AuD Miniklausur (WS16) Aufgabe 2: Rekursion

Philip K.

January 8, 2019

Aufgabenstellung

Die Potenzmenge $\mathcal{P}(n)$ sei die Menge aller Teilmengen der Zahlen von 1 bis n (jeweils einschließlich), wobei die leere Menge $\emptyset$ auch zu den Teilmengen gehört, z. B.

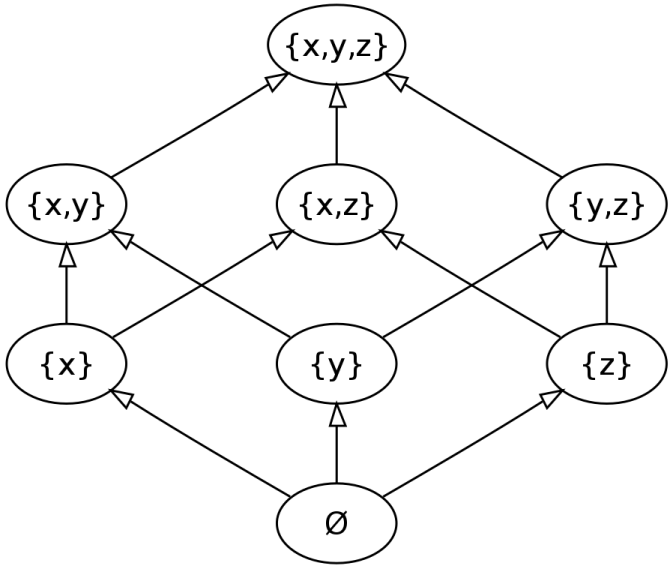
$$\mathcal{P}(3) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

Ergänzen Sie die Methode `potenzmenge`, die rekursiv $\mathcal{P}(n)$ bestimmen soll

Hinweis zu `ArrayList<E>`

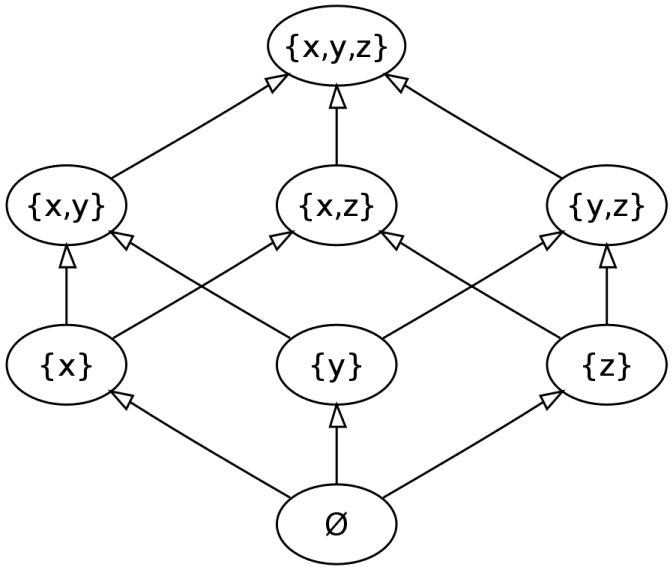
- `public ArrayList(Collection<? extends E> c)`: *Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.*
- `public boolean add(E e)`: *Appends the specified element to the end of this list.*

Weitere Hinweise



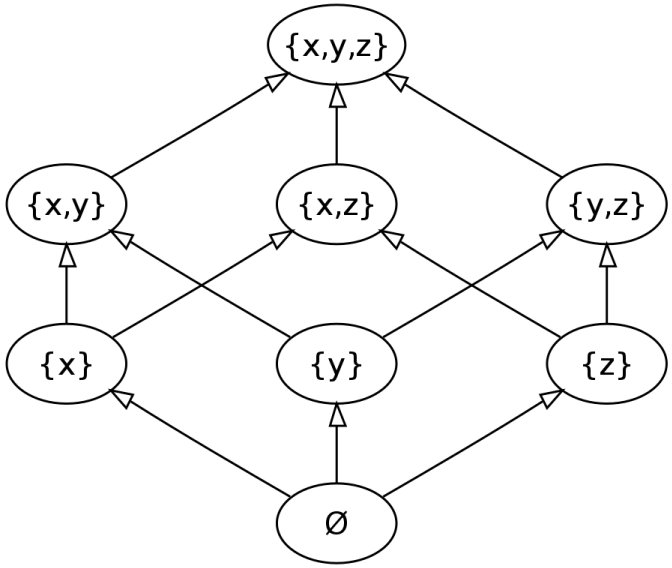
- Üblicherweise sind Potenzmengen für alle allgemeinen Mengen M definiert, hier aber nur für natürliche Zahlen:

