W08P02 - Potenzmenge

PGdP-TUT Fr-08 Shiqi Li

powerSet(LinkedList<Integer>)

Basisfall

- subsets als Ergebnis

Element der Potenzmenge : Menge

- Basisfall: leere Menge

Potenzmenge von leere Menge : [ ]

```
LinkedList<LinkedList<Integer>> subsets
        = new LinkedList<>();
if (inputSet.isEmpty()) {
    subsets.add(new LinkedList<>());
    return subsets;
}
```

Letzte rekursive Aufruf: [3]

- Ergebnis : [[3], [ ]]
- Verhalten:

1. Elemente selbst als Einzelelement => [3]
2. Verbleibenden Elemente sind Ergebnisse von rekursive Aufruf =>
   [3] ∪ Ergebnisse von Function (bleibende Menge) mit Element 3

1, Einzelelement

Teilen Einzelelement und bekommen Potenzmenge von verbleibenden Elemente

```
LinkedList<Integer> inputSetCopy = new LinkedList<>(inputSet);
int element = inputSetCopy.get(0);
inputSetCopy.remove( index: 0);
LinkedList<LinkedList<Integer>> subsetsFromRest = powerSet(inputSetCopy);
```

2, Vereinigung

- Elemente von Potenzmenge der verbleibender Menge als Element von Ergebnis => [ ] gehört zu subsets

- Elemente von Potenzmenge der verbleibender Menge addieren den Inhalt des Einzelelements => [3] ∪ [ ] = [3] gehört zu subsets

```
for (LinkedList<Integer> subset : subsetsFromRest) {
    subsets.add(subset);
    LinkedList<Integer> withX = new LinkedList<>(subset);
    withX.add(element);
    //add the subset with the current element
    subsets.add(withX);
}
```

findWithSum(LinkedList<Integer>, int)

Basisfall

- sum < 0

Return null

- Leere Menge

1. sum == 0 => return diese Menge
2. sum != 0 => return null

```
if (sum < 0) {
    return null;
}
if (inputSet.isEmpty()) {
    if (sum == 0) {
        return inputSet;
    }
    return null;
}
```

Letzte rekursive Aufruf: [3]

- Ergebnis : Beispiel: sum = 2 => null. sum = 3 => 3
- Verhalten:

1. Entscheiden die Menge ohne ersten Element (3) => Basisfall
2. Analysieren Ergebnis vom Basisfall.
   => null: Falls nicht, probieren wir es ohne x ;
   not null: return diese Menge

1, Teilen (3)

Teilen Einzelelement und vergleichen bleibenden Elemente

=> Basisfall mit sum = sum - 3

```
//make a copy of the input list and separate the first element
LinkedList<Integer> rest = new LinkedList<>(inputSet);
int head = rest.get(0);
rest.remove( index: 0);
//try to include the head from the subset recursively
LinkedList<Integer> withHead = findWithSum(rest,  sum: sum - head);
```

2, Vergleichen

- Prüfen, ob das Einbeziehen erfolgreich ist

1. withHead nicht null
   => return head(3) ∪ verbleibende Menge ([3] ∪ [ ] = [3])

2. withHead null
   => rekursive Function mit Menge ohne head aufrufen

```
if (withHead != null) {
    withHead.add(head);
    return withHead;
}
return findWithSum(rest, sum);
```