println();
92        }
93      }
94    }
95
96    static void versucheKanteZuZeichnen(final int start) {
97      for (int ziel = 1; ziel <= maxPunktAnzahl; ziel++) {
98        if (kanteZulässig[start][ziel] && !kanteGezeichnet[start][ziel]) {
99          zeichneKante(start, ziel);
100         if (!fertig()) {
101           versucheKanteZuZeichnen(ziel);
102         } else {
103           gibLösungAus();
104         }
105         löscheKante(start, ziel);
106       }
107     }
108   }
109
110   public static void main(final String[] arg) {
111     initialsiereFelder();
112     System.out
113
          ↪  .println("Das Programm bestimmt alle Lösungen des Problems, das Haus des Nikolaus in einem Zu
114     System.out.println("      4      ");
115     System.out.println("     . .     ");
116     System.out.println("    .   .    ");
117     System.out.println("   5-----3   ");
118     System.out.println("   |.   .|   ");
119     System.out.println("   | . . |   ");
120     System.out.println("   | . . |   ");
121     System.out.println("   |.   .|   ");
122     System.out.println("   1-----2   ");
123     for (int punktNr = 1; punktNr <= maxPunktAnzahl; punktNr++) {
124       lösungsWeg[0] = punktNr;
125       versucheKanteZuZeichnen(punktNr);
126     }
127     System.out.println();
128     System.out.println("Es ergaben sich " + lösungsAnzahl + " Lösungen.");
129   }
130 }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/muster/backtracking/Nikolaus.java`