rekursives Backtracking

(*Methode „fill()“*)

Stichwörter: Backtracking, Rekursion

rekursives Backtracking

Folgende Methode soll das Feld a (garantiert der Länge $2n$ und beim ersten Aufruf von außen mit 0 initialisiert) mittels rekursivem Backtracking so mit Zahlen $1 \leq x \leq n$ befüllen, dass jedes x genau zweimal in a vorkommt und der Abstand zwischen den Vorkommen genau x ist. Sie soll genau dann `true` zurückgeben, wenn es eine Lösung gibt.

Beispiele:

- `fill(2, [])` → `false`
- `fill(3, [])` → `[3; 1; 2; 1; 3; 2]`
- `fill(4, [])` → `[4; 1; 3; 1; 2; 4; 3; 2]`

```
boolean fill (int n , int[] a) {
  if (n <= 0) {
    return true;
  }
  // TODO
  return false;
}
```

Lösungsvorschlag

```
  public static boolean fill(int n, int[] a) {
    if (n <= 0) {
      return true;
    }
    for (int i = 0; i < a.length - n - 1; i++) {
      // Zwischen i und j müssen genau n andere Zahlen sein
      int j = i + n + 1;
      if (a[i] == 0 && a[j] == 0) {
        a[i] = a[j] = n;
        if (fill(n - 1, a)) {
          return true;
        }
        a[i] = a[j] = 0;
      }
    }
    return false;
  }
```

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

```
fill(0, []):
fill(1, []): false
fill(2, []): false
fill(3, []): 3 1 2 1 3 2
```

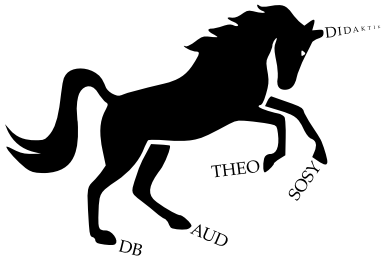
```
fill(4, []): 4 1 3 1 2 4 3 2
fill(5, []): false
fill(6, []): false
fill(7, []): 7 3 6 2 5 3 2 4 7 6 5 1 4 1
fill(8, []): 8 3 7 2 6 3 2 4 5 8 7 6 4 1 5 1
fill(9, []): false
fill(10, []): false
fill(11, []): 11 6 10 2 9 3 2 8 6 3 7 5 11 10 9 4 8 5 7 1 4 1
```

Kompletter Code

```
public class RekursivesBacktracking {

  public static boolean fill(int n, int[] a) {
    if (n <= 0) {
      return true;
    }
    for (int i = 0; i < a.length - n - 1; i++) {
      // Zwischen i und j müssen genau n andere Zahlen sein
      int j = i + n + 1;
      if (a[i] == 0 && a[j] == 0) {
        a[i] = a[j] = n;
        if (fill(n - 1, a)) {
          return true;
        }
        a[i] = a[j] = 0;
      }
    }
    return false;
  }

  public static void executeFill(int n) {
    int[] a = new int[n * 2];
    boolean result = fill(n, a);
    System.out.print("fill(" + n + ", []): ");
    if (result) {
      for (int i = 0; i < a.length; i++) {
        System.out.print(a[i] + " ");
      }
    } else {
      System.out.print("false");
    }

    System.out.println();
  }

  public static void main(String[] args) {
    executeFill(0);
    executeFill(1);
    executeFill(2);
    executeFill(3);
    executeFill(4);
    executeFill(5);
    executeFill(6);
```

```
    executeFill(7);
    executeFill(8);
    executeFill(9);
    executeFill(10);
    executeFill(11);
  }
}
```

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



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.

Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.Der TEX-Quelltext dieses Dokuments kann unter folgender URL aufgerufen werden: `https://github.com/bschlangaul-sammlung/examens-aufgaben/blob/main/Module/30_AUD/60_Algorithmenmuster/50_Backtracking/Aufgabe_Methode-fill.tex`