Weitere Hinweise



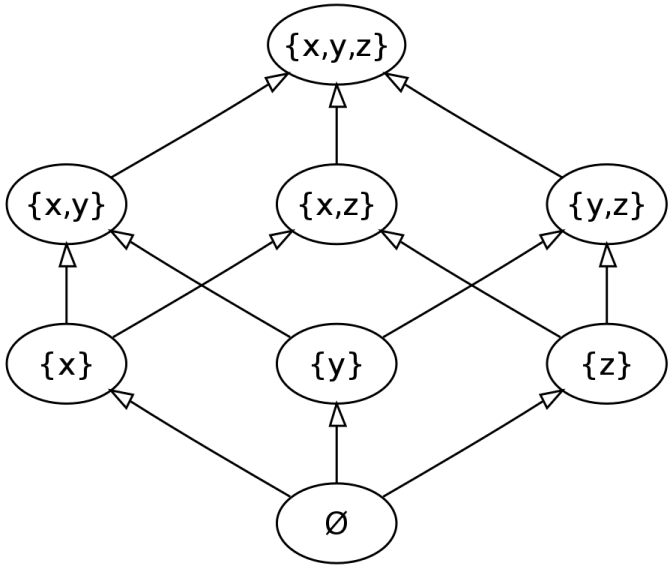
- Üblicherweise sind Potenzmengen für alle allgemeinen Mengen M definiert, hier aber nur für natürliche Zahlen:
  - Hier ist also "eigentlich" $\mathcal{P}(n) := \mathcal{P}(\{n|1 \geq n \geq n\})$ gemeint

Weitere Hinweise



- Üblicherweise sind Potenzmengen für alle allgemeinen Mengen M definiert, hier aber nur für natürliche Zahlen:
  - Hier ist also "eigentlich" $\mathcal{P}(n) := \mathcal{P}(\{n|1 \geq n \geq n\})$ gemeint
  - Da die Definition konkretisiert wurde, werdern wir dieses zu hillfe nehmen können/müssen!

Weitere Hinweise



- Üblicherweise sind Potenzmengen für alle allgemeinen Mengen M definiert, hier aber nur für natürliche Zahlen:
  - Hier ist also "eigentlich" $\mathcal{P}(n) := \mathcal{P}(\{n|1 \geq n \geq n\})$ gemeint
  - Da die Definition konkretisiert wurde, werdern wir dieses zu hillfe nehmen können/müssen!
- Weitere Beispiele:

$$\mathcal{P}(10) = \{\emptyset, \{1\}, \{2\}, \{1,2\}, \ldots,\}$$
$$\mathcal{P}(2) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$$
$$\mathcal{P}(1) = \{\emptyset, \{1\}\}$$
$$\mathcal{P}(0) = \{\emptyset\}$$
$$\mathcal{P}(-1) = ???$$

Vorgegebener Code

```
static List<List<Long>> potenzmenge(long n) {
    // Rueckgabe pm ist Potenzmenge der Zahlen von 1 bis n
    List<List<Long>> pm = new ArrayList<>();

    if (

        // Basisfall



    } else {
        // Rekursion

        List<List<Long>> rek =

        // Ergebnisse zusammenfuehren
        for (List<Long> ohneN : rek) {
            List<Long> mitN = new ArrayList<>(ohneN); // *noch* ohne n




        }
```

Problem: Feld 1

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if ( /* Basisfall */
```

Gesucht ist die Bedinung des Basisfalls:

Problem: Feld 1

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if ( /* Basisfall */
```

Gesucht ist die Bedinung des Basisfalls: Frage: Was ist der "einfachste" Fall?

Problem: Feld 1

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if ( /* Basisfall */
```

Gesucht ist die Bedinung des Basisfalls: Frage: Was ist der "einfachste" Fall?
Erstmal unwichtig: was ist `pm`? Wieso ist es eine `ArrayList`?

Mögliche Lösung (Feld 1)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
```

- Für $n = 0$ haben wir gesehen gilt $\mathcal{P} = \emptyset$.

Mögliche Lösung (Feld 1)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
```

- Für $n = 0$ haben wir gesehen gilt $\mathcal{P} = \emptyset$.
- Wahrscheinlich wird die Rekursion *abrollend* sein, dh, $n \to n-1 \to n-2 \to \ldots \to 1$ bzw 0.

Mögliche Lösung (Feld 1)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
```

- Für $n = 0$ haben wir gesehen gilt $\mathcal{P} = \emptyset$.
- Wahrscheinlich wird die Rekursion *abrollend* sein, dh, $n \to n-1 \to n-2 \to \ldots \to 1$ bzw 0.
- Achtet auf Syntax: Das) { ist notwendig!

