BacktrackingRucksackProblem: Implementationsdiagramm

Das Rucksackproblem lässt anhand des folgenden Beispiels beschreiben:

- In einen Rucksack kann man 200kg laden (=ganz schön schwer...).
- Es gibt die folgenden Gewichte: 28, 57, 33, 18, 99, 42, 17, 52
- **Welche Gewichte muss man in den Rucksack laden, damit die 200kg möglichst gut ausgenutzt, aber nicht überschritten werden?**

Allgemeiner Backtracking-Algorithmus:

```
Teillösung ist zu Anfang leer.
Stufe ist 0.

Funktion FindeLoesung (Stufe, Teillösung)
 1. Abbruchbedingung: Wenn die Stufe zu groß ist
                       oder es keinen Teil-Lösungsschritt mehr gibt.
 2. Abbruchbedingung: Wenn eine Lösung erreicht wurde.
                      Bearbeite ggf. die Lösung!
 3. wiederhole, solange es noch neue Teil-Lösungsschritte gibt:
    a) Wähle einen neuen Teil-Lösungsschritt.
    b) Erweitere Teillösung um Wahl.
    c) rekursiver Aufruf: FindeLoesung(Stufe+1, Teillösung)
    d) Mache die Erweiterung der Teillösung rückgängig
```

Implementationsdiagramm:

BacktrackingRucksackProblem
- gewichte: int[] - maxgewicht: int // so viel darf maximal in den Rucksack rein - dabei: boolean[] // hier steht für jedes Gewicht, ob es aktuell im Rucksack ist. - erreichtesGewicht: int // das aktuelle Gewicht des Rucksacks mit allen Gewichten aus `dabei` - besteLoesung: boolean[] // die beste bisher gefundene Lösung - bestesGewicht: int // das Gewicht der besten Lösung

+ sucheBesteLoesung() // die Rahmenmethode
+ sucheBesteLoesung(pStufe: int) // die rekursive Methode – zu implementieren!
+ ausgeben(b: boolean[]) // gibt die Gewichte aus, die in `b` auf `true` gesetzt sind.
+ dabeiArrayAusgeben() // gibt das Array dabei aus: `true` ist +, `false` ist -
+ kopiereInBesteLoesung() // kopiert die Werte aus `dabei` in `besteLoesung`
+ berechneGewicht(p: boolean[]) // berechnet das Gesamtgewicht von allen Gewichten,
// für die in `p` ein `true` eingetragen ist.