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. Er kann nach der Struktur der Rekursionsformel für die Bellschen Zahlen die Menge der Partitionen einer 5 elementigen Menge in 3 Teilen berechnen, indem er sein bisheriges Ergebis weiterverwendet. Er muss nun allen Partitionen der 4 elementigen Menge in 3 Teilen ein neues Element an jeder möglichen Stelle hinzufügen.

c)

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

$$
\begin{aligned}
S(3,2) &= S(2,1) + 2 \cdot S(2,2) = 1 + 2 \cdot 1 = 3 \\
S(4,2) &= S(3,1) + 2 \cdot S(3,2) = 1 + 2 \cdot 3 = 7 \\
S(4,3) &= S(3,2) + 3 \cdot S(3,3) = 3 + 3 \cdot 1 = 6 \\
S(5,3) &= S(4,2) + 3 \cdot S(4,3) = 7 + 3 \cdot 6 = 25
\end{aligned}
$$

Aufgabe 2

Algoritmus 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| !
                       subset.contains(x)).collect();
10                 partitionen(subset)
11                     .into_iter()
12                     .map(|mut partition| { partition.push(rest.clone());
                           partition })
13                     .collect::<Vec<_>>()
14             })
15     }).collect()
16 }
```

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 Ergebiss ergibt sich aus der Partition der Teilmengen vereinigt mit der Restmenge. Die Restmenge sind alle Elemente die nicht in der Teilmengen enthalten sind.

Algoritmus 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 Endergenbnisse ist $3^3 = 27$. Die Anzahl der möglichen Totalordnungen ist $3! = 6$. Die Anzahl der Ergebisse 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 $\frac{8}{27} = 29,6\%$.
Gewinnwarscheinlichkeit: $\frac{19}{27} = 70,4\%$.

b)

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

$$
\begin{aligned}
R_1 &= \{(1,1),(2,2),(3,3)\} \\
R_2 &= \{(1,1),(2,2),(3,3),(1,2)\} \\
R_3 &= \{(1,1),(2,2),(3,3),(1,3)\} \\
R_4 &= \{(1,1),(2,2),(3,3),(2,3)\} \\
R_5 &= \{(1,1),(2,2),(3,3),(2,1)\} \\
R_6 &= \{(1,1),(2,2),(3,3),(3,1)\} \\
R_7 &= \{(1,1),(2,2),(3,3),(3,2)\} \\
R_8 &= \{(1,1),(2,2),(3,3),(1,2),(2,3),(1,3)\} \\
R_9 &= \{(1,1),(2,2),(3,3),(3,1),(1,2),(3,2)\} \\
R_{10} &= \{(1,1),(2,2),(3,3),(1,3),(3,2),(1,2)\} \\
R_{11} &= \{(1,1),(2,2),(3,3),(2,1),(1,3),(2,3)\} \\
R_{12} &= \{(1,1),(2,2),(3,3),(2,3),(3,1),(2,1)\} \\
R_{13} &= \{(1,1),(2,2),(3,3),(3,2),(2,1),(3,1)\}
\end{aligned}
$$

R_8 bis R_{13} sind Totalordnungen.

Aufgabe 4

a)

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

b)

$M \times M$ Relationen auf einer Menge M mit $|M| = n$ kann man als eine Tabelle darstellen, in welcher man jede Abbildung von Element zu Element mit einem Kreuz kennzeichnet. Die Tabelle hätte die größe $n \cdot n$. Die Anzahl aller möglichen relationen entspricht demnach der Möglichen Belegungen der Tabelle 2^n. Eine Reflexive Relation muss in einer solchen Tabelle immer eine diagonale der länge n enthalten, da alle Elemte auf sich selbst abbilden müssen, demnach entfallen n möglichkeiten aus der tabelle, also $n \cdot n - n = n(n-1)$ demnach gibt es $2^{n(n-1)}$ reflexive relationen auf der menge M.

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 weißt allen Elementen eine eindeutige Reihenfolgen zu.

Aufgabe 5

Code:

```
fn partitions(n: u8, m: u8) -> Vec<Vec<Vec<u8>>> {
    if m == 0 || n == 0 {
        return Vec::new();
    }
    if m == 1 {
        return vec![vec![(1..=n).collect()]];
    }

    let mut result = Vec::new();
    for p in partitions(n - 1, m - 1).iter_mut() {
        p.push(vec![n]);
        result.push(p.clone());
    }
    for p in partitions(n-1, m) {
        for i in 0..p.len() {
            let mut p = p.clone();
            p[i].push(n);
            result.push(p);
        }
    }
    result
}
```

Funktionsaufruf:

```
let parts = partitions(5, 3);

for part in parts.iter() {
    println!("{:?}", part);
}
```

Ausgabe:

```
[[1, 2, 3], [4], [5]]
[[1, 2, 4], [3], [5]]
[[1, 2], [3, 4], [5]]
[[1, 3, 4], [2], [5]]
[[1, 3], [2, 4], [5]]
[[1, 4], [2, 3], [5]]
[[1], [2, 3, 4], [5]]
[[1, 2, 5], [3], [4]]
[[1, 2], [3, 5], [4]]
[[1, 2], [3], [4, 5]]
[[1, 3, 5], [2], [4]]
[[1, 3], [2, 5], [4]]
[[1, 3], [2], [4, 5]]
[[1, 5], [2, 3], [4]]
[[1], [2, 3, 5], [4]]
[[1], [2, 3], [4, 5]]
[[1, 4, 5], [2], [3]]
[[1, 4], [2, 5], [3]]
```

```
19  [[1 ,  4] ,  [2] ,  [3 ,  5]]
20  [[1 ,  5] ,  [2 ,  4] ,  [3]]
21  [[1] ,  [2 ,  4 ,  5] ,  [3]]
22  [[1] ,  [2 ,  4] ,  [3 ,  5]]
23  [[1 ,  5] ,  [2] ,  [3 ,  4]]
24  [[1] ,  [2 ,  5] ,  [3 ,  4]]
25  [[1] ,  [2] ,  [3 ,  4 ,  5]]
```