Problem: Feld 2

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
```

Gesucht ist der Rückgabewert des Basisfalls.

Problem: Feld 2

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
```

Gesucht ist der Rückgabewert des Basisfalls.
Frage:

- Was ist der Rückgabewert des Einfachsten Falls ($\mathcal{P}(0)$)?

Problem: Feld 2

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
```

Gesucht ist der Rückgabewert des Basisfalls.
Frage:

- Was ist der Rückgabewert des Einfachsten Falls ($\mathcal{P}(0)$)?
- Wie setzen wir dieses in Java um?

Mögliche Lösung (Feld 2)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          }
```

- Wir wissen: $\mathcal{P}(0) = \emptyset$

Mögliche Lösung (Feld 2)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          }
```

- Wir wissen: $\mathcal{P}(0) = \emptyset$
- Wir lesen aus der Aufgabenstellung (+ Partialcode): Wir arbeiten mit `List`'s und `ArrayList`'s.

Mögliche Lösung (Feld 2)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          }
```

- Wir wissen: $\mathcal{P}(0) = \emptyset$
- Wir lesen aus der Aufgabenstellung (+ Partialcode): Wir arbeiten mit `List`'s und `ArrayList`'s.
- `List<List<Long>>` steht für $\mathbb{N}^2$

Mögliche Lösung (Feld 2)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          }
```

- Wir wissen: $\mathcal{P}(0) = \emptyset$
- Wir lesen aus der Aufgabenstellung ($+$ Partialcode): Wir arbeiten mit `List`'s und `ArrayList`'s.
- `List<List<Long>>` steht für $\mathbb{N}^2$

  $\rightarrow$ Dh. $\emptyset$ muss eine leere Liste sein

Mögliche Lösung (Feld 2)

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          }
```

- Wir wissen: $\mathcal{P}(0) = \emptyset$
- Wir lesen aus der Aufgabenstellung (+ Partialcode): Wir arbeiten mit `List`'s und `ArrayList`'s.
- `List<List<Long>>` steht für $\mathbb{N}^2$

  $\rightarrow$ Dh. $\emptyset$ muss eine leere Liste sein z.b.: `pm`.

Problem: Feld 3

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          } else { /* Rekursion */
8                  List<List<Long>> rek =
```

Erster nicht-Trivialfall:

Problem: Feld 3

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          } else { /* Rekursion */
8                  List<List<Long>> rek =
```

Erster nicht-Trivialfall: Es ist nicht zwingend sofort klar was verlangt wird.

Problem: Feld 3

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          } else { /* Rekursion */
8                  List<List<Long>> rek =
```

Erster nicht-Trivialfall: Es ist nicht zwingend sofort klar was verlangt wird. Es ist besser das nächste restliche Programm anzuschauen.

Problem: Feld 3

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          } else { /* Rekursion */
8                  List<List<Long>> rek =
```

Erster nicht-Trivialfall: Es ist nicht zwingend sofort klar was verlangt wird. Es ist besser das nächste restliche Programm anzuschauen.

Wir sehen (unteranderem):

```
9  // Ergebnisse zusammenführen
10 for (List<Long> ohneN : rek) {
```

Problem: Feld 3

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          } else { /* Rekursion */
8                  List<List<Long>> rek =
```

Erster nicht-Trivialfall: Es ist nicht zwingend sofort klar was verlangt wird. Es ist besser das nächste restliche Programm anzuschauen.

Wir sehen (unteranderem):

```
9  // Ergebnisse zusammenführen
10 for (List<Long> ohneN : rek) {
```

Also wir schleifen über `rek` und nennen das Element "`ohneN`"...

Mögliche Lösung (Feld 3)

```
6   List<List<Long>> rek = potenzmenge(n-1);
```

- Die Aufgabe *heißt* "Rekursion", die Variable heißt `rek`.

Mögliche Lösung (Feld 3)

```
6   List<List<Long>> rek = potenzmenge(n-1);
```

- Die Aufgabe *heißt* "Rekursion", die Variable heißt `rek`.
- `ohneN` signalisiert etwas in der Richtung $1, \ldots, n-1$, bzw $n-1$

Mögliche Lösung (Feld 3)

```
6   List<List<Long>> rek = potenzmenge(n-1);
```

- Die Aufgabe *heißt* "Rekursion", die Variable heißt `rek`.
- `ohneN` signalisiert etwas in der Richtung $1, \ldots, n-1$, bzw $n-1$
- Man muss nicht sofort darauf kommen, ggf erst 4. Feld ausprobieren

Problem: Feld 4

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          /* caetera desunt */
12  }
13  /* ... */
14  return pm;
```

Wir haben den Fall $n-1$ und müssen scheinbar irgendwie ein n hinzufügen.

