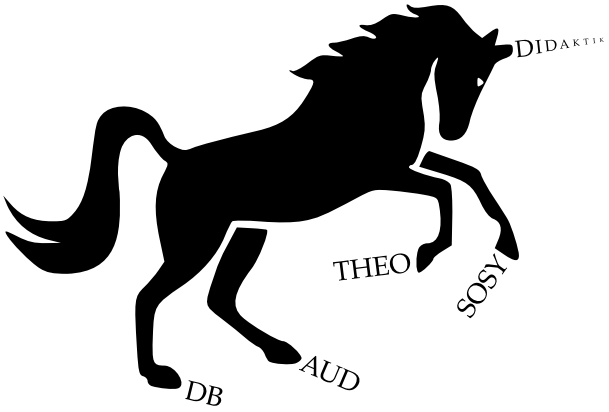
46115
Herbst
2018

Theoretische Informatik / Algorithmen / Datenstrukturen (nicht vertieft)

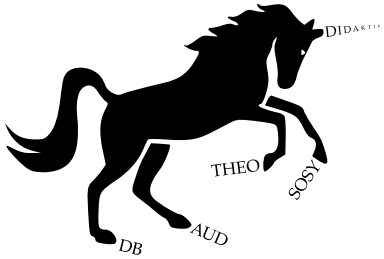
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

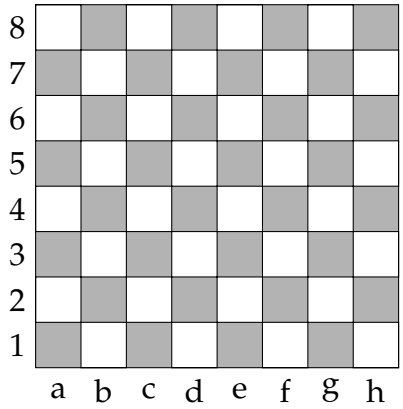
Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 2

Aufgabe 5 (Backtracking) [Springerproblem beim Schach]

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 hinausspringen.



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 `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.

Lösungsvorschlag

```
/**
 * Nach <a href=
 * "https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-
↪  1">geeksforgeeks.org</a>
 */
public class Springerproblem {
  static int felderAnzahl = 8;

  static int lösung[][];

  static int xBewegungen[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
  static int yBewegungen[] = { 1, 2, 2, 1, -1, -2, -2, -1 };

  static void druckeLösung() {
    for (int x = 0; x < felderAnzahl; x++) {
      for (int y = 0; y < felderAnzahl; y++) {
        System.out.print(lösung[x][y] + " ");
      }
      System.out.println();
    }
  }

  /**
   * Versuche zum angegeben Feld zu hüpfen.
   *
   * @param x Die x-Koordinate des als nächstes anzuspringenden Feldes.
   * @param y Die y-Koordinate des als nächstes anzuspringenden Feldes.
   * @param z Die aktuelle Rekursionstiefe.
   *
   * @return Wahr wenn das „angehüpfte" Feld in der Lösung ist, sonst falsch.
   */
  static boolean huepf(int x, int y, int z) {
    // nächste x-Koordinate
    int xN;
    // nächste y-Koordinate
    int yN;
    if (z == felderAnzahl * felderAnzahl) {
      return true;
    }
```

```
    for (int i = 0; i < 8; i++) {
      xN = x + xBewegungen[i];
      yN = y + yBewegungen[i];
      if (xN >= 0 && xN < felderAnzahl && yN >= 0 && yN < felderAnzahl && lösung[xN][yN]
      ↪  == -1) {
        lösung[xN][yN] = z;
        if (huepf(xN, yN, z + 1)) {
          return true;
        } else {
          // backtracking
          lösung[xN][yN] = -1;
        }
      }
    }
    return false;
  }

  static boolean löseSpringerproblem(int x, int y) {
    lösung = new int[8][8];

    for (int i = 0; i < felderAnzahl; i++) {
      for (int j = 0; j < felderAnzahl; j++) {
        lösung[i][j] = -1;
      }
    }

    lösung[x][y] = 0;

    if (!huepf(x, y, 1)) {
      System.out.println("Es konnte keine Lösung gefunden werden.");
      return false;
    } else {
      druckeLösung();
    }

    return true;
  }

  public static void main(String args[]) {
    löseSpringerproblem(0, 0);
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46115/jahr_2018/herbst/Springerproblem.java