

Mathematik für Informatiker

Kombinatorik, Stochastik und Statistik

Übungsblatt 4

Tom Paßberg , Iain Dorsch

Aufgabe 1

a)

Der Professor versucht die Menge der Partitionen einer 5 elementigen Menge in 3 Teilen aufzuschreiben. Er hat bereits die Menge der Partitionen einer 4 elementigen Menge in 3 Teilen notiert.

c)

Es gilt $S(3, 1) = S(2, 1) = S(2, 2) = S(3, 3) = 1$.

$$S(3, 2) = S(2, 1) + 2 \cdot S(2, 2) = 1 + 2 \cdot 1 = 3$$

$$S(4, 2) = S(3, 1) + 2 \cdot S(3, 2) = 1 + 2 \cdot 3 = 7$$

$$S(4, 3) = S(3, 2) + 3 \cdot S(3, 3) = 3 + 3 \cdot 1 = 6$$

$$S(5, 3) = S(4, 2) + 3 \cdot S(4, 3) = 7 + 3 \cdot 6 = 25$$

Aufgabe 2

Algorithmus zur Berechnung der Partitionen einer Menge in Rust:

```
1 fn partitionen(input: Vec<u8>) -> Vec<Vec<Vec<u8>>> {
2     if input.is_empty() {
3         return vec![vec![]];
4     }
5     (0..input.len() as u8).flat_map(|i| {
6         teilmengen(input.iter().cloned().skip(1).collect(), i)
7             .into_iter()
8             .flat_map(|subset| {
9                 let rest: Vec<u8> = input.iter().cloned().filter(|x| !
10                    subset.contains(x)).collect();
11                 partitionen(subset)
12                    .into_iter()
13                    .map(|mut partition| { partition.push(rest.clone());
14                        partition })
15                    .collect::
```

Erklärung:

- **2..5** : Basisfall für $n = 0$
- **5** : Schleife über alle $i = 0 \dots n$ analog zur Summe in der Formel für die Bellschen Zahlen
- **6** : Erzeugt alle Teilmengen der Eingabe ohne das erste Element. $\binom{n}{i}$ Teilmengen

- **7..15** : Rekursiver Funktionsaufruf für alle erzeugten Teilmengen. Das Ergebnis ergibt sich aus der Partition der Teilmengen vereinigt mit der Restmenge. Die Restmenge sind alle Elemente die nicht in der Teilmengen enthalten sind.

Algorithmus zur Berechnung der n elementige Teilmengen in Rust:

```

1 fn teilmengen(input: Vec<u8>, n: u8) -> Vec<Vec<u8>> {
2     if n == 0 {
3         return vec![vec![]];
4     }
5     if input.is_empty() {
6         return vec![];
7     }
8     let rest = input[1..].to_vec();
9     teilmengen(rest.clone(), n - 1)
10    .into_iter()
11    .map(|mut subset| {subset.push(input[0]); subset})
12    .chain(teilmengen(rest, n).into_iter())
13    .collect()
14 }
```

Funktionsaufruf:

```

1 fn main() {
2     for partition in partitionen((1..=4).collect()) {
3         println!("{}", partition);
4     }
5 }
```

Ausgabe:

```

1 [[1, 2, 3, 4]]
2 [[2], [1, 3, 4]]
3 [[3], [1, 2, 4]]
4 [[4], [1, 2, 3]]
5 [[3, 2], [1, 4]]
6 [[2], [3], [1, 4]]
7 [[4, 2], [1, 3]]
8 [[2], [4], [1, 3]]
9 [[4, 3], [1, 2]]
10 [[3], [4], [1, 2]]
11 [[4, 3, 2], [1]]
12 [[3], [4, 2], [1]]
13 [[2], [4, 3], [1]]
14 [[2, 3], [4], [1]]
15 [[3], [2], [4], [1]]
```

Aufgabe 3

a)

Die Anzahl der Möglichen Endergebnisse ist $3^3 = 27$. Die Anzahl der möglichen Totalordnungen ist $3! = 6$. Die Anzahl der Ergebnisse bei denen der Spieler verliert ist

$3! + 2 = 8$ da der Spieler auch verliert wenn alle Kanten des Dreiecks einen Spielstein der selber Ausrichtung enthalten. Die Wahrscheinlichkeit das der Spieler verliert ist also $\frac{8}{27} = 29,6\%$.