Problem: Feld 4

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          /* caetera desunt */
12  }
13  /* ... */
14  return pm;
```

Wir haben den Fall $n-1$ und müssen scheinbar irgendwie ein n hinzufügen.
Überlegungen Anhand einfachem, nicht-trivialem Fall:

$$\mathcal{P}(2) \setminus \mathcal{P}(1) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\} \setminus \{\emptyset, \{1\}\} = \{\{2\}, \{1,2\}\}$$

Problem: Feld 4

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          /* caetera desunt */
12  }
13  /* ... */
14  return pm;
```

Wir haben den Fall $n-1$ und müssen scheinbar irgendwie ein n hinzufügen.
Überlegungen Anhand einfachem, nicht-trivialem Fall:

$$\mathcal{P}(2) \setminus \mathcal{P}(1) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\} \setminus \{\emptyset, \{1\}\} = \{\{2\}, \{1,2\}\}$$

Wir müssen also $\{2\}$ und $\{1,2\}$ generieren.

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$
  - Wenn wir zu beiden Mengen 2 hinzufügen, bekommen wir $\{\{2\}, \{1, 2\}\}$

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$
  - Wenn wir zu beiden Mengen 2 hinzufügen, bekommen wir $\{\{2\}, \{1, 2\}\} = \mathcal{P}(2)$

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$
  - Wenn wir zu beiden Mengen 2 hinzufügen, bekommen wir $\{\{2\}, \{1, 2\}\} = \mathcal{P}(2)$
- `mitN` muss erstellt werden, da `List`'s Referenzattribute sind, `ohneN.add(...)` würde auch `mitN` ändern

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$
  - Wenn wir zu beiden Mengen 2 hinzufügen, bekommen wir $\{\{2\}, \{1, 2\}\} = \mathcal{P}(2)$
- `mitN` muss erstellt werden, da `List`'s Referenzattribute sind, `ohneN.add(...)` würde auch `mitN` ändern
- beide Attribute werden in das `pm`, für diese Rekursionsebene hinzugefügt

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$
  - Wenn wir zu beiden Mengen 2 hinzufügen, bekommen wir $\{\{2\}, \{1, 2\}\} = \mathcal{P}(2)$
- `mitN` muss erstellt werden, da `List`'s Referenzattribute sind, `ohneN.add(...)` würde auch `mitN` ändern
- beide Attribute werden in das `pm`, für diese Rekursionsebene hinzugefügt
  - Wir müssen auch jedes mal `ohneN` hinzufügen, da diese ansonsten nicht enthalten sein werden

Mögliche Lösung (Feld 4)

```
8   for (List<Long> ohneN : rek) {
9           Long<Long> mitN = new ArrayList<>(ohneN);
10          // *noch* ohne n
11          mitN.add(n);
12          pm.add(mitN);
13          pm.add(ohneN);
14  }
15  /* ... */
16  return pm;
```

- Am Beispiel von vorhin: $\mathcal{P}(1) = \{\emptyset, \{1\}\}$
  - Wenn wir zu beiden Mengen 2 hinzufügen, bekommen wir $\{\{2\}, \{1, 2\}\} = \mathcal{P}(2)$
- `mitN` muss erstellt werden, da `List`'s Referenzattribute sind, `ohneN.add(...)` würde auch `mitN` ändern
- beide Attribute werden in das `pm`, für diese Rekursionsebene hinzugefügt
  - Wir müssen auch jedes mal `ohneN` hinzufügen, da diese ansonsten nicht enthalten sein werden
  - `rek` kann nicht (elegant) zu `pm` mittels `.addAll` hinzugefügt werden.

Lösungsvorschlag, alles auf Einmal

```
1  static List<List<Long>> potenzmenge(long n) {
2          // Rueckgabe pm ist Potenzmenge der
3          // Zahlen von 1 bis n
4          List<List<Long>> pm = new ArrayList<>();
5          if (n == 0) {
6                  return pm;
7          } else { /* Rekursion */
8                  List<List<Long>> rek = potenzmenge(n-1);
9                  // Ergebnisse zusammenfuehren
10                 for (List<Long> ohneN : rek) {
11                         Long<Long> mitN = new ArrayList<>(ohneN)
12                         // *noch* ohne n
13                         mitN.add(n);
14                         pm.add(mitN);
15                         pm.add(ohneN);
16                 }
17         }
18         return pm;
19 }
```