Gewinnswahrscheinlichkeit: $\frac{19}{27} = 70,4\%$.

b)

Halbordnungen auf $\{1, 2, 3\}$

$$R_1 = \{(1, 1), (2, 2), (3, 3)\}$$

$$R_2 = \{(1, 1), (2, 2), (3, 3), (1, 2)\}$$

$$R_3 = \{(1, 1), (2, 2), (3, 3), (1, 3)\}$$

$$R_4 = \{(1, 1), (2, 2), (3, 3), (2, 3)\}$$

$$R_5 = \{(1, 1), (2, 2), (3, 3), (2, 1)\}$$

$$R_6 = \{(1, 1), (2, 2), (3, 3), (3, 1)\}$$

$$R_7 = \{(1, 1), (2, 2), (3, 3), (3, 2)\}$$

$$R_8 = \{(1, 1), (2, 2), (3, 3), (1, 2), (2, 3), (1, 3)\}$$

$$R_9 = \{(1, 1), (2, 2), (3, 3), (3, 1), (1, 2), (3, 2)\}$$

$$R_{10} = \{(1, 1), (2, 2), (3, 3), (1, 3), (3, 2), (1, 2)\}$$

$$R_{11} = \{(1, 1), (2, 2), (3, 3), (2, 1), (1, 3), (2, 3)\}$$

$$R_{12} = \{(1, 1), (2, 2), (3, 3), (2, 3), (3, 1), (2, 1)\}$$

$$R_{13} = \{(1, 1), (2, 2), (3, 3), (3, 2), (2, 1), (3, 1)\}$$

R_8 bis R_{13} sind Totalordnungen.

Aufgabe 4

a)

$$R = \{(1, 1), (2, 2), (1, 2)\} R = \{(1, 1), (2, 2)\}$$

c)

Die Anzahl der Totalordnungen auf einer n elementigen Menge ist $n!$. Das ergibt sich daraus, dass es $n!$ mögliche Reihenfolgen gibt eine n elementige Menge anzuordnen. Eine Totalordnung weist allen Elementen eine eindeutige Reihenfolgen zu.

Aufgabe 5

Code:

```

1 fn partitions(n: u8, m: u8) -> Vec<Vec<Vec<u8>>> {
2     if m == 0 || n == 0 {
3         return Vec::new();
4     }
5     if m == 1 {
6         return vec![vec![(1..=n).collect()]];
7     }
8
9     let mut result = Vec::new();
10    for p in partitions(n - 1, m - 1).iter_mut() {
11        p.push(vec![n]);
12        result.push(p.clone());
13    }
14    for p in partitions(n - 1, m) {
15        for i in 0..p.len() {
16            let mut p = p.clone();
17            p[i].push(n);
18            result.push(p);
19        }
20    }
21    result
22 }

```

Funktionsaufruf:

```

1 let parts = partitions(5, 3);
2
3 for (i, part) in parts.iter().enumerate() {
4     println!("{:3}: {:?}", i, part);
5 }

```

Ausgabe:

```

1      0: [[1, 2, 3], [4], [5]]
2      1: [[1, 2, 4], [3], [5]]
3      2: [[1, 2], [3, 4], [5]]
4      3: [[1, 3, 4], [2], [5]]
5      4: [[1, 3], [2, 4], [5]]
6      5: [[1, 4], [2, 3], [5]]
7      6: [[1], [2, 3, 4], [5]]
8      7: [[1, 2, 5], [3], [4]]
9      8: [[1, 2], [3, 5], [4]]
10     9: [[1, 2], [3], [4, 5]]
11    10: [[1, 3, 5], [2], [4]]
12    11: [[1, 3], [2, 5], [4]]
13    12: [[1, 3], [2], [4, 5]]
14    13: [[1, 5], [2, 3], [4]]
15    14: [[1], [2, 3, 5], [4]]
16    15: [[1], [2, 3], [4, 5]]
17    16: [[1, 4, 5], [2], [3]]
18    17: [[1, 4], [2, 5], [3]]
19    18: [[1, 4], [2], [3, 5]]
20    19: [[1, 5], [2, 4], [3]]
21    20: [[1], [2, 4, 5], [3]]
22    21: [[1], [2, 4], [3, 5]]

```

23	22:	$[[1, 5], [2], [3, 4]]$
24	23:	$[[1], [2, 5], [3, 4]]$
25	24:	$[[1], [2], [3, 4, 5]]$